UNIVERSITÉ PARIS SUD

ECOLE DOCTORALE D'INFORMATIQUE PARIS-SUD

LABORATOIRE: INRIA, SACLAY, ILE DE FRANCE

DISCIPLINE: INFORMATIQUE

THÈSE DE DOCTORAT

Soutenue le 9 Mai 2014 par

BENJAMIN BACH

CONNECTIONS, CHANGES, AND CUBES: UNFOLDING DYNAMIC NETWORKS FOR VISUAL EXPLORATION

Directeur de thèse :	M. Jean-Daniel Fekete	Directeur de Recherche (INRIA Saclay, France)
Co-directeur de thèse :	M. Emmanuel Pietriga	Chargé de Recherche (INRIA Saclay, France, and INRIA Chile, Chile)

Composition du jury :

Rapporteurs :	M. Jarke J. van Wijk	Professeur (Eindhoven University of Technology, Pays-Bas)
	M. Tim Dwyer	Maitre de Conférence avec habilitation (*Senior Lecturer*) (Monash University, Australie)
Examinateurs :	Mme. Silvia Miksch	Professeur (Vienna University of Technology, Autriche)
	M. Guy Melançon	Professeur (Université Bordeaux, France)
	Mme. Chantal Reynaud	Professeur (Laboratoire de Recherche en Informatique, Université Paris Sud, France)

Abstract

NETWORKS are models that help us understanding and thinking about relationships between entities in the real world. Many of these networks are *dynamic*, i.e. connectivity changes over time. Understanding changes in connectivity means to understand interactions between elements of complex systems; how people create and break up friendship relations, how signals get passed in the brain, how business collaborations evolve, or how food-webs restructure after environmental changes. However, understanding static networks is already difficult, due to size, density, attributes and particular motifs; changes over time very much increase this complexity. Quantification of change is often insufficient, but beyond an analysis that is driven by technology and algorithms, humans display a unique capability of understanding and interpreting information in data, based on vision and cognition.

This dissertation explores ways to interactively explore dynamic networks by means of visualization. I develop and evaluate techniques to unfold the complexity of dynamic networks, making them understandable by looking at them from different angles, decomposing them into their parts and relating the parts in novel ways. While most techniques for dynamic network visualization rely on one particular type of *view* on the data, complementary visualizations allow for higher-level exploration and analysis.

Covering three aspects *Tasks*, *Visualization Design* and *Evaluation*, I develop and evaluate the following unfolding techniques: (i) temporal navigation between individual time steps of a network and improved animated transitions to better understand changes, (ii) designs for the comparison of weighted graphs, (iii) the Matrix Cube, a space-time cube based on adjacency matrices, allowing to visualize dense dynamic networks, as well as GraphCuisine, a system to (iv) generate synthetic networks with the primary focus on evaluating visualizations in user studies. In order to inform the design and evaluation of visualizations, we (v) provide a task taxonomy capturing users' tasks when exploring dynamic networks.

Finally, (vi) the idea of unfolding networks with Matrix Cubes is generalized to other data sets that can be represented in space-time cubes (videos, geographical data, etc.). Visualizations in these domains can inspire visualizations for dynamic networks, and vice-versa. We propose a taxonomy of *operations*, describing how 3D space-time cubes are decomposed into a large variety of 2D visualizations. These operations help us exploring the design space for visualizing and interactively unfolding dynamic networks and other spatio-temporal data, and may serve users as a mental model of the data.

Résumé

LES réseaux sont des modèles qui nous permettent de comprendre les relations entre éléments du monde réel. Une grande quantité de réseaux sont dynamiques, c'est-a-dire que leur connexité change au cours du temps. Comprendre les changements de connexité signifie comprendre les interactions entre les éléments de systèmes complexes: comment se forment les relations sociales et commerciales, comment sont transmis les signaux entre les régions du cerveau, comment s'organisent les réseaux trophiques après des catastrophes environnementales. Au-delà de ce que nous permet la technologie et les algorithmes d'analyses, l'homme dispose d'une capacité unique pour comprendre et interpréter des informations : la vision et la cognition.

Cette thèse développe et examine des moyens pour explorer les réseaux dynamiques d'une manière interactive et visuelle. Je propose des techniques pour *déplier* la complexité des réseaux, avec le but de les rendre compréhensibles, de les voir à partir de perspectives différentes, d'examiner leurs composantes. Déplier des réseaux est une métaphore, comme la création des cartes bidimensionelles d'objects tridimensionnels comme la Terre : chaque méthode de projection a comme résultat une carte différente qui permet de voir des relations différentes entre la taille des continents et des océans, des distances, etc.

Je propose les techniques de dépliage suivantes, implementées et évaluées dans des systèmes interactifs : (i) une navigation temporelle qui permet de naviguer plus efficacement entre des différents instants, ainsi qu'un feedback visuel qui permet de mieux comprendre les changements dans la réseaux entre deux instants arbitraires. (ii) Des designs permettant la comparaison directe de deux réseaux avec des liens pondérés. (iii) Un modèle de visualisation pour des réseaux denses avec des liens pondérés, ainsi que (iv) la génération de réseaux synthétiques utilisés pour l'évaluation des visualisations. Afin de mieux créer et évaluer des visualisations, nous (v) proposons une taxonomie de tâche pour décrire des tâches accomplies par des analystes des réseaux.

Pour compléter, (vi) nous généralisons l'idée de *dépliage* pour décrire d'autres genres de données temporelles, représentable dans des cubes espace-temps. Cela concerne la visualization de vidéos, des données multi-variées, ainsi que la géographique. Une telle généralisation a pour but de fournir une base commune pour échanger des techniques de visualisation et de mieux comprendre l'espace de design pour les réseaux dynamiques. Dans cette optique, nous proposons une taxonomie d'opérations génériques qui nous permet de transformer un cube espace-temps en visualisation bidimensionelle, ainsi qu'une description des formes évoquées par les données dans le cube espace-temps.

Remerciements

First, I would like to thank my PhD advisors Jean-Daniel Fekete and Emmanuel Pietriga for their inspiration, their critique, time for discussions, permanent availability, and wise patience, during the entire period of my PhD research. I could never have imagined any better super-visors. Further, I want to thank Emmanuel Pietriga and Fanny Chevalier for encouraging me to pursue a PhD research.

I very thank the members of my jury; Jarke J. van Wijk, Tim Dwyer, Silvia Miksch, Guy Melançon, and Chantal Reynaud, for their critique, their feedback, and for eventually coming to Paris to attend my defense.

I would further like thank all of my collaborators with whom I had the pleasure to do research during the past three years, for their discussions and opportunities to learn from such nice and experienced people (alphabetical order): Basak Alper, Daniel Archambault, Sheelagh Carpendale, Pierre Dragicevic, Nathalie Henry-Riche, Christophe Hurter, Tobias Isenberg, Evelyne Lutton, and André Spritzer. I am looking forward to continue our collaborations.

I want to thank all the people from the both research groups in Orsay/Saclay; Aviz and InSitu, for their general support and the nice and friendly atmosphere they created, turning the lab into a special place; Wendy Mackay and Michel Beaudon-Lafon, Pierre Dragicevic, Wes Willet, Lora Oehlberg, Petra and Tobias Isenberg, Stephane Huot, Oliver Chapuis, Romain Vuillemot, Mathieu Nancel, Julie Wager, Yvonne Jansen, Charles Perin, Jeremy Garcia, Samuel Huron, Nicolas Hulot, Romain di Vozzo, Jeremy Boy and many more.

There have further been numerous people who impressed and inspired me and my research in one way or the other. Any explicit list would remain incomplete, though.

Finally, but not less important; I would not be what I am today without my parents care. Yet, there is one very special person, one who is certainly even happier than me that this work eventually found its way to digits and print; my Love, Annekathrin.

Paris, May 2014

Publications

Parts of this dissertation work have been published in international conferences and journals, as full paper and/or poster presentations. This section lists all articles published during my PhD research. Chapters in this thesis are indicated in brackets.

Papers at International Journals and Conferences

1. **Benjamin Bach**, Pierre Dragicevic, Daniel Archambault, Christophe Hurter, Sheelagh Carpendale, *A Review of Temporal Data Visualizations Based on Space-Time Cube Operations*, EuroVis, State-of-the-Art-Report (STAR),

   to appear. (Chapter 7)

2. Evelyne Lutton, Hugo Gilbert, Waldo Cancino, **Benjamin Bach**, Pierre Collet, *GridVis: Visualisation of Island-Based Parallel Genetic Algorithms*, EvoApps 2014, Granada, Spain, to appear.

3. **Benjamin Bach**, Emmanuel Pietriga, Jean-Daniel Fekete, *Visualizing Dynamic Networks with Matrix Cubes*, In Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems, Toronto, Canada, 2014, to appear.

   (Chapter 6)

4. **Benjamin Bach**, Emmanuel Pietriga, Ilaria Liccardi, *Visualizing Populated Ontologies with Ontotrix*, International Journal of Semantic Web and Information Systems, 2014, to appear *Note: This project has not been included in this dissertation because it does not address problems related to dynamic networks. The work was done as an intern, prior to starting my dissertation research.*

5. **Benjamin Bach**, Emmanuel Pietriga, Jean-Daniel Fekete, *GraphDiaries: Animated Transitions and Temporal Navigation for Dynamic Networks*, IEEE Transactions on Visualization and Computer Graphics PP (99), 2013.

   (Chapter 4)

6. Basak Alper, **Benjamin Bach**, Nathalie Henry Riche, Tobias Isenberg, Jean-Daniel Fekete, *Weighted Graph Comparison Techniques for Brain Connectivity Analysis*, In Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems, Paris, France, 2013, pages 483–492, (Best Paper).

   (Chapter 5)

7. **Benjamin Bach**, Andre Spritzer, Evelyne Lutton, Jean-Daniel Fekete, *Interactive Random Graph Generation with Evolutionary Algorithms*, Proceedings of Graph Drawing, Seattle, WA, 2012, Pages 177-180, Springer.

   (Chapter A)

Posters at International Conferences

8. **Benjamin Bach**, Emmanuel Pietriga, Jean-Daniel Fekete, *Visualizing Dense Dynamic Networks with Matrix Cubes*, IEEE Information Visualization Conference Poster Session, Atlanta, GA, October 2013.

9. **Benjamin Bach**, Pierre Dragicevic, Samuel Huron, Petra Isenberg, Yvonne Jansen, Charles Perin, Andre Spritzer, Romain Vuillemot, Wesley Willett, and Tobias Isenberg, *Illustrative Data Graphics in 18th–19th Century Style: A Case Study*, IEEE Information Visualization Conference Poster Session, Atlanta, GA, October 2013.

10. Pierre Dragicevic, **Benjamin Bach**, Nicole Dufournaud, Samuel Huron, Petra Isenberg, Yvonne Jansen, Charles Perin, Andre Spritzer, Romain Vuillemot, Wesley Willett, and Tobias Isenberg, *Visual Showcase: An Illustrative Data Graphic in an 18th–19th Century Style*., In Visual Showcase at the Joint ACM/EG Symposium on Computational Aesthetics, Sketch-Based Interfaces and Modeling, and Non-Photorealistic Animation and Rendering (Expressive 2013, July 19–20, Anaheim, CA, USA). 2013.

11. **Benjamin Bach**, Emmanuel Pietriga, Jean-Daniel Fekete, *Temporal Navigation in Dynamic Networks*, IEEE Information Visualization Conference Poster Session, Seattle, WA, October 2012.

12. **Benjamin Bach**, Dietrich Kammer, Jan Polowinski; *Facettice: Integrating Faceted Navigation and Concept Lattices for Visual Data Exploration*. IEEE Information Visualization Conference Poster Session, Seattle, WA, October 2012.

13. **Benjamin Bach**, Emmanuel Pietriga, Ilaria Liccardi, G. Legostaev, *OntoTrix: A Hybrid Visualization for Populated Ontologies*, Proceedings of the 20th international conference companion on World Wide Web, Pages 177-180, Hyderabad, India, March 2011.

14. **Benjamin Bach**, G. Legostaev, Emmanuel Pietriga, *Visualizing Populated Ontologies with OntoTrix*, Proceedings of the International Conference on Semantic Web 2010 Posters and Demonstrations Track, Collected Abstracts, pages 85-88, Shanghai, China, November 2010.

Contents

1. Introduction

"Historians of science [...] glorify great thinkers and their conceptual breakthroughs. Less heralded are the inventors of scientific instruments [...]"
Seoung, 2012, p. 138

This dissertation is about creating scientific instruments: interactive visualizations for the exploration of dynamic networks.

1.1. Background and Motivation

NETWORKS are models that help us understand and reason about relationships between entities in the real world. As such, networks appear in a variety of domains; sociologists investigate relationships, and collaboration between people; historians and geographers analyze migration streams and trading routes; enterprises monitor computer networks, analyze traffic load and watch out for fraud; biologists research reaction networks and interactions between proteins; epidemiologists investigate the spreading of viruses and neurologists investigate functional correlations between regions in the brain, in order to detect the patterns of an individual activity or external stimuli, as well as the impact of drugs and diseases on the brain's functionality.

Networks in their general form are used where they help in answering questions about the *connectivity* of *nodes* (also: *vertices, actors*); Which two nodes are *related*? Are two nodes reachable via a *path*? Where are the *clusters* in the network? Is there *structure* in the network? How *vulnerable* is the network? How *dense* is it? While these questions imply a **static** network model, there are many questions that require a **dynamic network** model; Where does the network *grow fastest*? Which nodes *behave* differently? *In which order* do nodes connect, how do clusters *exchange* nodes and *evolve*? Is the network *stable* or *unstable*? Is there a *trend*? Which nodes do *not follow* the trend? When do major changes *impact* the network?

While static networks describe topology and structure, *dynamic networks*[1] describe changes, interactions, behavior and evolution on its elements. Dynamic networks allow the exploration of rules within a system and how its individual parts behave. They

[1] In this dissertation I use the term *Dynamic Network* to describe networks with a temporal dimension, as opposed to *static networks*. See Section 2.3.

represent the impact of environmental factors on ecosystems and organisms, as well as restructuring mechanisms thereafter. They describe how friendship relations emerge and break up, how signals are exchanges between regions in the brain, and many others (see e.g., Holme & Saramäki, 2012 and Sporns, 2011).

Many of these systems exhibit similar patterns. Networks *grow* and *shrink*, connections *break* and *get established*, nodes *enter communities*, clusters *merge* and networks *divide into disconnected components*. Observations in these networks are made over time, i.e. *events* at a particular time point as well as the *order* of events. *Changes* are described between time points as well as over periods of time, including the evolution of statistical measures. Low-level events involving a single node or a pair of nodes (*dyads*) such as the appearance or disappearance of a node or a *connection* (also: *link, edge*) can be responsible for higher-level changes on clusters, paths, as well as the entire network. Similar to static networks, dynamic networks can be investigated on different topological *scales* (nodes, dyads, clusters, etc.). Furthermore, there are different *temporal* scales; individual events on a short time scale, or changes on a higher structural level that happen over a longer time period, and which provoke patterns of *repetition*, *instability*, or *convergence*.

While investigating static networks is already complex, dynamic networks combine two types of data: networks and time-varying (or *temporal*) data. Static networks get complex as the number of nodes, edges and *density*, increases, because this complicates the search for, and the exploration of, paths, individual connections, and clusters. Dynamic networks increase in complexity with the number of time steps, the amount of change between individual time steps, the lack of clear trends, as well as the presence of attributes varying over time.

Making sense of, and understanding such complex data, generally relies on two complementary methods: *analysis*, based on measures, and *observation*, based visualization.

Exploratory Data Analysis

Networks can be analyzed using basic measurements such as the number of nodes, density, or clustering coefficient. Such network measures can indicate the importance of a node or how stable a network is to node removal. Often, an analysis requires an aggregation of information across multiple measures, but the more measures are involved in an analysis, the more complex it becomes. In fact, numerical analysis appropriate for hypothesis testing, is also called *confirmative* analysis. The opposite, *Exploratory Data Analysis (EDA)* (Tukey, 1993), is described as a type of analysis without a particular hypothesis in mind. EDA is about formulating hypotheses about the data, which therein can be confirmed or rejected by confirmative analysis. Confirmatory and exploratory analysis are hence complementary.

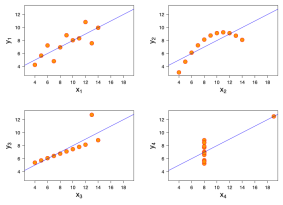
Data set	1-3	1	2	3	4	4
Variable	x	y	y	y	x	y
Obs. no. 1 :	10.0	8.04	9.14	7.46 :	8.0	6.58
2 :	8.0	6.95	8.14	6.77 :	8.0	5.76
3 :	13.0	7.58	8.74	12.74 :	8.0	7.71
4 :	9.0	8.81	8.77	7.11 :	8.0	8.84
5 :	11.0	8.33	9.26	7.81 :	8.0	8.47
6 :	14.0	9.96	8.10	8.84 :	8.0	7.04
7 :	6.0	7.24	6.13	6.08 :	8.0	5.25
8 :	4.0	4.26	3.10	5.39 :	19.0	12.50
9 :	12.0	10.84	9.13	8.15 :	8.0	5.56
10 :	7.0	4.82	7.26	6.42 :	8.0	7.91
11 :	5.0	5.68	4.74	5.73 :	8.0	6.89

TABLE. Four data sets, each comprising 11 (x, y) pairs.

(a)

Number of observations (n) = 11
Mean of the x's ($\bar{x}$) = 9.0
Mean of the y's ($\bar{y}$) = 7.5
Regression coefficient (b_1) of y on x = 0.5
Equation of regression line: $y = 3 + 0.5\,x$
Sum of squares of $x - \bar{x}$ = 110.0
Regression sum of squares = 27.50 (1 d.f.)
Residual sum of squares of y = 13.75 (9 d.f.)
Estimated standard error of b_1 = 0.118
Multiple R^2 = 0.667

(b)



(c)

Figure 1.1.: *Anscombe's quartet (Anscombe, 1973). (a) Values in four different data samples (1-4), (b) statistical measures are the same across all four samples, (c) scatter plots showing actual differences in the four samples.*

A proper analysis requires multiple tools including powerful computers, fast algorithms and clever analysis methods. Yet, beyond what can be achieved by technology and engineering, humans display a unique capability of understanding and interpreting information, based on vision and cognition. EDA therefore emphasizes *"seeing results – graphically, or numerically"*[2] and *"stripping off layer after layer of what could be described."*[3] An example of the importance of *seeing* the data, in contrast to obtaining the statistical numbers, is given by Anscombe (1973), called *Anscombe's quartet* (Figure 1.1). Anscombe's quartet is made of four data samples (columns in Figure 1.1(a)), each containing the same number of observations (rows). Yet, each sample contains a different distribution of observations. However, all sets have exactly the same values for various statistical measures (Figure 1.1(b)). Eventually, showing the data as points in a scatterplot reveals obvious differences (Figure 1.1(c)). As further illustrated in Chapter A, networks with equal measures can have quite different topologies.

Information Visualization

Vision is a powerful tool to capture information, because its processing is highly parallel, other than reading numbers or listening to language. This parallelism allows us to *see things in their context*, and detecting relationships pre-attentively without having to infer them cognitively. However, providing effective visual communication requires interdisciplinary knowledge, proper methodologies, effective ways of evaluation, as well as algorithms to organize glyphs on the screen and interactively explore the data.

Developing means for an effective visual communication of information is called *Information Visualization*[4]. The process of information visualization can be described through the InfoVis pipeline (Card et al., 1999), shown in Figure 1.2: during four stages *raw data* gets turned into *information*; (i) raw data is transformed into structure data (*data tables*), (ii) a *visual mapping* turns data into visual structures, *marks* and their visual

[2] Tukey, 1993, p. 5
[3] Tukey, 1993, p. 3
[4] Information Visualization is defined by Colin Ware as *"the use of interactive visual representations of abstract data to amplify cognition"*[5].

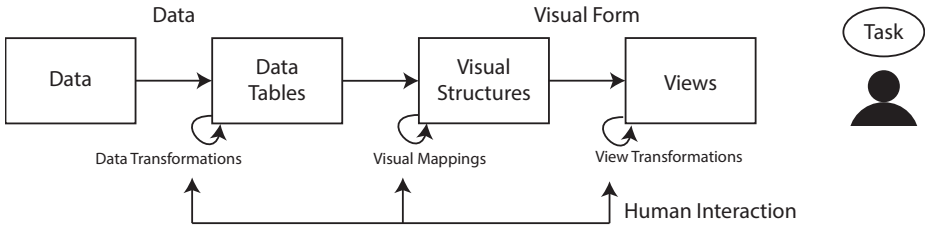


Figure 1.2.: *Infovis Pipeline model according to Card, Mackinlay, and Shneiderman (1999)*

properties, (iii) the visual form gets rendered on the screen to create a *view*, which allows (iv) perception of the image by the user and to solve particular *tasks*. An effective visual mapping is essential to every visualization and involves a large body of methods and literature on its own (e.g., Bertin, 1973; Tufte, 1986, 1990; Card et al., 1999; Ware, 2004).

Information visualization involves the use of computers and algorithms to perform each of the steps on the pipeline and to produce the final image. A single visual representation hardly answers all users' questions (Yi, Kang, Stasko, & Jacko, 2007). Users should be able to interact with the visualization in order to change the part of the data shown and adapt parameters along the InfoVis pipeline, as different user tasks require different visual representations, filtering, clusterings or visual mappings. Such an interactive and exploratory process can, to a certain extent, be described by *"Overview first, zoom and filter, detail-on-demand"*, as formulated by Shneiderman (1996). Other methods have been described as *"search, show context, expand on demand"*(van Ham & Perer, 2009), yet in general, exploration does not follow a particular methodology but every finding may lead to a new hypothesis or question.

Visualizing Networks

The most common way of looking at networks is through node-link diagrams, showing nodes as points and connections as lines between points. Nodes are placed in a 2-dimensional space using a *graph layout*. In the 1930s, Jacob Levy Moreno used node-link diagrams to study social groups and to communicate his findings. His graph layouts were hand-drawn and emphasized densely connected groups of nodes (Figure 1.3). Additional information about persons and their relations was encoded using color and different shapes. Today, many automatic layout algorithms exist, but the amount of information contained in networks (edge types, weighted edges, dynamic, etc.) requires additional visual and interactive means.

Dynamic networks with their changing topology pose numerous challenges to creating visualizations and providing interaction to explore them; visualization and interaction techniques developed for static networks, with a few exceptions, are not sufficient. While static networks are commonly understood and interpreted as static 2D maps, dynamic

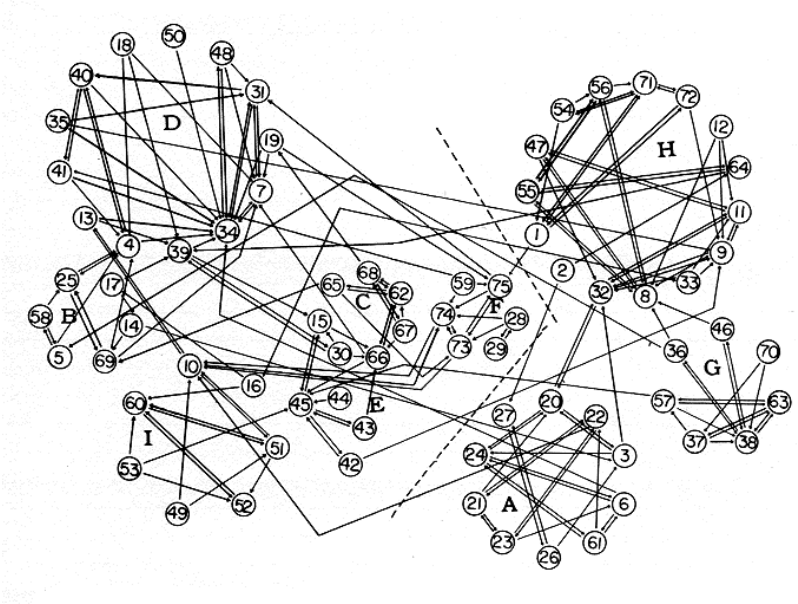


Figure 1.3.: *Handdrawing of a social network by Moreno, 1932*

networks imply dynamic media such as animation and interaction to "perceive change", "replay the evolution", and "navigate in time". Figure 1.4 gives a preview of current techniques to visualize dynamic networks and illustrates some of the problems in the field: *How to visualize topology and changes at the same time?, How to show differences between time steps? How to visualize attribute changes?, How to deal with increased (visual) density of connections, resulting from the increased number of connections over time?*.

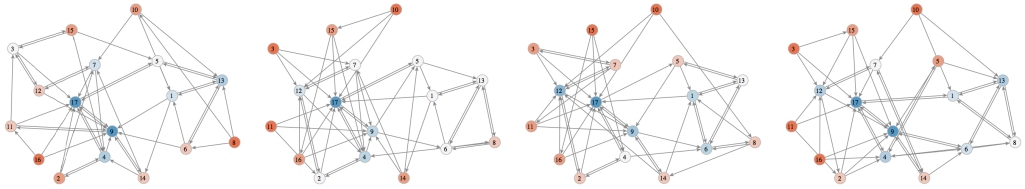
1.2. Research Questions and Approach

In this dissertation, I address the general research question:

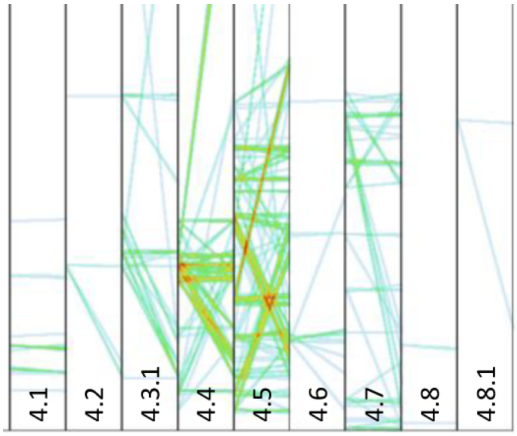
▷ **How to support the interactive visual exploration of dynamic networks?**

Supporting visual exploration means to improve the ability to an effective visual communication that allows users to gain insights into the data, i.e. obtain an overview, dig into the data, discover relevant and unexpected relations, as well as formulate and confirm hypotheses. My effort to support such a visual exploration involves the three aspects: tasks, visualization design, and evaluation.

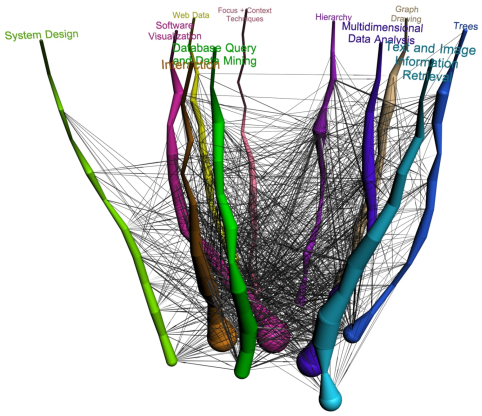
Tasks First, we must know which tasks must be supported. Tasks inform the design of interfaces as well as serve to assess their value through user evaluations afterwards. Task taxonomies exist for information visualization in general (Amar, Eagan, & Stasko, 2005), as well as for static networks (Lee, Plaisant, Parr, Fekete, & Henry, 2006).



(a) Small multiples, Brandes and Wagner (1997)



(b) Parallel Edge Splatting, Burch et al. (2011)



(c) Space-time cube, Ahmed et al. (2005)

Figure 1.4.: *Examples of dynamic network visualizations. (a) Small multiples: one picture per time point, read from left to right. (b) Parallel edge splatting: time also running from left to right, nodes are ordered on vertical axis, edges indicated between them. (c) Space-time cube: node-link diagram getting extruded into additional spatial dimension (from bottom to top), simulating time.*

▷ *How to describe and structure user tasks, necessary for dynamic network exploration?*

For dynamic networks, a taxonomy must capture tasks related to the exploration of changes over time, but no such task taxonomy has been published yet.

Visualization Design Many visualization techniques have already been proposed for the visualization of dynamic networks, including "network movies" (Moody, McFarland, & Bender-DeMoll, 2005), small multiples (Figure 1.4(a)), time line views (Figure 1.4(b)) and 3D node-link diagrams (Figure 1.4(c)), as well as methods showing temporal information in addition to the network's topology (Chapter 2). However, each of these techniques supports a particular set of user tasks, while falling short on others. Since many techniques are in fact complementary, the questions is:

▷ *How can techniques be integrated in a consistent way by keeping the interface simple?*

As current techniques are far from optimal and do not cover all possible user tasks, there is a need to improve existing techniques and think of new ones to tackle the many open problems in visualizing dynamic networks, user navigation and network characteristics such as size, density and amount of change.

▷ *How to design novel visualizations to tackle open problems in the visualization of dynamic networks?*

While there are many open problems, I address the particular problems:

▷ *How to improve temporal navigation between time steps? (Chapter 4)*

▷ *How to improve the perception of changes between two time slices? (Chapter 5)*

▷ *How to visualize dense dynamic networks with changing edge weight? (Chapter 6)*

Evaluation Visualizations can be evaluated by various methods, ranging from quantitative and controlled user studies, to qualitative evaluations, involving lay-persons, domain experts or information visualization experts (Munzner, 2009; Lam, Bertini, Isenberg, Plaisant, & Carpendale, 2012). If data is rare or does not show the desired properties to test the visualization, it must be generated synthetically:

▷ *How to generate synthetic network data for controlled user experiments?*

1.3. Thesis Statement

Exploring dynamic networks involves a multitude of tasks, which vary in their level of difficulty (task *complexity*). While many simple (*low-level*) tasks may require only one specialized visualization, more complex tasks (higher-level) tasks require multiple perspectives on the data, i.e. multiple complementary visual representations, made manageable and navigable by users though interaction.

This dissertation explores ways to interactively explore dynamic networks by means of visualization. I develop and evaluate techniques to *unfold* the complexity of dynamic networks, making them understandable by looking at them from different angles, decomposing them into their parts and relating the parts in novel ways. *Unfolding networks* is a metaphor such as different geographical projections *unfold* the spherical shape of the earth into easy-to-read two-dimensional maps, each making visible different relations between land masses and oceans, distances and areas, traveling routes and areas of interest (Figure 1.5).

We categorize tasks as queries along three dimensions: WHERE *in the network*, WHEN *in time* and WHAT *has happened* (Chapter 3). Higher-level tasks involve answering questions from different dimensions and explore these dimensions. *GraphDiaries* (Chapter 4) is a system that facilitates navigation between, as well as comparison and exploration

of individual time steps. It is based on the three dimensions of our taxonomy and aims to support higher-level exploratory tasks. GraphDiaries integrates various techniques such as animations, small multiples and difference highlighting techniques. For dense networks with changing attributes, we require matrix visualizations (Chapter 5), which can be integrated in systems such as GraphDiaries. In order to allow for multiple views, while keeping the interface complexity low, we introduce the concept of the *Matrix Cube* (Chapter 6). A Matrix Cube represents fully connected and weighted dynamic networks as a space-time cube, while allowing users to manipulate and decompose the cube to explore the data.

Three-dimensional space-time cubes provide a powerful way to *unfold* data that contains two dimensions plus time (geo-temporal data, videos, etc.). Dynamic networks are one case of such data, being represented as node-link diagrams or matrices, for example. We discuss space-time cubes as a general *mental model* (Johnson-Laird, 1983; Liu & Stasko, 2010) (Chapter 7) that:

- allows us to define a taxonomy of *operations* that span a *design space* for 2D visualizations,
- allows us to describe *characteristics* of data sets,
- helps to *discuss* visualizations and technique, and
- can serve as user *interface metaphor*, to facilitate understanding of and navigation between views on the data.

Chapter 7 generalizes the idea of unfolding networks to other data sets that involve one temporal dimension and two spatial dimensions (graph layout and adjacency matrices are 2-dimensional representations for (static) networks). The visualization of video data, multivariate data, as well as changes in geographical information can inspire the visualization of dynamic networks, as well as vice-versa. We propose a taxonomy of basic *operations* that describe the decomposition of 3-dimensional space-time cubes, leading to novel 2-dimensional visualizations. These operations help us to explore the design space for visualizing and interactively unfolding dynamic networks and other spatio-temporal data, as well as to transfer techniques across the domains.

1.4. Outline

Chapter 2 gives an overview of the major definitions and concepts about networks, temporal data and dynamic networks, as well as describing visualizations for (static) networks in general. Chapter 2 finishes with a review of techniques that are used to visualize dynamic networks, and introduces problems specific to dynamic network visualization. Following Chapter 2, this dissertation is divided into five chapters, and finishes with a chapter on conclusions and future work (Chapter 8).

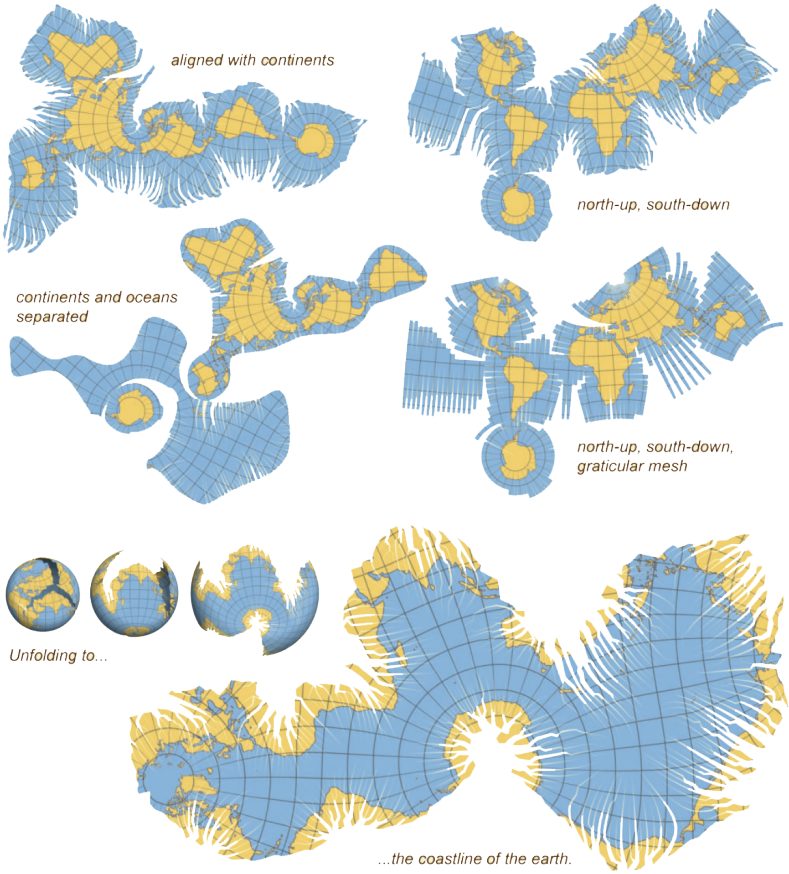
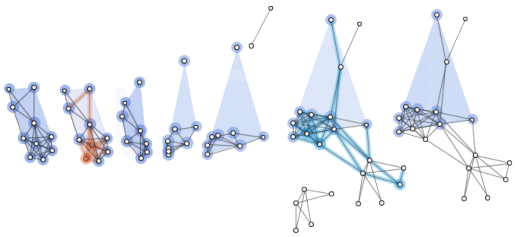


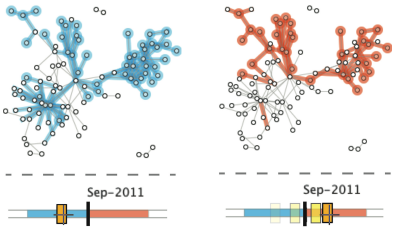
Figure 1.5.: *Unfolding the earth: different projections by van Wijk (2008), each showing a different aspect of the earth's topology.*

Chapter 3: Towards a Task Taxonomy for Dynamic Networks This chapter describes our task taxonomy, which is inspired by an approach from geography (Peuquet, 1994), as well as existing taxonomies. Tasks are structured along three dimensions WHERE, WHAT, and WHEN. We further discuss the mapping from tasks in the data space to tasks in visual space.

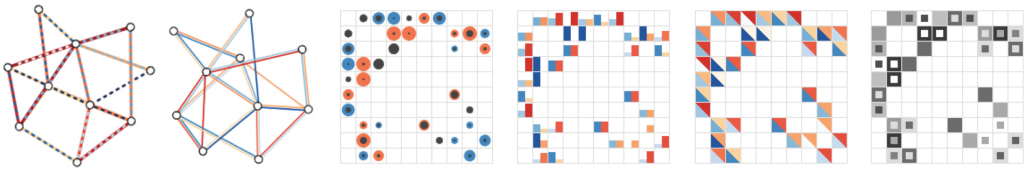
Chapter 4: Animated Transitions and Temporal Navigation Informed by our task taxonomy and its implications, described in Chapter 3, we designed a system called GraphDiaries and evaluate some of the implemented techniques in a controlled user study. Besides small multiples, difference highlighting, and animated transitions (Figures 1.6(a) and 1.6(b)) with change highlighting between individual time steps of the network, GraphDiaries includes adaptive layout stabilization and a way to track a set of nodes over time. While navigation through time in common interfaces is only supported in a linear way, we provide navigation between non-adjacent time steps as well different levels of visual feedback when navigating between time steps. A controlled user study, comparing our interface with others, concludes the chapter.



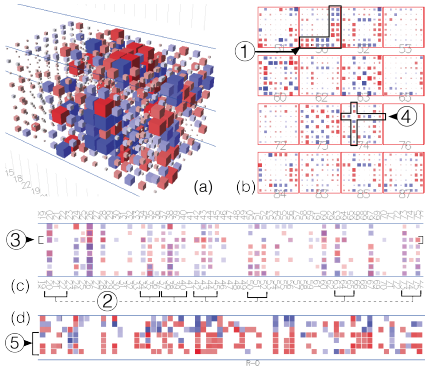
(a) Animated Transitions, Chapter 8



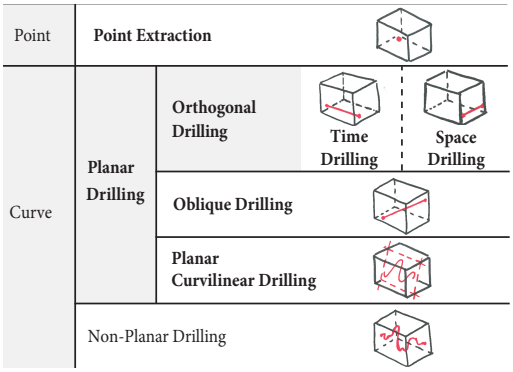
(b) Animated Transitions, Chapter 8



(c) Comparing networks, Chapter 5



(d) Matrix Cubes, Chapter 6



(e) Space-time cube operations, Chapter 7

Figure 1.6.: *Visualizations to unfold dynamic networks, presented in this dissertation.*

Chapter 5: Comparison of Dense Weighted Networks For dense networks and those with weighted edges, node-link techniques as described in Chapter 5 are not sufficient. Starting with a case study on connectivity in the brain and describing associated user tasks, we develop designs for the comparison of two dense networks with weighted edges, using both node-link and matrix representations (Figure 1.6(c)). We conducted a controlled user study to compare the most promising designs. Results show that our design for matrices outperformed that for node-link diagrams.

Chapter 6: Unfolding Networks with the Matrix Cube While matrices are effective to visualize dense networks (Ghoniem, Fekete, & Castagliola, 2005), they have not yet been studied much to visualize dynamic networks. In this chapter we describe the Matrix Cube and its parts, as well as *Cubix*, our interface that allows interaction with the cube and navigation between a set of predefined views. Cubix is demonstrated in

a walkthrough exploring a co-authorship network. We evaluate the Matrix Cube in a qualitative study with two experts from the domains of astronomy and neurology.

Chapter 7: A Framework for Unfolding Space-Time Cubes In this chapter we show how different visualizations from multiple domains (geo-spatial data, video data, etc.), including the techniques for dynamic networks explained at the end of Chapter 2, can be described as operations on space-time cubes (Figure 1.6(e)). By describing a detailed taxonomy of space-time cube operations, we generalize the concept of an interactive cube for visualization, developed in Chapter 6. This allows us to eventually discuss operations, including the resulting visualizations in terms of effectiveness. We further discuss other space-time cube systems, which employ operations on a space-time cube, and report on a use case to design a visualization for dynamic networks based on a minimal set of operations from our taxonomy.

While evaluating GraphDiaries, the need to generate data emerged. However, current graph generators require input parameters, whereby it is sometimes hard to understand how these parameters influence the topology of the final graph. Moreover many synthetically generated graphs do look artificial. We tried to invert the problem of graph generation; rather than letting the user define input parameters, we generate graphs while a user selects the graphs he considers most appropriate. In an appendix to this dissertation (*Random Graph Generation for Evaluating Network Visualizations*), we describe *GraphCuisine*, our random graph generator, which is based on an evolutionary algorithm. We report on its components and show examples of generated graphs. We finally report on a way to use GraphCuisine for the generation of dynamic networks.

1.5. Individual Contributions

The individual contributions of this dissertation are as follows:

1. A survey of the state-of-the-art in visualizations for dynamic networks (Chapter 2)
2. A task taxonomy for the exploration of dynamic networks (Chapter 3).
3. An interface for the exploration of dynamic networks based on node-link diagrams, named “GraphDiaries” (Chapter 4).
4. A controlled user study estimating the effectiveness of the transitions and navigation techniques proposed in GraphDiaries (Chapter 4).
5. Designs for node-link and matrix representations to compare weighted networks, as well as the results from a controlled user study on their efficiency (Chapter 5, involving Basak Alper (UC Santa Barbara, CA), Nathalie Henry-Riche (Microsoft Research, WA), and Tobias Isenberg (INRIA, France)).

6. Matrix Cubes as a visualization and interaction model for the exploration of dynamic networks and the description of corresponding decomposition operations (Chapter 6).
7. An interactive interface, named “Cubix”, for the exploration of dynamic weighted networks, using Matrix Cubes (Chapter 6).
8. An evaluation of Cubix by experts using the tool to explore their data (Chapter 6).
9. A general taxonomy of space-time cube operations, a typology of data sets based on shapes inside the space-time cube, as well as, an extensive discussion about visualization techniques, which are captured by the design space that is defined by space-time cube operations (Chapter 7, involving Pierre Dragicevic (INRIA, France), Christophe Hurter (ENAC, France), Daniel Archambault (Swansea University, UK), and Sheelagh Carpendale (University of Calgary, Canada)).
10. An interface and genetic algorithm for the generation of random graphs with user defined properties, allowing for the creation of data sets for controlled user studies (Chapter A, involving Andre Spritzer (Universidade Federal do Rio Grande do Sul, Brasil) and Evelyne Lutton (INRIA, France)).
11. GraphTabasco, an extension to GraphCuisine allowing to create dynamic networks with random noise.

2. State of the Art

Contents

DYNAMIC networks combine two problem domains; network connectivity and changes over time. This chapter clarifies the main definitions for graphs and networks, as well as temporal data and dynamic networks. It then gives an overview of how visualizations help in exploring (static) networks, and discusses visualizations for dynamic networks. The chapter concludes with a list of open problems in visualizing dynamic networks.

2.1. Graphs

2.1.1. Terminology

A **graph** $G = (N, E)$ consists of a finite set of *nodes* (or *vertices*) N and a finite set of *edges* (or *links, connections*) E, between pairs (n,v) of N. Nodes and edges of a graph are called its *elements*. A layout for G is called the *embedding for G*, finding a position for any node in N in $\mathbb{R} \times \mathbb{R}$. If the layout can be drawn in 2-dimensinonal space, G is called **planar**. The **adjacency matrix** M of a graph is a matrix of size $|N| \times |N|$ with an entry $m_{ij} = 1$ for every existing edge between two nodes i and j, otherwise $m_{ij} = 0$.

Edges in the graph can be modeled in different ways. Multiple edges between the same nodes are called a *multi edge* and the graph is called a **multigraph**. If edges are directed, i.e. one node is the *source*, while the other is the *target*, the graph is called **directed**

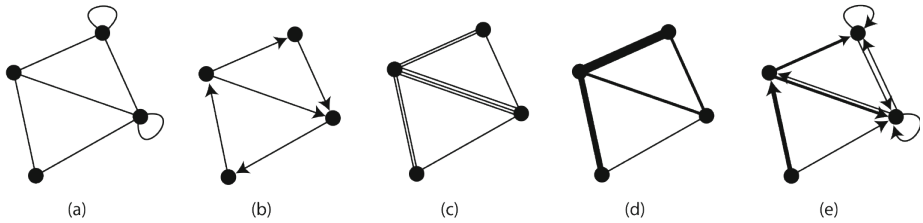


Figure 2.1.: *Graphs: (a) with self-edges, (b) directed graph (digraph), (c) multigraph, (d) weighted graph, and a combination of all (e) weighted directed multigraph with self-edges.*

(also: *digraph*). For any directly connected node pair in a directed graph, one node is the *predecessor* (the *source* of the edge, $source(e)$) and the other one is the *successor* (the *target* of the edge, $target(e)$). Edges can have a *weight* w_i associated with them, which is usually a real number. Depending on the application domain and data set, edge weight can mean different things, such as frequency of calls between persons, strengths of signals between antennas or number of messages between computers. A graph with weighted edges is called a **weighted graph**. An edge with the same source and target is called *self edge*. Combinations of graph types are possible, for example, a *directed multigraph with self-edges* (alternatively: *directed graph with multiple and self edges*). A **simple graph** has neither multi- or self-edges nor directed or weighted edges.

A *path* is a sequence of connected nodes in G, with one node being the start and another being the end. In directed graphs, all edges between nodes in a path must point into the same direction. If there are at least two paths between two nodes in G, the *shortest path* is the path with less nodes, also called *hops* between nodes. If the start and end nodes of the path are the same, the path is a *cycle*. Cycles in directed networks are called *closed walks*.

A subset of nodes and edges in G is called a *subgraph* $G' = (N', E')$ with $\forall e \in E' \Rightarrow source(e) \in N' \wedge target(e) \in N'$. A *graph partition* is a division of G into a set of *non-overlapping* subgraphs, i.e. $K'_i \bigcap K'_j \neq \emptyset \Rightarrow i = j$. The set of all edges that connect nodes in two subgraphs is called a*graph cut*. Removing these edges splits the graph into *disconnected components*, i.e. there exists no path between any two nodes from different components. By convention the path length between two nodes in unconnected components is infinite.

A **k-partite graph** is a graph that can be partitioned into k partitions so that edges exist only between nodes of different partitions. In other words, all edges in E are in the graph cut. In the case where $k = 2$, the graph is called a **bigraph**.

Graph *motifs* are subgraphs with specific topological structures, such as *groups*, *cliques*, *cycles*, and *stars* (*fans*). A *group* is a set of nodes with a high number of edges between them. In the case where all pairs of nodes in a group are connected, the group is a *clique* (fully connected group). A *star* is a node n with many *neighbors* where none of the neighbors is directly connected to another neighbor of n. A star is a special type

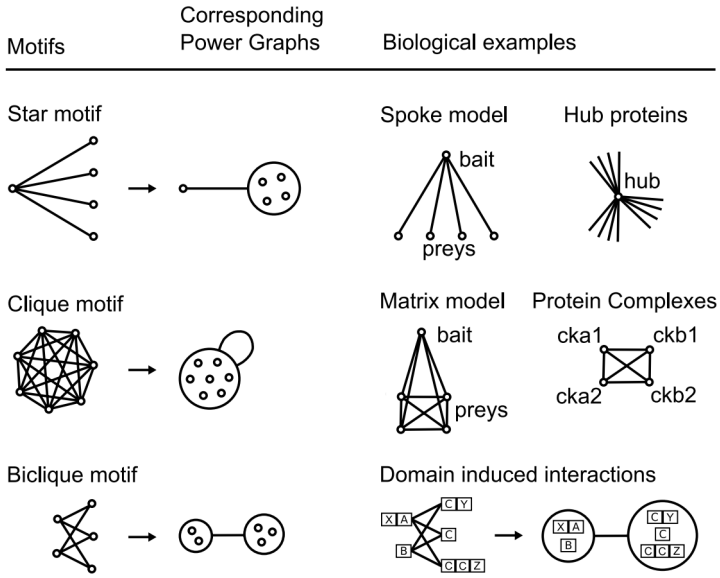


Figure 2.2.: *Motifs in a protein-interaction network (left) abstracted in a visualization called* Power Graphs *(middle) (Royer et al., 2008).*

of bigraph; one component is the central node, the other are its neighbors. In directed networks, stars with more *outgoing* nodes are called *hubs*, while *sinks* are stars with more *incoming* edges. In practice, specific motifs exist for any domain that uses graphs for calculation or data analysis (Yeger-Lotem et al., 2004)(Royer, Reimann, Andreopoulos, & Schroeder, 2008). Figure 2.2 shows a set of motifs with two to three nodes from biology.

2.1.2. Graph Measures

Graph measures quantify characteristics of a graph and its elements. Measures on nodes include the number of neighbors (*degree*), predecessors (*in-degree*) and number of successors (*out-degree*). Degree is one way to indicate the importance of a node in the network. The higher its degree, the more it is more integrated. Another important *centrality measure* is the number of shortest paths that a node is part of (*in-betweenness centrality*). Nodes that connect two densely connected components have a high in-betweenness centrality. Removing nodes with high centrality makes the network unstable and can break the network into unconnected components. More information on network measures and centrality measures can be found in Brandes and Wagner (2003).

A graphs has a *size* that indicates the number of nodes in the graph ($|N|$). Edge density is used to characterize graphs as *sparse*, *dense*, or *fully-connected*. Graph density is a visually important characteristic that poses numerous challenges to graph visualization. For density, several measures exist; not only depending on whether the graph has directed and/or self edges, but what density means for the visual density of graphs (Melançon, 2006).

The equation:

$$density_1(G) = \frac{2|E|}{|N| * (|N| - 1)}$$

expresses the ratio of existing edges to possible edges. However, in this case density grows exponentially with the number of nodes, i.e. visual density does not increase linearly. One alternative equation where density grows linearly, is

$$density_2(G) = \frac{|E|}{|N|}$$

indicating the ratio between edges and nodes.

The *diameter* of a graph is the longest shortest path, i.e. the maximal number of hops necessary to reach every node from every other one. Dense graphs usually have small diameters, since nodes are highly connected. However, density does not account for *where* nodes connect to. The *local clustering coefficient* c_l in undirected graphs for a node n indicates how much a node is part of a group (or clique) and is the ratio between the existing edges between the neighbors of a node n and the possible edges between them. That is, given a graph $G' = (N', E')$, with $N' = \{m \in N | \exists e \in E, e = (n, m)\}$ and $E' = \{e \in E | e = (m_1, m_2), m_1, m_2 \in N'\}$, then

$$c_l(n) = density_1(G')$$

The global clustering coefficient c_g in a graph G results from:

$$c_g(G) = \frac{1}{|N|} \sum_{n}^{N} c_l(n)$$

An important trait of a graph is whether it has small-world characteristic (Watts & Strogatz, 1998). In small worlds, paths between any two nodes are relatively short and the average path length only grows logarithmically with the number of nodes. Because of the short path length, for example, diseases or messages spread faster than in randomly connected networks. However, in order to asses if a graph is a small world graph, two measures with particular values are necessary:

- low average path length, and
- a high average clustering coefficient.

Small worlds are interesting not only because they are very frequent, but also because they describe an in-between state between a regularly structured network and a randomly wired network Figure 2.3.

Another important measure is the distribution of node degree. *Scale free networks*, described by Barabási and Albert (1999), have a power law distribution of node degrees meaning that there are few nodes with a high degree, but many nodes with low degree.

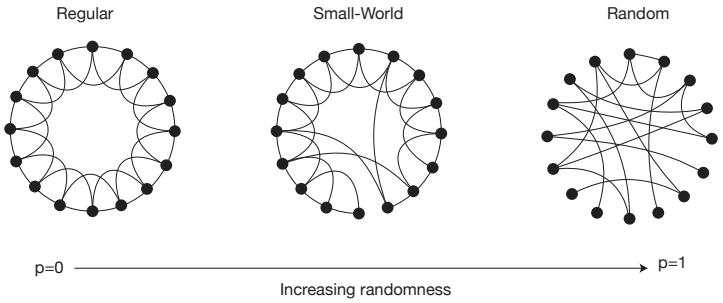


Figure 2.3.: *Small-world graphs (Watts & Strogatz, 1998). p indicating the probability of random re-wiring (re-attaching an edge to another node). Illustration after Watts and Strogatz (1998)*

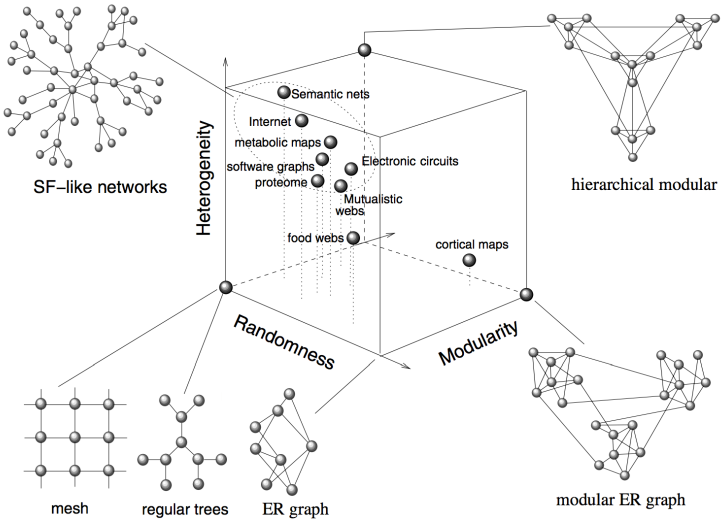


Figure 2.4.: *Representative examples of complex graphs, described along three dimensions Heterogeneity, Randomness, and Modularity Sole and Valverde (2004).*

Nodes with a high degree shorten paths between nodes, in a sense the network becomes small. Showing that a particular network has power-law degree distribution allows to reason about its stability and vulnerability to random node removal; the likelihood that the network splits into disconnected components if particular nodes are removed.

Figure 2.4 illustrate further types of complex graphs and their characteristics in a "space", described along three dimensions: *randomness*, *heterogeneity*, and *modularity* (Sole & Valverde, 2004). The figure demonstrates the need for visualization, as well as the variety of visual and topological patterns in these graphs. Translated to dynamic networks, each of these graphs is likely to produce individual patterns of change. More interestingly are the *dynamic* processes over time that organize a network into one of these particular ways.

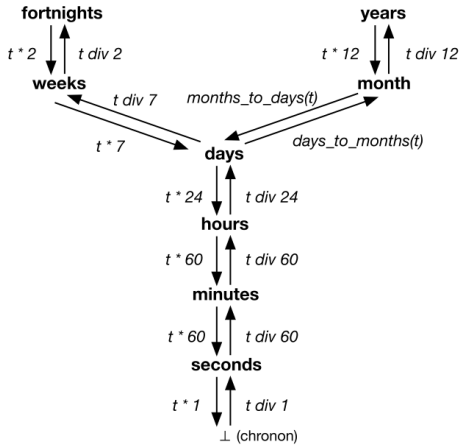


Figure 2.5.: *Conversion between (orbital) temporal units (Aigner, Miksch, Schumann, & Tominski, 2011).*

2.1.3. Networks

According to (von Landesberger et al., 2011), the term *network* in visualization refers to a graph where nodes or edges are associated with attributes. Sometimes attributes are graph measures, referred to as *topological attributes* (J. Ahn, Plaisant, & Shneiderman, 2012). Alternative terms for networks are *attributed graph* (Xu, Ke, Wang, Cheng, & Cheng, 2012; Zhou, Cheng, & Yu, 2010) and *multivariate network (*or *graph)* (Wattenberg, 2006; Pretorius & Wijk, 2006).

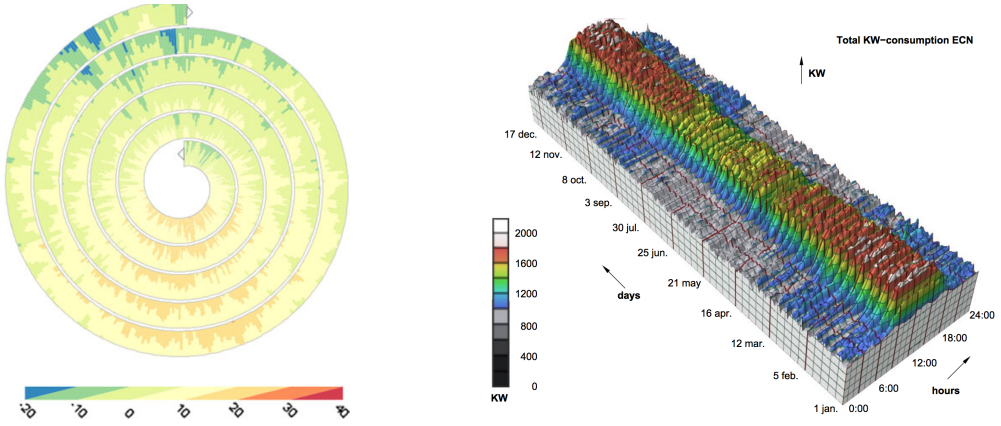
A network is defined as $\mathcal{N} = (N, E, A_N, A_E)$ with $A_N(n) : N \to V$ and $A_E(e) : E \to V$ being functions that map a node or an edge to a set of attributes V. Values can be of any type, including numerical and alphanumerical values, and depend on the data model used. If nodes and edges have explicit *types* (or *modes*) assigned, such as persons and institutions, the corresponding graph is called *heterogeneous* or *multi-mode network*.

An important aspect of visualizing *graphs* is to find a good layout that satisfies certain aesthetic criteria (Purchase, Carrington, & Allder, 2002). Visualizing *networks* faces multiple problems related to the presence of multidimensional information in addition to topology (Herman, Melançon, & Marshall, 2000; von Landesberger et al., 2011).

Dynamic networks can be seen as networks with temporal attributes on nodes and edges, plus additional information about the presence of nodes and edges. Yet time is a particular data type, as detailed in the next section.

2.2. Characteristics of Time

Time appears simple since it progresses linearly. It is divided into a *past*, a *present* and a *future*. Events in the past cannot made un-done, while events in the future are unknown,



(a) (b)

Figure 2.6.: *Examples of circular time visualization. (a) Spiral glyph showing temperature along the year (one rotation), and (b) energy consumption during the year, organized by day (Van Wijk & Van Selow, 1999)*

and present is where events *happen*. Yet time is a much more complex, and several ways have been proposed to describe and structure time. For example, time can also be thought of as being circular and recurrent, based on temporal units such as days, weeks, months, years and seasons, called *granules* (Aigner et al., 2011). This section describes the basic notions of time, temporal granules and their relations.

Temporal Granularity

Time is structured using many different *granules* which differ in their duration as well as beginning and ending time. For example:

- **orbital** units (seconds, minutes, hours, days, weeks, months years),
- domain-specific **organizational** units (fiscal year, academic year),
- **biological** cycles (REM sleep cycles, hormone cycles),
- cycles in **nature** (tide, seasons), or
- **planning** units (working steps, milestones).

Any coherent system of granularities, which allows for conversion between granularities, is called a *calendar*. Some granules are easy to convert (e.g., seconds and minutes), others overlap (e.g., weeks and months) and cannot be converted without knowing the exact time points (Figure 2.5). Units on the same granularity level (a) do not overlap, and (b) are *close*. Close means that between two units (e.g., two days), there is no gap other than another instant of that same unit (another day). The smallest temporal unit in any system is called the *chronon*. The smallest temporal unit in nature is the Planck

time which captures the minimal time that is required to change a state in the universe. A Planck time unit is 5.3906×10^{-44} seconds[1].

While the linear model of time allows us to understand and measure duration and order of events in time, circular time helps in understanding granularity and cyclic patterns in events, for example, by comparing values across the same weekday. Common visualizations are spirals (Figure 2.6(a)) and tables (Figure 2.6(b)).

A common notion of time is that of *continuous* or *discrete*. Continuous time means that *every* time point can be associated with data. Discrete time means that there are "gaps" between time points. Yet, in practice the difference is that of the level of temporal granularity. A data set that uses milliseconds to time events may not be appropriate to visualize the evolution over years. Discretizing time means aligning events from a low granularity (e.g., milliseconds) to a higher granularity (e.g., months). The difference is whether data is *sampled*, i.e. some low level granularity time points are representative for higher-level ones, or whether values and events are *aggregated* over the low-level time points. For example, weeks can be sampled by extracting the events and values for each Monday. Aggregation takes all events within this week into account and averages values.

Events and Values

Time on data can be described in two ways: *events* and *time series*. Time series reflect the evolution of ordinal or numeric attributes and are associated to a data object for *any* time point, such as the daily temperature at a specific place. Events indicate *state changes* and either describe:

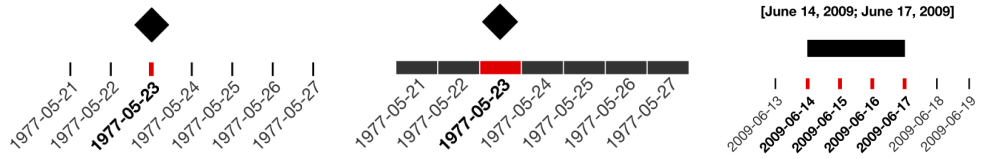
- changes in a nominal attribute (e.g., presence of a node in the network: add, remove),
- describe changes in a non-nominal attribute, such as *increase by x*, or *decrease by x*.
- describe a single occurrence in time, such as a metro incident or the presence of smog in the city.

While time series exist for any point in time (at least during a certain period), events do not necessarily exist for any time point. Yet, events can have a duration.

Temporal Primitives and Relations

When analyzing time, the important information is *when* and in *what order* something happens, as well as *how long* it takes. In order to describe temporal data and relations between data in time, we need *temporal primitives*, associated to data. Aigner et al. (2011) describe the following temporal primitives. I indicate alternative terms in brackets.

[1] http://physics.nist.gov/cgi-bin/cuu/Value?plkt



(a) Instant (b) Instant (c) Interval

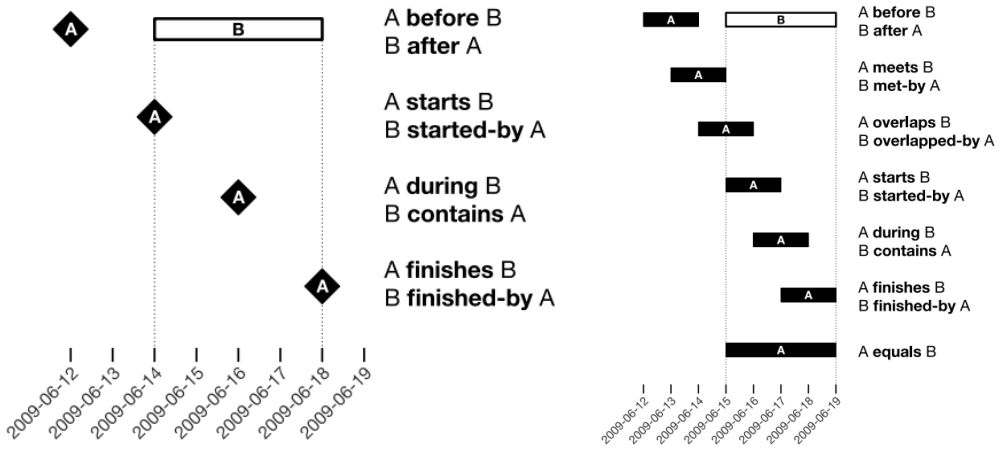
Figure 2.7.: *Categorization according to Aigner et al. (2011): Instants can refer to (a) a discrete point in time, (b) a unit on a higher-level of granularity. Intervals include multiple instants. (c) an interval spanning multiple instants.*

- An **instant** is a unique time point at an arbitrary level of granularity. An instant can be a specific second, a day or a year. Figure 2.7(a) illustrates instants at the lowest granularity level, while Figure 2.8(b) shows an instant that represents an interval on a higher-level of temporal granularity. Instants on different levels of granularity follow the same rules as units: they are non-overlapping and close.
- An **interval (period)** is defined by either two instants or by an instant and a duration. The instant can mark the beginning of the interval, or its end (Figure 2.8(a)). An interval spans a sequence of instants.
- A **span** indicates a duration and is not related to any particular time, i.e. it is a *relative* temporal primitive. Examples for spans are "two hours" or "three weeks". Instants and intervals are *absolute* temporal primitives because they are specified at a particular time.

"Monday, May 20, 2013" represents an instant, while *"Monday, May 20, 2013, 00:00 - Tuesday, May 21, 2013, 00:00"* is an interval, although the designated time is the same. That is, the notion of interval and instant depends on the current level of temporal granularity.

Temporal relations in data (*before, after, longer than*) can be described as relations between temporal primitives (Figure 2.8).

- **Two Instants:** Two instants A and B, within the same level of granularity, are either in sequence (*before*, *after*), or *equal*.
- **Interval and Instant:** An interval can be *before* or *after* an instant, can *start* or *end* at an instant, or can *contain* an instant (Figure 2.8(a)).
- **Two Intervals:** Allen (1984) describes seven possible relations between two intervals (13 including symmetry), illustrated in Figure 2.8(b). Independent from their duration, intervals can be in sequence (*before, after*), touching each other (*meet*), *overlap*, *start* or *finish* at the same time, one being completely *contained* by the other or both intervals happen in parallel (*equal*).



(a) (b)

Figure 2.8.: *Relations between temporal primitives according to Allen (1984) (figures taken from Aigner et al., 2011). (a) Relations between instants, (b) relations between intervals.*

2.3. Dynamic Networks

Dynamic networks are networks with some component that changes over time (*time-varying*). Many different terms occur in literature to describe such networks:

- **time-varying**, e.g. Hoppe and Rodgers (2013); Nicosia et al. (2013),
- **temporal**, e.g., J.-W. Ahn, Taieb-Maimon, Sopan, Plaisant, and Shneiderman (2011); J. Ahn et al. (2012),
- **longitudinal**, e.g., Moody et al. (2005); Brandes and Nick (2011),
- **evolving**, e.g. Erten, Harding, Kobourov, Wampler, and Yee (2003),
- **dynamic**, e.g., Archambault and Purchase (2013); Diehl and Görg (2002); Saffrey and Purchase (2008); Bender-deMoll and McFarland (2005),

While terms significantly differ across domains, to the best of my knowledge, there are no fundamental differences in their definition. *Dynamic network* seems to be the most frequently used term in the information visualization literature, so throughout this dissertation I use *dynamic network* as opposed to **static network**.

The following list contains a small typology of dynamic networks, based on which elements (nodes, edges, attributes) are affected by change; literature has not yet provided such a categorization.

- **Varying connectivity:** Edges are added or removed, nodes remain in the network, even if not connected, e.g. face-to-face conversation between people.
- **Varying node-presence:** Both, nodes and edges are added or removed to the network. Nodes not present in the network are unable to create connections. Note that if a node gets removed from the network its incident edges get automatically removed, as well. Examples of dynamic networks with varying node-presence are group conversations (people can join and leave the group), or food webs (species may die-out or being otherwise removed from the ecosystem).
- **Varying attribute values:** attribute values associated with nodes and/or edges over time, e.g. strength of brain connectivity, or strengths of friendship between people.
- **Transactions:** Transactions represent individual events being send from one node to the other, such as messages between people, signals between devices, or vehicles in a transportation network. The actual network topology functions as a carrier of events, i.e. only if two nodes are connected, events can be passed between both nodes. Multiple individual messages can travel along the same physical connection between machines, and multiple individual cars travel along the same roads. Changes in connectivity can be described by frequency of events (messages, cars, etc.) or changes in the underlying connection (loss of connection between machines, road work, etc.).

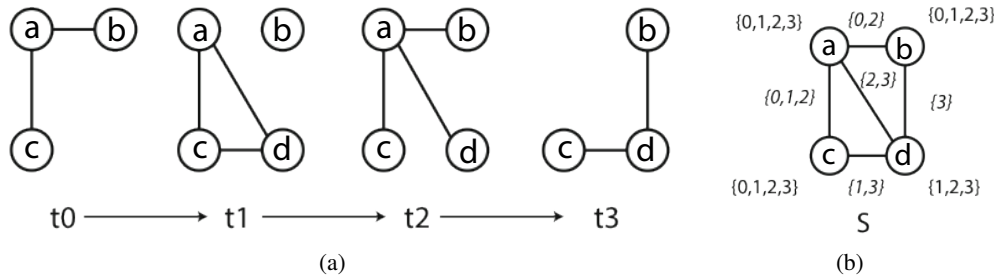
Each model conveys particular information and combinations of these cases are possible, for example a network with varying edge weight and varying node presence. More important, a network can be expressed in different ways. For example, vehicles between cities (transactions) can be modeled as attribute-varying, where the number of vehicles per hour indicates edge weight.

2.3.1. Dynamic Network Exploration

Analysis and exploration of dynamic networks can be divided into two major scenarios: *online* and *offline*.

Online scenarios In online scenarios the network is explored while data gets still updated and the visualization has to be updated as well. Examples for online scenarios are the monitoring of computer and traffic networks. The task focus in online scenarios is on judging *current* events and developments, such as abnormal behaviours and exceptions.

Offline scenarios In an offline scenario, the entire network is present at the time of observation, i.e. data does not update. Common offline scenarios are the analysis of



(a) (b)

Figure 2.9.: *Three representations of the same dynamic network: (a) Snapshots per instant, (b) supergraph with lifespans (Diehl & Görg, 2002).*

social networks or brain connectivity data resulting from cognitive experiments. The focus in offline scenarios is on the analysis and exploration of the entire network in order to estimate trends in evolution, sequences of events as well as causalities.

The particular scenario has an important influence, for example, on the choice of layout: online scenarios require layouts that remain stable *while* future topologies are unknown. In offline scenarios, an stable graph layout can be calculated and remains fixed (see Section 2.7).

2.3.2. Definitions for Dynamic Networks

A dynamic network $D = (N, E, T)$ consists of a set of nodes N, a set of edges E, and a set of time points T. If D has attributes associated to nodes and edges, two functions $A_N(a, n, t)$ and $A_E(a, e, t)$ return the value of attribute a on node n or edge e, for time t. For each dynamic network, an ordered set of *time steps* $[G_0, G_1, G_2, ...G_{|T|}]$ is called a *sequence*, with G_t being a static representation of D at time t (Figure 2.9(a)).

Nodes N_t and edges E_t are subsets of N and E respectively, whereby all incident vertices of all edges in E_t have to be in N_t. G_{t-1} is called the *preceding time step*, while G_{t+1} is called the *succeeding time step*. Preceding and succeeding time steps are *adjacent* to G_t. Since time steps compare to instants, and instants are close, no information is lost between two adjacent time steps; any event in the network is captured in either time step.

Nodes and edges are associated with a *birth*, which refers to the first instant they enter the network, and a *death*, referring to the last time step before they leave the network. Both nodes and edges can be *re-inserted* into the network The time between birth and death is called *lifespan* and nodes may *re-enter* the network after they have have been removed from it. The **supergraph** (Diehl, Görg, & Kerren, 2001) $S = (N, E)$ is the graph that contains all nodes and edges present in at least one time step of D. Figure 2.9(b) shows the supergraph of the time steps on the left, indicating the time steps where each node (non-italic) and edge (italic) are present. For instance, node a has a lifetime of 3

time steps, while the edge between nodes *b* and *d* has a lifetime of 1 time step. The edge between *C* and *D reappears*.

For the quantification of dynamic networks, measures for static graphs can be calculated for every time step individually and then observed over time. Furthermore, a range of metrics for dynamic networks there has been described (Nicosia et al., 2013). For example, two nodes n_0 and n_i are connected by a *temporal path* if there exist a sequence of edges $[(n_0,n_1),(n_2,n_2),..(n_{i-1},n_i)]$ where each edge (n_{p-1},n_p) is *prior* to the adjacent edge (n_p,n_{p+1}). *The temporal distance* of two nodes is consequently defined as the duration between the first and last connection. The distance between two graphs $G_1 = (N_1,E_1)$ and $G_2 = (N_2,E_2)$ can be calculated in many different ways (Bilgin & Yener, 2010). One possibility is the size of the *common subgraph* (Bunke & Shearer, 1998):

$$distance(G_1,G_2) = \frac{|N_1 \cap N_2|}{max(|N_1|,|N_2|)}$$

Federico, Pfeffer, Aigner, Miksch, and Zenk (2012a) measure the ratio between added and removed nodes in the neighborhood of a node and define a *change centrality* metric. The value effectively quantifies the change in a node's neighborhood. As a direct use of change centrality, Federico et al. suggest to use it to stabilize the layouts; nodes with high change centrality must be allowed to move more, while those with low change centrality may keep their positions.

2.3.3. Models

Static networks can be represented in different forms, such as *edge lists* or *adjacency matrices*. Similarly, dynamic networks can be stored and thought of in different ways: *snapshots*, *events*, and *time series*. Pajek (*Pajek - Program for Large Network Analysis*, 1997) is a commonly tool for network analysis (providing layout algorithms for node-link diagrams, data structures for network analysis, network analysis measures, etc.) and supports all three kinds of representations.

Snapshots Snapshots represent the network's state and topology for every instant. The term "snapshot" comes from photography and in terms of dynamic networks can capture data from a longer time interval (cf. Figure 2.8(b)). For example, Pajek can export adjacency matrices, one for every time step. Node and edge presence, as well as attribute values are encoded explicitly within each snapshot. Birth events and lifetimes must be inferred by comparing individual snapshots.

Events In online scenarios, data may be described as events which update the current data. Other (raw) data is in the form of individual events, such as e-mail messages or face-to-face conversations between people, or people entering or leaving a discussion room. Events can correspond to instants (node birth, node death) or intervals (conversation

Command	Description	Command	Description
TI t	initial event: following events happen at start of time segment t	DE uv	delete edge (u:v)
TE t	end event: following events happen at end of time segment t	DE uv	delete edge (u:v)
AV vns	add vertex v with label n and properties s	CA uvs	change property of arc (u, v) to s
HV v	hide vertex v	CE uvs	change property of edge (u:v) to s
SV v	show vertex v	CT uv	change type (arc/edge) of line (u, v)
DV v	delete vertex v	CD uv	change direction of arc (u, v)
AA uvs	add arc (u, v) with properties s	PE uvs	replace pair of bidirected arcs by a single edge
HA uv	hide arc (u, v)	AP uvs	add pair of arcs with properties s
SA uv	show arc (u, v)	DP uv	delete pair of arcs
DA uv	delete arc (u, v) EP uvs replace edge by pair of opposite arcs		
AE uvs	add edge (u:v) with properties s		
HE uv	hide edge (u:v)		
SE uv	show edge (u:v)		

Table 2.1.: *Commands and events defined in Pajek (Mrvar & Batagelj, 2004).*

between people). Internally, Pajek uses events to describe changes to the network, summarized in Table 2.1 (Pajek uses the term *arcs* for directed edges). Events describe a change on nodes and edges that happen in a single instant. Events in Pajek describe simple events on the network topology.

Time Series Alternatively to multiple matrices, Pajek can encode dynamic networks as time series of node and edge presence. The following shows an excerpt from how Pajek stores networks inside a `.paj` file.

```
*Vertices 3
 1 "a" [5-10,12-14]
 2 "b" [1-3,7]
 3 "e" [4-*]
*Edges
 1 2 1 [7]
 1 3 1 [6-8]
```

The example defines three nodes; *a*, *b*, and *e*. Time steps where nodes and edges are present in the network are indicated in squared brackets. Node *a* is present in time steps 5 to 10 and 12 to 14, node *b* from 1 to 3 and in time step 7, and node *e* is present from 4 on (till the very last time step). Edges exist between nodes 1 and 2 in time 7 and between 1 and 3 (with weight 1) from time 6 to 8. Time series can also represent attribute values over an interval of instants.

Snapshots, events and time series represent the same data, although each may be more suited for certain queries and use cases. Conversion between these is possible: individual

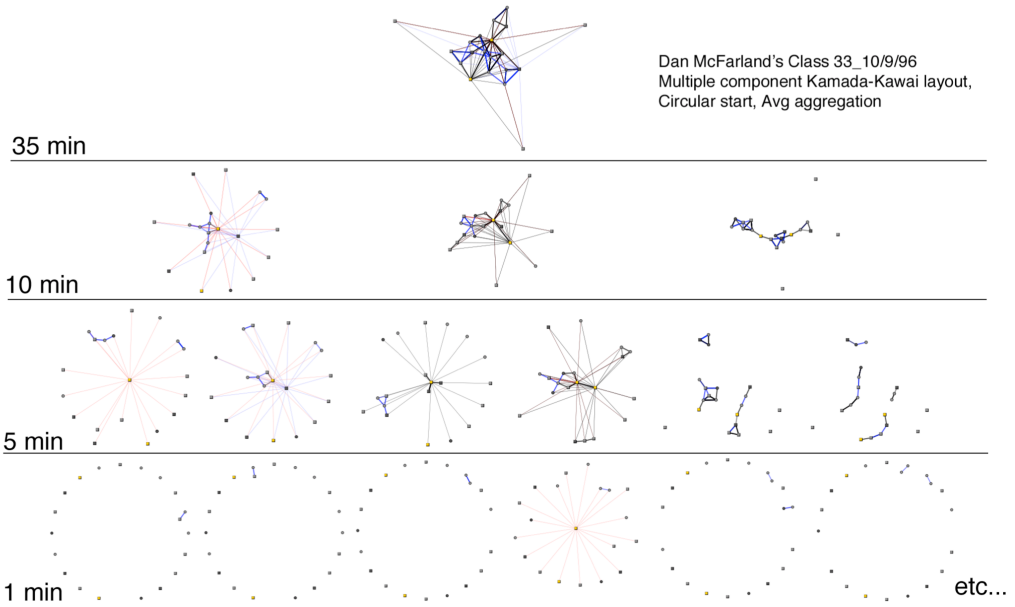


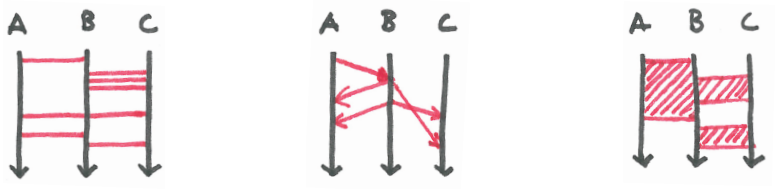
Figure 2.10.: *Network at different levels of temporal granularity (Bender-deMoll & McFarland, 2005). The bottom line shows snap shots at a very fine temporal granularity, while the top most graph shows the supergraph, i.e. at the highest temporal granule.*

events can be aggregated into snapshots; time series can be inferred from snap shots (cf. Figure 2.9); events can be derived from both, and vice versa.

2.3.4. Temporal Granularity and Aggregation

Temporal granularity is an essential aspect of dynamic network exploration. Moody et al. (2005) summarize that any (dynamic or static) network structure is a rather artificial construction that emerges only from *"aggregating dead past events"* (Moody et al., 2005, p. 1208). A network that describes messages sent between people does not exist in permanence and does not "change" from one second to the next. In fact, the network's particular topology depends on the interval that is aggregated; topology and the visible structure depends on the size of the chosen *time frame* (or *time window*).

Finding the right size of temporal granularity is important for visual network exploration. Figure 2.10 shows the same network at different levels of temporal granularity. The bottom line shows snap short at a very fine temporal granularity, while the top most graph shows the supergraph, i.e. at the highest temporal granule. Networks at low granularity contain too little information to see any topological patterns, while the highest granularity levels show too many connections and loose information about time. Yet, the duration of a connection is of major importance to temporal aggregation.



(a) (b) (c)

Figure 2.11.: *Types of connectivity. A, B and C represent nodes, time goes from top to bottom, edges are indicated in red. (a) Instantaneous edges, (b) Transmissive edges, (c) Persistent edges.*

Types of Connectivity

Nodes in dynamic networks usually represent persistent entities in the data, such as persons, countries or neurons. However, edges in dynamic networks can have multiple interpretations, depending on the data domain. Here, I introduce the following grouping, illustrated in Figure 2.11.

- **Instantaneous edges** are single events between two nodes and without any duration (Figure 2.11(a)). Instantaneous edges represent messages between people or computers, or signals between neurons in the brain. They have no duration, only a single time stamp.
- **Transmissive edges** represent moving objects such as airplanes, metro trains, blood cells or abstract transmissions in progress. Such edges have a *start* and an *end* time (Figure 2.11(b)). Between start and end, their state conveys the fraction where the moving object is.
- **Persistent edges** can also have a duration, but represent permanent states, such as anatomic connections between neurons, the existence of a flight route, or friendship between individuals (Figure 2.11(c)).

While each of these types imply a proper data model and individual visualization, they all can be modeled as edges in a network, depending on the temporal granularity.

Time Windows

A time window W represents a temporal span of duration d. An interval $[p,q]$ that is defined by W, defines a graph sequence $[G_p, G_{p+1}, ...G_{q-1}, G_q]$. W can define a supergraph $S' = (N', E')$ with $N' = \bigcup N_i, i \in [p,q]$ and $E' = \bigcup E_i, i \in [p,q]$. Multiple time windows could be defined, while two windows W_1 and W_2 can overlap.

Figure 2.12 illustrate a time window between two time points (“slice points”), while horizontal gray bars indicate the lifetime of certain nodes and edges (“arc”). Whenever a

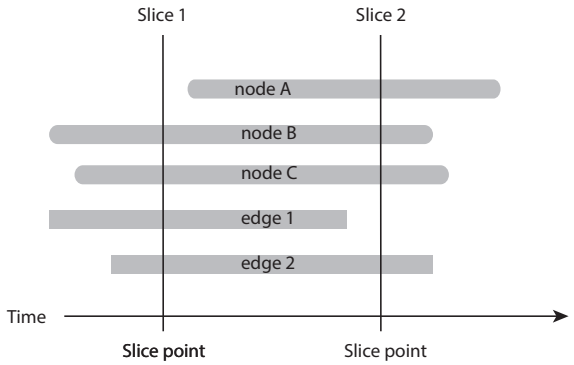


Figure 2.12.: *Time window between two instances ("slice point"). Gray vertical bars represent the lifetime of nodes and edges ("arc") in the network (Illustration according to Bender-deMoll & McFarland, 2005).*

node or edge is present inside a time window, it is part of the supergraph's topology for that time window. In the example, the time window contains all nodes and arcs.

In Figure 2.10, graphs are *aggregated* from the bottom to the top, from the top to the bottom *de-aggregated* or *decomposed*. Given a time window W with a sequence $[G_p, G_{p+1}, ...G_{q-1}, G_q]$, the supergraph $S' = (N', E')$ which results from W is the graph for the next higher level of granularity.

If nodes and edges in D have numerical attributes with associated time series, we can define an aggregation function for each attribute. This function returns a single measure for the attribute value between two time points. Common aggregated values from a series of values are *average*, *median*, *min*, *max*, *number* and *sum* (cf. Elmqvist, Do, Goodell, Henry, & Fekete, 2008; Elmqvist & Fekete, 2010).

In summary, most static networks can be seen as both a snapshot of a dynamic network or a completely aggregated dynamic network.

2.4. Network Visualization

The visualization of networks is related to several issues:

- the description of tasks that user's perform when exploring and analyzing a network,
- visual representations, such as node-link diagrams, matrices, arc diagrams (Wattenberg, 2002), etc.,
- graph layouts (*graph drawing*) and matrix reordering algorithms,
- user navigation in networks,
- clustering and aggregation for compound graphs,
- visual encoding of nodes, links and their attributes, as well as

- applying and developing evaluation methods for empirically assessing the effectiveness of network visualizations.

This section first gives an overview of the most common representations for networks and their visual encodings, and then briefly discusses general problems in visualizing (static) networks.

2.4.1. Visual Graph Representations

A graph representation describes the general visual representation of nodes and edges and how connectivity is encoded visually. Figure 2.13 shows a systematic design space of graph representation techniques and layouts, compiled by Bertin (1973). Some of these techniques have not been used yet, others have been used frequently and obtained proper names, for example: Arc Diagram (1), Circular Layout / Ring Layout (4), Node-link diagram (7), Containment diagram (10), Directed Acyclic Graph or Tree (11), Adjacency Matrix (20).

The following presents visualization techniques for the two most common representations: node-link diagrams and adjacency matrices.

Node-link Diagrams

Node-link diagrams have been used for most of the techniques in graph visualization and their advantage is that they are easy to understand and more intuitive than adjacency matrices (see next subsection “Adjacency Matrices”).

Node-link diagrams are usually displayed using 2D layouts, rarely 3D layouts (e.g. Balzer & Deussen, 2007). The field concerned with finding layout algorithms is called *graph drawing*. Layout algorithms try to satisfy a number of *aesthetic criteria*, which are measures that characterize the human readability of the layout, i.e. the graph (Purchase, 1997, 2000, 2002) and many studies on graph layout techniques have been conducted (Dwyer et al., 2009). Examples of such criteria are (a) minimize edge crossings, (b) minimize edge bends, (c) maximize symmetry, (d) minimize total drawing area, (d) place connected nodes close to each other, and (e) prevent nodes from overlapping.

Techniques for creating layouts include, but are not limited to the techniques described in the first column in the upper table in Figure 2.13 (numbers in brackets refer to the corresponding case):

- force-directed approaches (7), e.g., Fruchterman and Reingold (1991); Kamada and Kawai (1989),
- geometric optimizations, a sub-type (7), e.g. Eades and Hong (2013),
- circular layouts (4), e.g. Six and Tollis (2013),

- layouts for clustered compound graphs, (7) and (10) , e.g. Bertault and Miller (1999); Frishman and Tal (2004)),
- layered layouts for directed acyclic graphs, (11), e.g. Sugiyama, Tagawa, and Toda (1981), or
- topographical structures, a sub-type of (7), e.g. Archambault, Munzner, and Auber (2007)

Further information on graph layouts can be found in Battista, Eades, Tamassia, and Tollis (1998), von Landesberger et al. (2011), as well as Tamassia (2013). However, the main problem with node-link diagrams remains graph density, since dense graphs result in more lines crossing and overlapping. At the same time, node positions may be less representative for node connection in dense networks (criteria (d)), since any node has a large set of neighbors.

Many network visualization systems support node-link diagrams, such as the representative selection. While JUNG (O'Madadhain, Fisher, & Nelson, 2003) and Tulip (Auber, 2003) are programming libraries, Pajek (*Pajek - Program for Large Network Analysis*, 1997), Cytoscape (*Cytoscape: An Open Source Platform for Complex Network Analysis and Visualization*, 2013), Gephi (Bastian, Heymann, & Jacomy, 2009), Orion (Heer & Perer, 2011) or Ploceus (Liu, Navathe, & Stasko, 2011) are applications that allow users to visualize their own data. Cytoscape and Gephi also provide APIs for developers to write plugins. NodeXL (*NodeXL: Network Overview, Discovery and Exploration for Excel*, n.d.) is an Excel extension to visualize tables as networks. NodeXL features several extensions such as motif simplification (Dunne & Shneiderman, 2013) and visualization of dynamic networks (J.-W. Ahn et al., 2011). All of the mentioned systems support multiple layout algorithms, apply different colors to nodes and visualize clusters.

A further commonly used type of node-link diagrams are *arc diagrams*, which order nodes in one dimension (Figure 2.13(1)), while showing links between nodes as visual arcs (e.g. Wattenberg, 2002; Neumann, Schlechtweg, & Carpendale, 2005).

Adjacency Matrices

Adjacency matrices represent nodes as rows and columns, while edges correspond to matrix cells. Adjacency matrices do not suffer from the problem of edge-crossings and can visualize dense and fully-connected graphs. Although matrix representations have been in use for a long time (Bertin, 1973; Ziegler, Kunz, & Botsch, 2002; Abello & van Ham, 2004; Liiv, 2010), they have not received wider attention until the controlled user study by Ghoniem et al. (2005) showed that matrices outperform node-link diagrams on most tasks, except finding a path between two nodes.

Similar to a graph layout, matrices require an ordering on the nodes (ordering of rows and columns) in order to make particular connection patterns emerge inside the matrix. Matrix ordering is an optimization problem that tries to minimize the euclidean distance

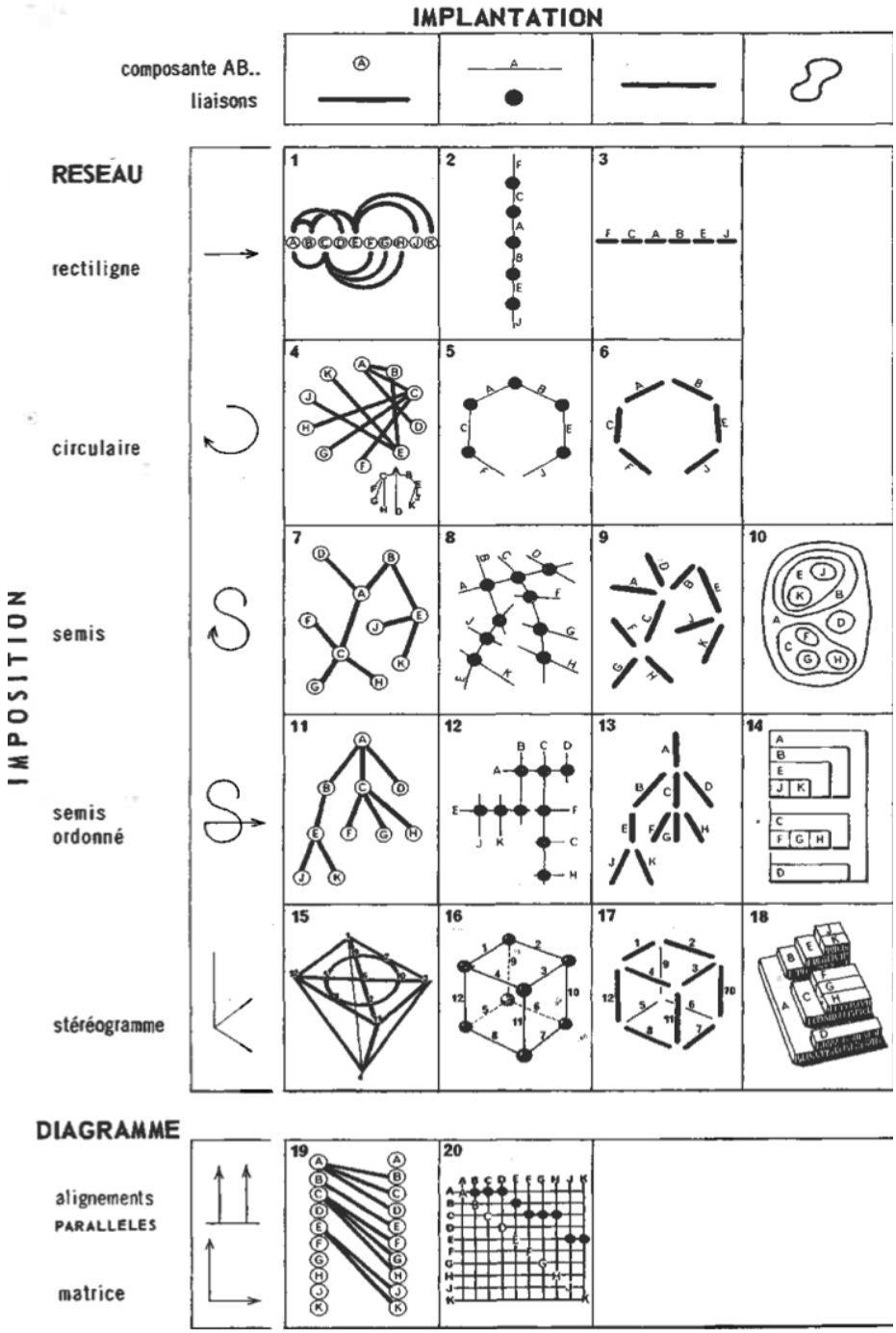


Figure 2.13.: *Graph representations taken from Bertin (1973). Some of these representations are commonly referred to by certain names, others have not been used so far. For example, (1) Arc Diagram, (4) Circular Layout / Ring Layout (7) Force directed, (10) Containment diagram, (11) Directed Acyclic Graph (or Tree), (20) Adjacency Matrix.*

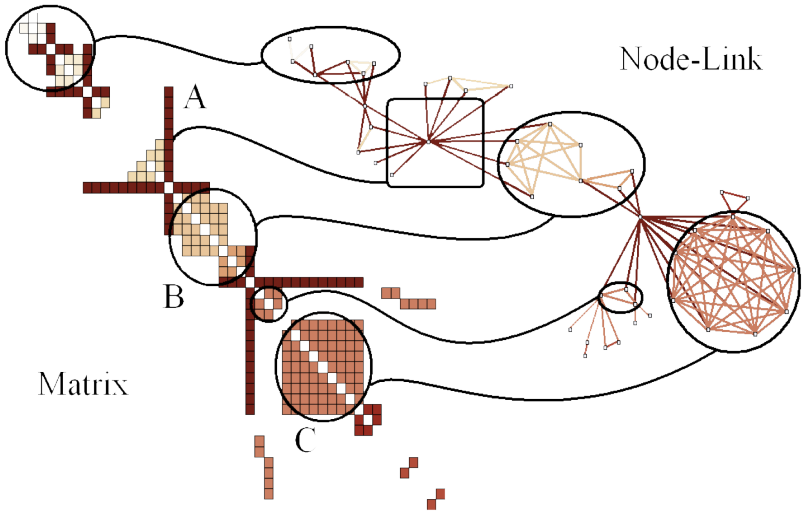
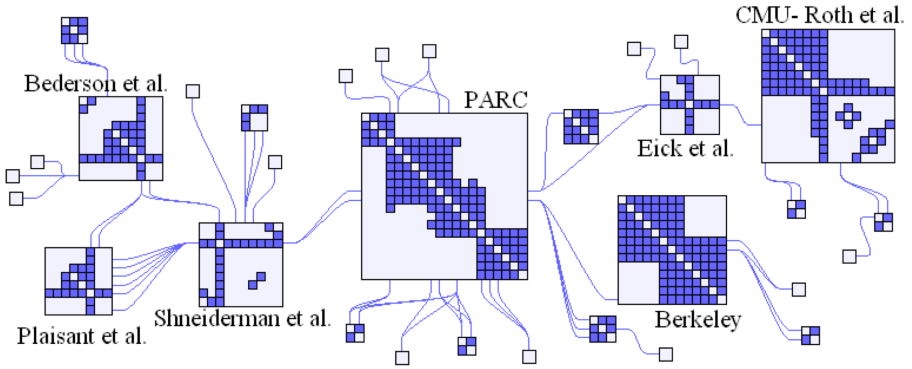
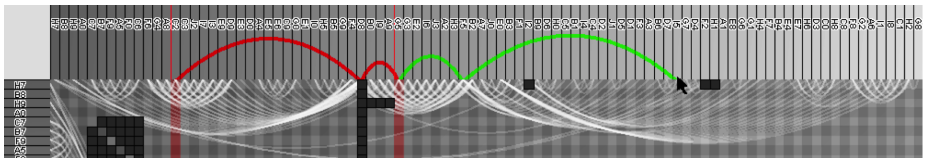


Figure 2.14.: *Network topological patterns in both, adjacency matrix and node-link representation (Henry & Fekete, 2006). The pattern labeled with an A shows a star motif, while pattern B shows a dense cluster. Pattern C shows a clique (complete subgraph).*



(a) Nodetrix



(b) MatLink

Figure 2.15.: *Matrix visualizations: (a) NodeTrix hybrid visualization for locally dense (matrix) and globally sparse (node-link diagrams) networks (Henry, Fekete, & McGuffin, 2007). (b) Matlink showing links between nodes (columns) to support path following (Henry & Fekete, 2007).*

between adjacent rows and columns. A summary of matrix reordering techniques can be found, for example, in Liiv (2010) as well as (Henry & Fekete, 2006).

Figure 2.14 illustrates how topological patterns are represented in both an ordered matrix and a node-link diagram using a force-directed layout algorithm. The pattern labeled with an *A* (dark red) shows a star motif, while pattern *B* (bright yellow) shows a dense cluster. Pattern *C* (orange) shows a clique (complete subgraph).

Some networks are only dense on a local level, that is they contain dense groups, and few connections between groups. As Figure 2.14 shows, such networks cause much space in the matrix to remain empty. Matrix Explorer (Henry & Fekete, 2006) shows a node-link diagram and a matrix side-by-side using brushing and linking to relate graph elements in both representations. NodeTrix (Figure 2.15(a)) integrates matrices directly into node-link diagrams (Henry et al., 2007), which facilitates the visualization of dense components in networks, while edges between dense groups are shown as lines. This technique was later extended to heterogeneous networks (Bach, Pietriga, & Liccardi, 2013).

Visually, there are two major drawbacks with matrices: finding paths between nodes and dealing with the empty visual space in sparse networks. Following paths in symmetric matrices (directed graphs) results in following a row until a connection is found, then traversing that column until the next connection is found and so on. In short, following paths in matrices is highly cognitive work. To facilitate finding and following paths between two nodes, Henry and Fekete (2007) show links between columns on matrices (Figure 2.15(b)) while Sheny and Maz (2007) connect cells in the matrix by lines. Other common approaches indicate the shortest path distance between any node pair as value in the corresponding cells.

In order to reduce the empty visual space in large matrices, Elmqvist, Do, et al. (2008) aggregate the network by combining nodes into meta-nodes and effectively reducing the number of nodes. Elmqvist, Henry, Riche, and Fekete (2008) use folding of the 2D space to compare distant rows. Dinkla, Westenberg, and van Wijk (2012) compress matrices and Bezerianos, Dragicevic, Fekete, Bae, and Watson (2010) remove quasi-non existent cells to visualize directed acyclic graphs.

2.4.2. Problems in Network Visualizations

Large Networks Not only do large networks pose significant problems to a visualization and a user's perception, also calculating layouts and managing large numbers of nodes in memory is challanging. While some layouts are less time intense to calculate (e.g. Kamada & Kawai, 1989), the GPU (graphics processing unit) can be used to calculate layouts (Frishman & Tal, 2007). Multilevel graph layouts decompose the network into hierarchical layers and then calculate a layout independently for each cluster in parallel. Layers can stem, for example, from a hierarchical clustering (Walshaw, 2001), or topological features (Archambault et al., 2007). Alternatively, only a part of the network

can be visualized at a time. As users change the scope, additional elements are displayed, while others get removed. Corresponding techniques are further discussed in Section 2.5.

Dense Networks Since matrices do not suffer from edge-crossing issues, they are commonly used to visualize dense networks. If matrices cannot be used, for example because nodes have geographic positions that must be presented, density in node-link diagrams can be reduced by filtering connections, for example those with low weights. An effective method is presented by Holten (2006) who bundle links with similar source or target regions. Another solution are lenses that locally remove edges that are not connected to the nodes in the lens' focus (Tominski, Abello, van Ham, & Schumann, 2006).

Multivariate Networks Links in node-link diagrams can show directionality (Holten & van Wijk, 2009) and multi-edges can be visualized by arcs (Henry Riche, Dwyer, Lee, & Carpendale, 2012). Extending the NodeTrix technique (Henry et al., 2007), OntroTrix (Bach et al., 2013) uses a specific color encoding of matrix cells as well as allows for reorganizing matrices by interactive node reordering, as well as splitting and merging matrices. Pivot graphs (Wattenberg, 2006) show a node for every attribute value, while links between these nodes represent actual relations in the network.

Compound Networks If networks get large, subgraphs can get abstracted by meta nodes and edges between them represented as meta edges (Eades & Feng, 1997; Shen, Ma, & Eliassi-Rad, 2006). Motif simplification replaces certain graph motives, such as fans, and cliques by perceptually simple glyphs such as rhomboids or circle sectors (Dunne & Shneiderman, 2013). Hierarchies of clusters can be represented using common tree representations, such as nested circles (Archambault, Munzner, & Auber, 2008), rectangles (Neumann et al., 2005), or treemaps (Fekete, Wang, Dang, & Plaisant, 2003). Overlapping subgraphs pose further problem when visualizing graphs. Corresponding techniques have been developed using Euler diagrams (Henry-Riche & Dwyer, 2010), LineSets (Alper, Riche, Ramos, & Czerwinski, 2011) and as fuzzy communities (Vehlow, Reinhardt, & Weiskopf, 2013).

2.5. Network Navigation

Many graphs are too big to fit on screen while remaining readable. Even if they do, users may not be interested in all parts of the graph at the same time. Only a part of the network can be shown, while users *navigate* through the graph in order to reveal different parts. Graph navigation techniques define and adjust the visible part of the graph, such as *panning* defines the *position*, and *zooming* the *viewing distance* of the viewer with respect to the data. While pan and zoom techniques are performed on the view (*view navigation*) and are independent from the underlying visualization technique,

graph navigation techniques rely on the graph's topology to define what is made visible to users.

Graph navigation is performed by interaction, as only the user can decide what he is interested in and change his focus accordingly (von Landesberger et al., 2011). In analogy with the information visualization pipeline (Card et al., 1999), we can describe navigation on each of the pipeline's steps; navigation in data space (network structure), navigation in the visualization, and navigation in the view. In this section, I categorize graph navigation techniques into four groups, whereby techniques from different groups can be combined in the same interface. In each of these groups, *horizontal navigation* refers to a change in the subgraph that is seen, while *vertical navigation* refers to the number of elements and/or the level of detail in the current view.

- **View Navigation** changes the user's position with respect to the visualization, while remaining independent from the data (time, topology, attributes, etc.)
- **Structural navigation** changes the visible parts of the graph,
- **Temporal navigation** changes temporal scope and granularity, and
- **Navigation in the visualization** changes the representation on the graph within the visualization.

Among all four groups, the challenges for effective visualization include (i) interaction for navigation, (ii) visual clues while changing the visible part of the network, as well as (iii) to enable user to maintain context about the space they navigate in.

2.5.1. View Navigation

A simple technique for vertical navigation has been presented by van Ham and van Wijk (2004), keeping node size fixed with respect to screen size (Figure 2.17(a)). By zooming out, nodes implicitly get combined into clusters, without calculating any clustering. However, the problem is that links in dense are (Figure 2.17(a) left, "Classical area") vanish quickly. *Graph folding* (Carpendale, Cowperthwaite, Fracchia, & Shermer, 1996) uses a geometric fisheye on graphs and, by using multiple foci, enables direct comparison between distant subgraphs.

In order to maintain orientation while panning, Ghani and Elmqvist (2011) tested different background and node encoding techniques, such as dividing the background into colored regions, adding landmark glyphs, and color nodes according to their position in the graph. Ghani and Elmqvist conclude that background coloring in combination with landmark glyphs perform best.

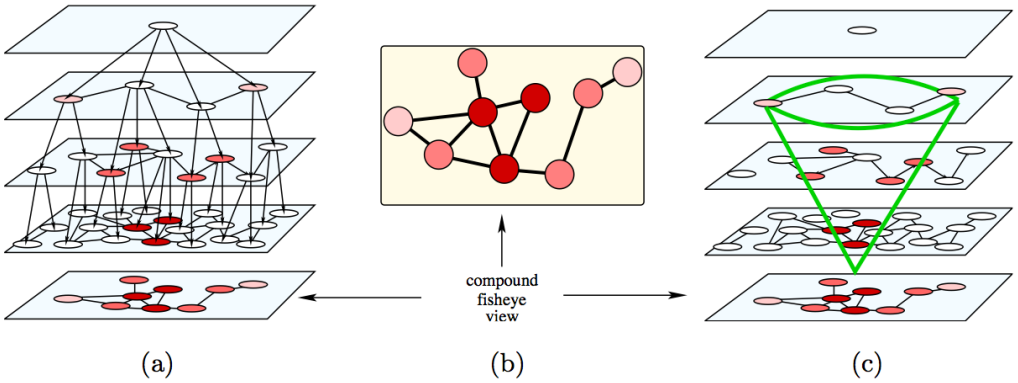
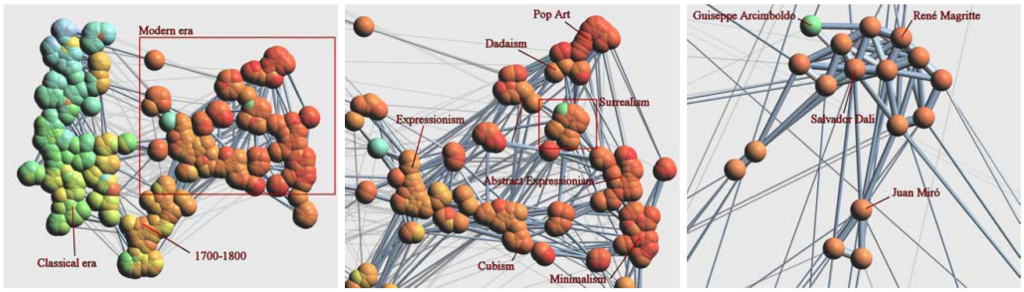
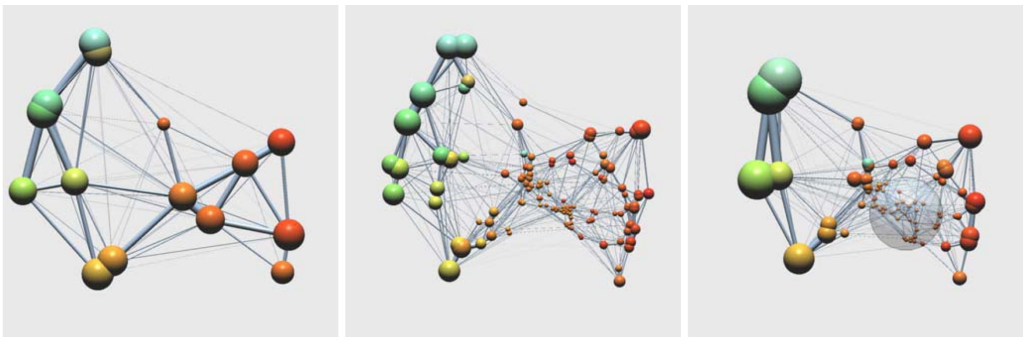


Figure 2.16.: *Compound fisheye technique as illustrated by Abello, Kobourov, and Yusufov (2004). van Ham and van Wijk (2004) developed a similar technique at the same time. (a) Hierarchical clustering, (b) resuling view with focus on dark red nodes, (c) nodes within the green cone are expanded, i.e. its children are shown.*



(a) Node size is fixed relative to screen size



(b) Fisheye used to resolve clustering and show underlying graph elements.

Figure 2.17.: *Techniques for vertical graph navigation as described by van Ham and van Wijk (2004).*

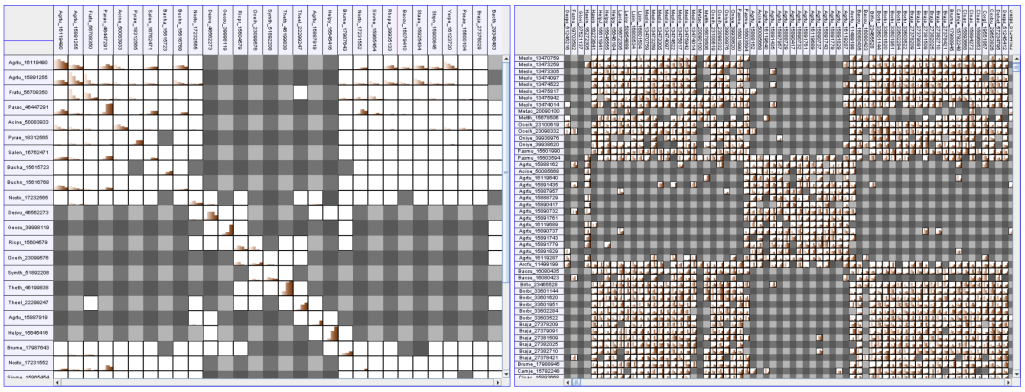


Figure 2.18.: *Same data set shown at two different zoom levels in ZAME (Elmqvist, Do, et al., 2008)*

2.5.2. Structural Navigation

Vertical Structural Navigation

Vertical navigation in networks is similar to *zooming* on the view level in a user interface; a low zoom level gives an overview, while a high zoom level shows the details. At a low zoom level, graph structures are aggregated into meta nodes and meta edges. Sarkar and Brown (1992) were the first to apply the fisheye technique (Furnas, 1986) to graphs by taking into account the graph's structure. The size of nodes does not depend on the spatial screen distance to the focus, but on topological distance. Fisheye techniques for graphs have then been developed that aggregate nodes which are far from the fisheye focus. Schaffer et al. (1998), Gansner, Koren, and North (2005) as well as Abello et al. (2004) and van Ham and van Wijk (2004) calculate a hierarchical clustering on the graph and use fisheye lenses to open nodes that are close to the focus node. Figure 2.16 illustrates the usage of such a *compound* (Abello et al., 2004) or *semantic fisheye* (van Ham & van Wijk, 2004). The resulting graph shows nodes from different levels of detail in one graph (Figure 2.16(b)). Ask-Graph (Abello, van Ham, & Krishnan, 2006) is a comprehensive graph navigation and exploration system, allowing for interactively opening and closing clusters. It builds on previous work by Abello and van Ham (2004) and Abello et al. (2004).

In a similar approach, Abello and van Ham (2004) use matrices to navigate hierarchically clustered graphs and let users interactively open and close nodes in a cluster hierarchy, shown on the side as done later in Ask-Graph. The coloring of matrix cells reflects the number of edges between nodes or meta nodes. Abello and van Ham also present algorithms to manage very large graphs, by keeping only a part of the graph in memory and reloading parts of the graph from hard disk on demand. Also using an adjacency matrix, Elmqvist, Do, et al. (2008) do not pre-calculate the clustering hierarchy, but nodes get aggregated as the users zoom in. When the user zooms out, two nodes

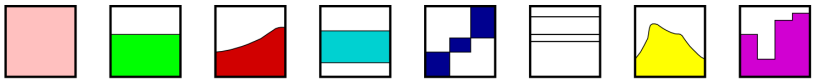


Figure 2.19.: *Glyph designs in ZAME to visualize and aggregate attributes on edges (Elmqvist, Do, et al., 2008)*

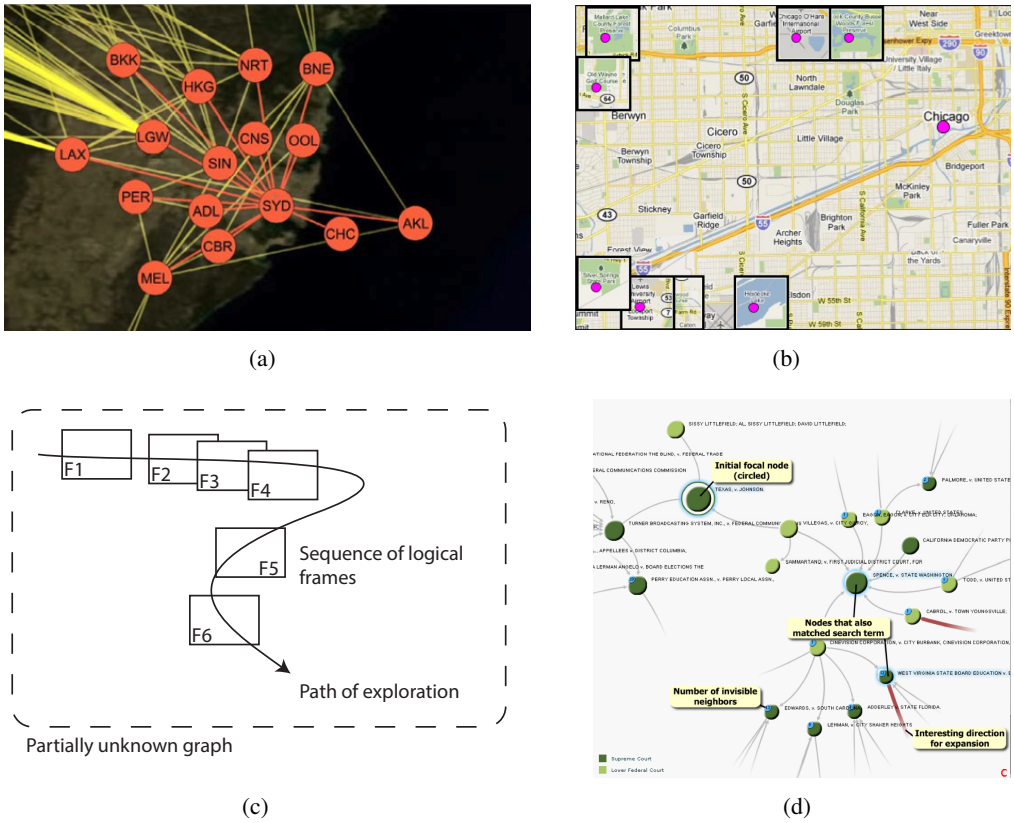
being neighbors in the matrix row and column ordering become aggregated into a meta node (Figure 2.18), while matrix cells show aggregated information about the underlying edges in a single value. Elmqvist, Do, et al. (2008) further proposed embedded glyphs to abstract distributions or show trends (Figure 2.19).

Horizontal Structural Navigation

Similar to panning, horizontal navigation brings other parts of the graph into focus, but remains at the same level of detail. It is also referred to as *incremental navigation* (Herman et al., 2000). Horizontal navigation is for example required to find neighbors of a node in large graphs, and/or travel along paths. Furthermore, many network exploration tasks require revisiting graph elements (Lee et al., 2006), i.e. switch back to a node or group that have already been observed. If edges in the graph span large distances and nodes of interest are far from each other, free panning is less useful. However, the network's edges can guide the user like hyperlinks in document collections. Moscovich, Chevalier, Henry, Pietriga, and Fekete (2009) describe two such techniques for network visualization: *Bring-and-go* interactively brings related nodes close to the focus node (Figure 2.20(a)) and *link sliding* attaches the mouse cursor to a link and enables to slide along the link to the target node. Similar to *bring-and-go*, Ghani, Riche, and Elmqvist (2011) use *dynamic insets*, which are small portals that show the neighbors to a selected node on the margins of the screen (Figure 2.20(b)).

The principle of horizontal navigation also helps to navigate graphs that do not fit entirely in memory. Users start by a single node or a subset of nodes and their immediate neighbors, then subsequently, further neighbors of the neighbors are loaded and displayed. Huang, Eades, and Cohen (1998) call the current visible graph a *logical viewing frame* (Figure 2.20(c)). Whenever the user clicks a node (a website), the viewing frame is moved and nodes that are no longer part of the frame are removed from the display. Nodes that have been clicked by the user always remain visible, resulting in a history trail, similar to Nestor (Eklund, Sawers, & Zeiliger, 1999).

Hyperbolic layout shows logical viewing frames in a circular tree layout and usually shows more than the direct neighborhood. Hyperbolic layouts are calculated from the spanning tree, rooted in the selected nodes given each node the same importance, depending only on the distance to the focus node (Yee, Fisher, Dhamija, & Hearst, 2001; Lamping, Rao, & Pirolli, 1995; Munzner, 1998).



(a) (b)

(c) (d)

Figure 2.20.: *Techniques for horizontal graph navigation. (a) Bring and Go (Moscovich et al., 2009), (b) Dynamic Insets (Ghani et al., 2011), (c) Schematic view of Logical frames, illustration after Huang, Eades, and Cohen (1998), (d) Degree of Interest graph visualization by van Ham and Perer (2009).*

van Ham and Perer (2009) extend Furnas' notion of *degree of interest* (DOI) function to graphs. If a node is shown depends on its *a priori importance*, e.g. its relevance with respect to a user-posed query, and on the length of the shortest path to a selected focus node. van Ham and Perer call their approach "Search, Show Context, Expand on Demand", in analogy to "Overview first, zoom and filter, detail on demand".

Navigating individual links between nodes can be time intensive. Facet Graphs (Heim, Ertl, & Ziegler, 2010) is a solution for *set-based browsing* that retrieves *all* neighbors of a focus set of nodes. Neighbors are grouped by type and can be used for further navigation. GraphTrail (Dunne, Henry Riche, Lee, Metoyer, & Robertson, 2012) is similar to Facet Graphs but embeds visualizations in each set. Visualizations can show bar charts, world clouds or matrices. Users can filter elements using the visualizations and refine the other sets visible on the screen.

2.5.3. Temporal Navigation

Temporal navigation is only required in dynamic networks, unless users manipulate a static graph and require to navigate the history (Heer, Mackinlay, Stolte, & Agrawala, 2008). Similar to navigation in static networks (see Section 2.5), navigation in dynamic networks refers to moving the user with respect to the data and/or visualization, in time. That is, rather than viewing the graph at different levels of detail (horizontal navigation), or different parts of the graph (vertical navigation), temporal navigation changes the temporal focus and granularity. Changing the temporal focus means navigating along the time axis and corresponds to a *horizontal* navigation, while changing granularity corresponds to a *vertical* navigation that changes detail of the displayed data.

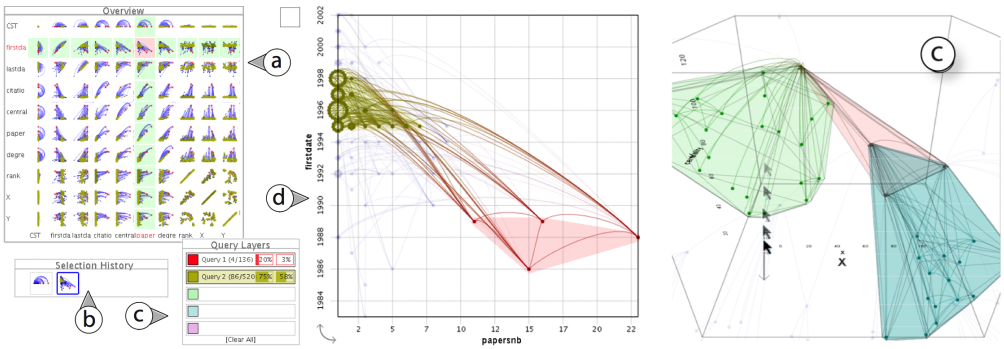
Horizontal Temporal Navigation Horizontal temporal navigation changes the hour, day or current time step, usually using time sliders to navigate animated sequences of the dynamic network (e.g. SoNIA (Moody et al., 2005), TempoVis (J.-W. Ahn et al., 2011), and Gephi (Bastian et al., 2009)). Other techniques originate from video interfaces with play, pause, forward and backward controls. Horizontal temporal navigation with sliders is mostly linear, unless the user can move the slider head instantly to a particular time point, which causes an abrupt change in the graph sequence. A detailed description of techniques to animate changes within a sequence of graphs is given in Section 2.6.2.

Vertical Temporal Navigation Vertical temporal navigation sets the temporal granularity of instants (see Section 2.3.4). Gephi and TempoVis allow the user to define a time window and to interactively set its start and end time, while the window is moved along a time slider.

2.5.4. Navigation in the Visualization

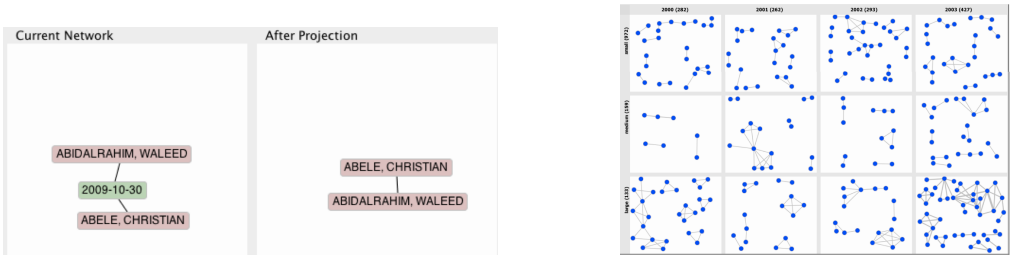
In visualization navigation, the user navigates between different *visualizations* on the graph, i.e. it changes the way the graph is represented. A visualization is a representation of the graph and can offer alternative views on the graph and its elements. Techniques for visualization navigation are much less intuitive to design than for structural and temporal navigation. It requires overviews and metaphors to help users orient themselves. GraphDice (Bezerianos, Chevalier, Dragicevic, Elmqvist, & Fekete, 2010) is an example that visualizes homogeneous, multivariate, and dynamic graphs by locating nodes in a scatter plot, where axes represent attribute dimensions or time. Edges between nodes are shown on top of the scatterplot. A scatterplot matrix serves as overview and interaction widget, animated transitions change axes in the scatterplot as described in ScatterDice (Elmqvist, Dragicevic, & Fekete, 2008).

Ploceus (Liu et al., 2011) is a system that allows users to create node-link representations of huge homogeneous graphs, where each view shows a different subset of nodes and edges, filtered by type. To define a view on the data, Ploceus provides a widget



(a) (b)

Figure 2.21.: *View navigation in GraphDice (Bezerianos, Chevalier, et al., 2010). (a) Scatterplot Matrix for selecting views. (b) View transition in the scatterplot using staged animated transitions described by Elmqvist, Dragicevic, and Fekete (2008).*



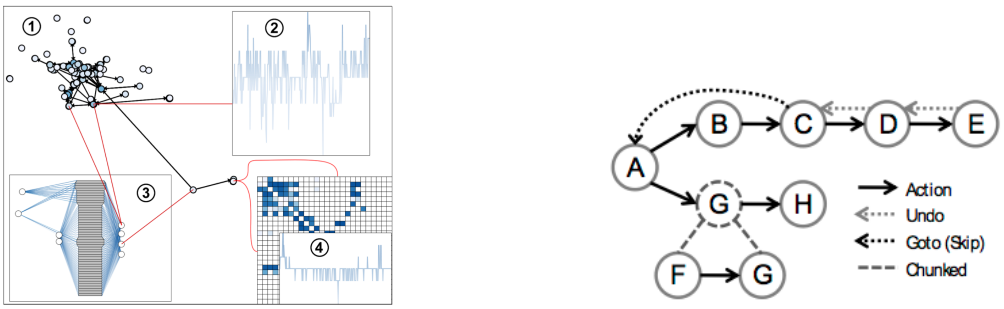
(a) (b)

Figure 2.22.: *View creation and overview in Ploceus (Liu et al., 2011). (a) Model view to define which nodes (according to node type) and relations should be visible. (b) Multiple views, each showing different nodes and times. Here, columns indicate years, while rows indicate domain specific node attributes.*

(Figure 2.22(a)) that allows users to define which node and edge types should be shown and how they are related. For example, Ploceus can show indirect relations between nodes that are not related directly connected. Views in Ploceus change instantly without any transitions, but can be displayed side-by-side for comparison (Figure 2.22(b)). To visualize and explore dynamic networks, Hadlak, Schulz, and Schumann (2011) change representations individually for subgraphs (Figure 2.23(a)). Representations are nested, similar to NodeTrix (Henry et al., 2007) or nodes are duplicated and the new visualization is shown in a portal window.

2.6. Visualizations for Dynamic Networks

While research on dynamic networks involves the same areas as visualization for static networks (Section 2.4), it faces additional problems:



(a) (b)

Figure 2.23.: *(a) Insitu graph views by Hadlak, Schumann, Cap, and Wollenberg (2013) and (b) navigation and interaction history by Heer et al. (2008).*

- **Dynamic graph layout:** finding a graph layout that does not change too much with the topological changes in the network over time,
- **Time-varying attributes:** design visualization techniques that encode time and changes to graph elements and attributues,
- **Topology and Time:** How to show topology and time at the same moment.
- **Temporal navigation:** enable navigation in time (horizontal and vertical).

In order to describe visualizations for dynamic networks, Hadlak et al. (2011) propose two orthogonal dimensions: *network structure* and *time*, illustrated in Figure 2.24[2]. In any visualization, network structure and/or time can be shown *unreduced*, *reduced* or *abstracted*. *Reducing* space or time focuses on a subgraph or set of time points (horizontal structural and temporal navigation), while *abstracting* removes detail from either dimension. *Unreduced* structure and time means that the entire network is visible for every time step. The taxonomy by Hadlak et al. supports comparison compare visualizations and to discuss the tradeoff between abstracting and showing time or structure. The taxonomy clarifies *what* a visualization shows, but does not inform *how* information is (or can be) shown, i.e. how time and space are abstracted.

A survey on dynamic networks is yet missing. In this section, I introduce a simple typology of visualizations, according to *how* they visualize time, topology, and attributes. The typology is based on what are the visual means that encode time and topology. Later in this dissertation (Chapter 7), we define a more detailed classification that can can serve as a design space and allows to better describe, discuss and design new visualization techniques.

- **Temporal Multiples:** Graph snapshots are shown side by side on the screen.
- **Animated Sequences:** Snapshots of the graph are shown in a sequence, often employing animations between them.
- **Temporal Aggregations:** Shows a supergraph of the dynamic network and encodes time explicitly using visual variables other than *position* (Bertin, 1973).

[2]Numbers in the figure refer to references in the paper by Hadlak et al., 2011.

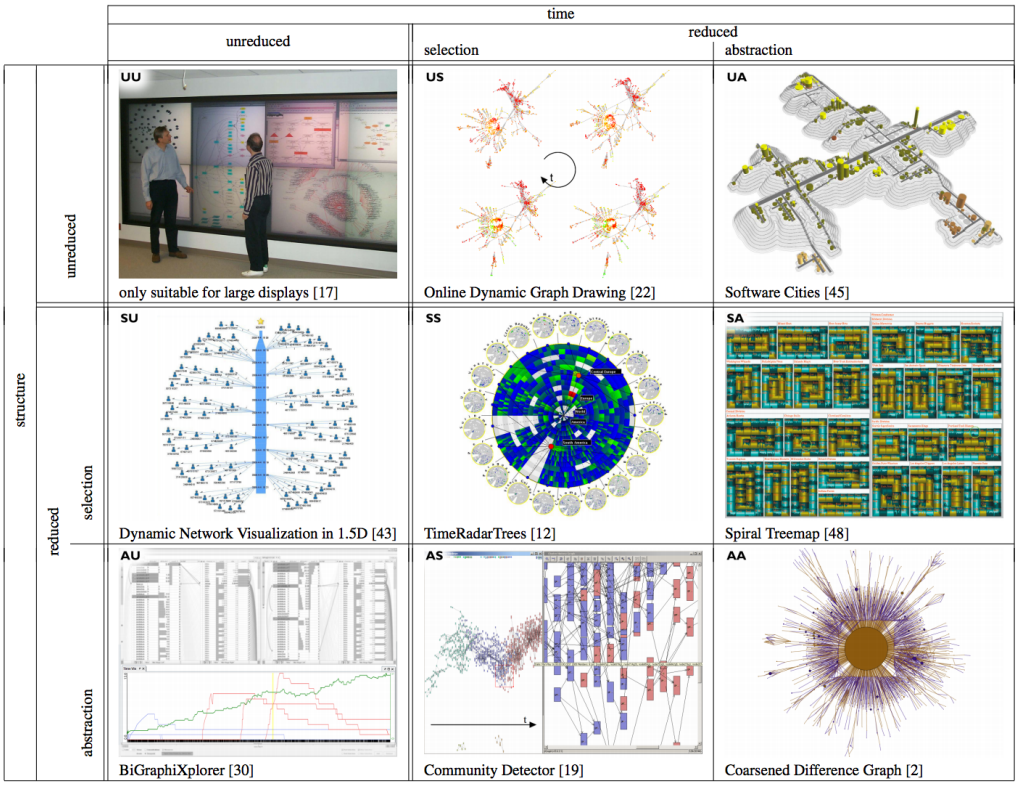


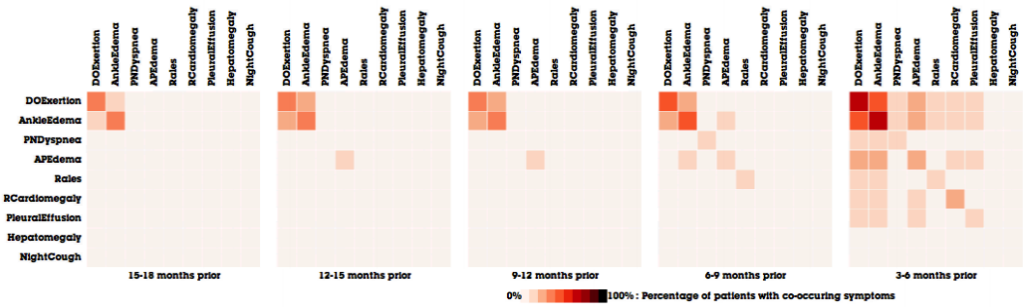
Figure 2.24.: *Taxonomy for dynamic tree and network visualizations as described by Hadlak et al. (2011). Visualizations are grouped according to whether they abstract time (columns) or network structure (rows).*

- **Nesting:** Changes of a node or edge attribute over time are shown in the corresponding glyphs in topological representations (node-link or matrices).
- **Timelines:** Time is shown along one spatial dimension, while the remaining screen space is used to convey network structure.
- **Ego networks:** Shows the evolution of a single node's neighborhood.
- **Space-Time Cube representations:** 3D representations of dynamic networks as space-time cubes.

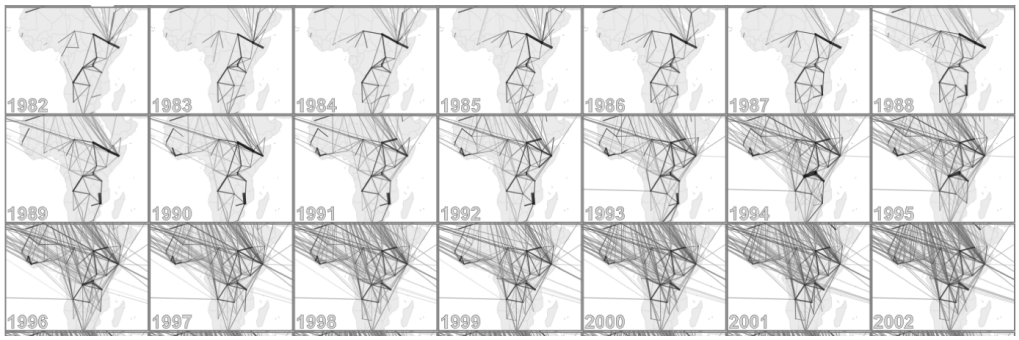
In the following, I discuss visualizations in all of these groups and afterwards will refer to the few comparative studies that exist. Table 2.2 summarizes the visualizations discussed in this section.

2.6.1. Temporal Multiples

Temporal multiples show the network at every time step, side-by-side on the screen. This technique is often called *small multiples*, but the term small multiples does not refer to time explicitly. Technically, temporal multiples is the simplest technique to represent dynamic networks. One image per time step can be created with any common graph



(a)



(b)

Figure 2.25.: *Examples of temporal multiples. (a) Matrices juxtaposed in Matrix Flow (Perer & Sun, 2012). The line chart shows the evolution of connection weigth between two selected nodes. (b) Design for a user study on temporal multiples in Boyandin, Bertini, and Lalanne (2012).*

visualization systems for static networks. Pictures can then be shown in parallel using any kind of (multi-view) image viewer. Ploceus (Liu et al., 2011) can produce and lay out temporal multiples interactively (Figure 2.22(b)). More recently, Perer and Sun (2012) use adjacency matrices in MatrixFlow (Figure 2.25(a)). Cell darkness indicates strength of relation. When the user hovers over a cell in a matrix, the evolution of the connection's weight is plotted in a line graph below the matricess. Matrices benefit from the advantage that, if properly placed on the screen, rows or columns align and connections of nodes can be tracked along rows across matrices (time steps). Figure 2.25(b) originated from a study that compares animated sequences and temporal multiples, (Boyandin et al., 2012). Within these temporal multiples, users are able to synchronously zoom the views and highlight edges across them. Line thickness indicates edge weight.

Temporal multiples provide a quick overview of the network's evolution without hiding or abstracting any information. Yet the size of multiples depends on their number and on the available screen space; larger images show more details of the network, smaller images mean that more time steps can be shown simultaneously. Another problem is the comparison of distant times and to track particular graph elements across time steps. Temporal multiples therefore hardly allow for a detailed analysis of changes, as those

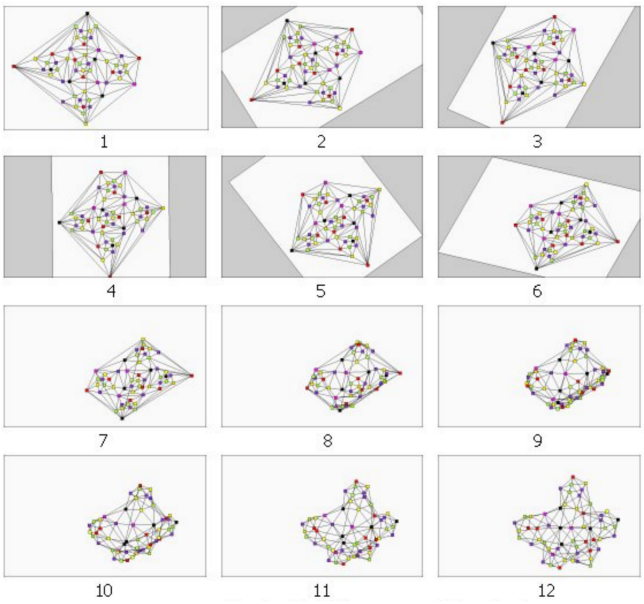


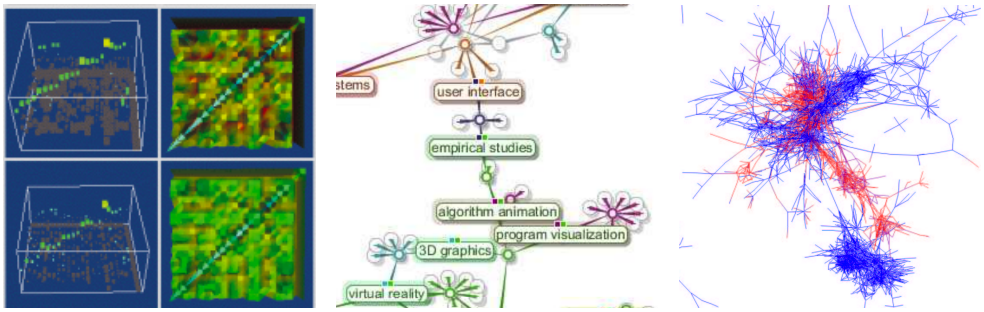
Figure 2.26.: *Example of animation in Friedrich and Eades (2002) employing geometric transformations before moving nodes to their final positions. Images 1-6 indicate affine transformations (rotation, scaling, shearing, translation), images 7-12 show nodes moving to their individual positions.*

require direct comparison across multiple small images and do not scale to large networks or networks with many time steps. Letting users change the temporal granularity (vertical temporal navigation), or allow for brushing-and-linking, could perhaps help to overcome some of these problems. To the best of our knowledge, such techniques have not been implemented yet.

2.6.2. Animated Sequences

Sequences of the network's time steps is by far the most common visualization approach. In order to convey changes between time steps, animations show node and edge addition and removal as well as move nodes to their positions in the new layout. The simplest animations interpolate between the position of nodes, possibly using fast-in-fast out pacing, while fading in new nodes and edges and fading out nodes and edges which get removed from the graph (Diehl & Görg, 2002; Erten et al., 2003; Friedrich & Eades, 2001; Moody et al., 2005; Bender-deMoll & McFarland, 2005). Crucial to any such animation is the amount of change in node positions, which is directly related to the layout used. While in Gephi (Bastian et al., 2009), each time the user navigates through time, the force-directed layout gets updated continuously, techniques exist to stablize the layout, discussed in detail in Section 2.7.

Although direct interpolation between node positions is easy to understand, nodes can cross each other during animation. Friedrich and Eades (2002) described staged



(a) (b) (c)

Figure 2.27.: *Examples of aggregated dynamic networks. (a) Matrices in information visualization spreadsheets (Chi, Riedl, Barry, & Konstan, 1998), (b) superimposed node-link diagrams in Hascoët and Dragicevic (2012), (c) nodes colored by age (Collberg, Kobourov, Nagra, Pitts, & Wampler, 2003), and (d) line graphs showing time series on network nodes (Hadlak et al., 2013).*

animations employing general geometric transformations to the graph in order to make the current layout appear as close to the new one as possible, without changing node positions relative to each other (Figure 2.26). The employed stages are as follows: (1) removing network elements, (2) transforming the entire network by rotation, scaling, and translation, (3) moving each node individually to its final position, and finally (4) showing new network elements. While added nodes grow or fade-in and removed nodes shrink or fade-out, links are not animated. Although node movements can be tracked, it is difficult to track many changes that happen to the topology, especially for large networks made of unconnected components. Visone (Brandes & Wagner, 2003) features three-stage transitions: first fade out nodes and their incident edges; then remove and add edges between nodes that remain in the network while also updating nodes' positions; and finally add new nodes and their incident edges. In TempoVis (J.-W. Ahn et al., 2011), nodes do not change positions during the animation and intensity of node color diminishes depending on the node's age, i.e. its time in the network.

Animated sequences allow to use the entire screen space to show a single time step, while encoding changes implicitly as change between two time steps. Sequences can scale up to hundreds and thousands of time steps, yet users must memorize the network at earlier time steps and navigate though time. Studies on the efficiency of animated sequences are discussed together with other studies in Section 2.8.

2.6.3. Temporal Aggregation

Temporal aggregation represents the entire dynamic network into a single picture of the network without duplicating elements, (the supergraph) while encoding:

- **values** over time (such as the average number of connections) (*value encoding*), or

- **time** (when did a particular node first appear in the network) (*time encoding*).

Value Encoding *Value encoding* is very common and does not require sophisticated visual encodings. For example, Chi et al. (1998) color cells in a matrix according to the frequency of connections and propose to use the third spatial dimension to highlight very frequent connections. However, all information about precise time points or distribution over time, is lost. Similarly, NetVisia (Gove et al., 2011) shows adjacency matrices with varying cell saturation, and Honeycomb (van Ham, Schulz, & Dimicco, 2009) summarises the number of edges between any node pair over time in a single brightness value per matrix cell.

Time Encoding *Time encoding* assigns each time step a particular encoding, such as a color hue value to encode, for example, the first appearance of nodes and edges in the entire network (Collberg et al., 2003). Difference graphs can be a particular form of time encoding if both graphs are still distinguishable as in Andrews, Wohlfahrt, and Wurzinger (2009). Hascoët and Dragicevic (2012) as well as Erten, Kobourov, Le, and Navabi (2004) compare multiple networks by coloring each network individually and *superimposing* the networks.

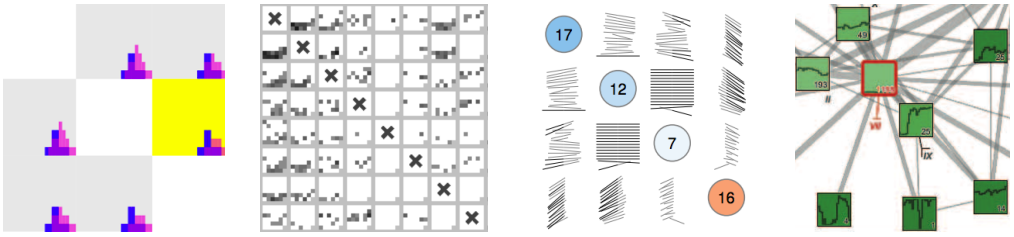
Aggregated images preserve the topology and show where the graph is affected over time. While value encodings may still yield readable images, much of the temporal information is lost. Time encoding can show more temporal information, also it can increase the visual complexity of the network representation. Difference graphs work well when comparing few graphs, but are insufficient to explore long sequences of time steps.

2.6.4. Nestings

Nestings represent changing values for node or edge time attributes over time, inside glyphs for graph elements, while the topology of the network remains visible.

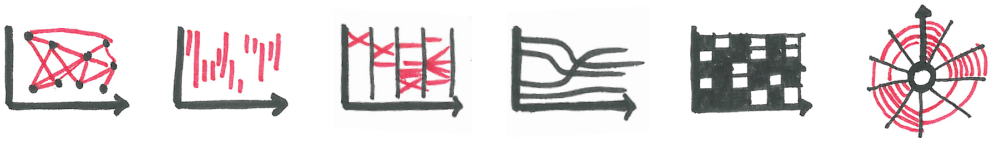
Matrices have been used more often for nesting temporal information on edges, likely because cells provide more space than lines in a node-link diagram, for encoding information. To show the same information, Yi, Elmqvist, and Lee (2010) (Figure 2.28(a)) use bar charts inside matrix cells, similar to ZAME (Elmqvist, Do, et al., 2008). Stein, Wegener, and Schlieder (2010) employ a space filling algorithm to map temporal evolution of connection activity betwen nodes into the matrix cells (Figure 2.28(b)). Brandes and Nick (2011) use *Gestaltlines* inside matrix cells to represent changing edge weight over time (Figure 2.28(c)). Time runs from bottom to top while length and inclination of the horizontal indicates weight in both directions of the relation.

While matrices show nestings inside their cells, node-link diagrams have been used to embed time-series on nodes, using line graphs Saraiya, North, and Duca (2010) and Hadlak et al. (2013) (Figure 2.28(d)).



(a) (b) (c) (d)

Figure 2.28.: *Examples of aggregated dynamic networks. (a) Time Matrix showing edge time-varying attributes as bar charts inside matrix cells (Yi et al., 2010), (b) Pixel-oriented visualizations using space filling curves to indicate time-varying edge attributes (Stein et al., 2010), (c) Gestaltlines in matrix cells indicating weight changes in bi-directed weighted networks (Brandes & Nick, 2011)*



(a) (b) (c) (d) (e) (f)

Figure 2.29.: *Techniques to visualize dynamic networks on a timeline. The timeline goes horizontally from left to right, and nodes are shown on the vertical axis (topology axis). Red glyphs indicate connections. (a) Node-link based, (b) Arc-diagram based, (c) Parallel edge splatting, (d) Flow chart, (e) Heat map, (f) Polar layout*

Nestings can show changes in edge values without necessarily hiding information. They can show trends in attribute values for a single graph element as well as allow for comparison of these trends. Since the network topology is shown explicitly, trends can easily be related to topology. However nestings can be very detailed, especially if they show information for many time steps. The problems are similar to those with temporal multiples: (i) as the network gets large and contains many time steps, screen space might not be sufficient to show all nestings in sufficient detail; (ii) perception of one particular time step is difficult, as well as (iii) the comparison of time steps.

2.6.5. Timelines

Timeline visualizations for dynamic networks consist of a timeline along one spatial dimension, while network structure is displayed using the remaining spatial dimension, for each time step. Timeline views are useful because they allow to estimate the exact sequence and duration of events, as well as allow to compare durations at different locations in time. The main problem in time line visualizations is to find a node ordering that keeps the chart readable and creates patterns among connections in the chart. The other problem is to deal with the resulting over plotting of connections.

Different techniques have been proposed to address one or both of these problems; some duplicate nodes, some use fixed node position, some do show and other do not show connections. In the following I group the visualizations into the the following (non-exclusive) types, illustrated in Figure 2.29:

(a) **Node-link based**: no node or edge duplication, edges connect nodes placed along time;
(b) **Arc-Diagram based**: nodes have fixed vertical positions, and connections are shown as arcs or lines between horizontal lines;
(c) **Edge splatting**: node duplication;
(d) **Flow Charts**: node order change in the topological dimension;
(e) **Heat-maps**: show node attributes per time step;
(f) **Polar layout**: Use a polar layout to indicate time.

Node-link based Node-link based techniques are basically node-link diagrams but nodes are laid out according to time (Figure 2.29(a)). There is no node or edge duplication. For example, Falkowski, Bartelheimer, and Spiliopoulou (2006) first calculate a layout for the entire super graph, and then show time along a vertical axis, while keeping the y-coordinates of nodes (Figure 2.30(a)). A node's position along the time axis indicates the time point at which it joined the network. Communities in the graph are calculated in advance and their coloring in the timeline view reveals the evolution of each community. GraphDice (Bezerianos, Chevalier, et al., 2010) shows nodes located in a scatterplot where one axis shows time (Figure 2.30(c)). Since every node appears only once, relations between different nodes can clutter the visualization and are hard to follow. Semantic Substrates (Shneiderman & Aris, 2006) solves the problem of visual clutter by showing links only on users' demand, as they hover a particular node (Figure 2.30(b)).

Arc-Diagram based Arc-Diagram based timelines solve the clutter problem by assigning each node a unique position on the vertical dimension (given that time is represented on the horizontal screen dimension). Connections are represented as arcs or lines between these horizontal layers (Figure 2.29(a)). Greilich, Burch, and Diehl (2009) represent connectivity between nodes in a tree and use nodes' positions in the tree as order (Figure 2.30(e)).

Yet, this ordering might not be optimal to reduce clutter in the diagram. van den Elzen, Holten, Blaas, and van Wijk (2013) propose several ordering mechanisms on *Massive Sequence Views* including one that minimizes lengths of arcs in the diagram (Figure 2.30(d)). Many timeline visualizations use discrete time steps and show *multiple* connections at the same time step, massive sequence views show only one connection per step on the time axis. Further orderings by van den Elzen et al. therefore include mechanisms to minimize block overlap, order nodes by degree measures, and according to when nodes are connected for the first time. Eventually, ordering techniques can be combined in order to produce specific pre-attentive visual patterns in the chart.

Parallel Edge Splatting Parallel Edge Splatting (Burch et al., 2011) also uses fixed positions on the topology axis, while showing time steps on the time axis (Figure 2.29(c)). Edge over plotting is avoided by introducing gaps between time steps and showing edges between nodes in adjacent time steps, looking similar to a parallel coordinates plot (Figure 7.3(b)).

Flow Charts Other approaches abstract even further by removing edges and instead letting nodes change their position on the topology axis over time (Figure 2.29(d)). Node positions are re-calculated for every time step, similar to calculating a new layout for every time step in an animated sequence (Section 2.6.2). Reda, Tantipathananandh, Johnson, Leigh, and Berger-Wolf (2011) show node groups (defined by attribute value) as horizontal layers with fixed positions, while individual nodes are shown as lines and switch between these groups (Figure 2.30(g)). Sallaberry, Muelder, and Ma (2013) visualize only nodes as lines and find an ordering for every time step (Figure 2.30(h)). Given a "good" ordering, these flow charts can show the evolution of clusters.

Heat maps Heat maps do not show any connections either, but summarize node attributes over time, i.e. topological information is completely lost (Figure 2.29(e)). For example, in NetVisia (Gove et al., 2011) rows indicate nodes, columns indicate time points and cells encode specific network topology metrics for nodes (Figure 2.30(j)). Similar to matrices, rows in the heatmap can be ordered according to value similarity. Heatmaps in NetVisia are coupled with an adjacency matrix that shows an aggregated picture of the network structure for a certain, user defined, time period.

Polar layout A common problem in finding a linear node ordering for timeline views is that the connected but distant nodes produce *long* arcs in the diagram. Polar representations avoid long arcs by showing each time as concentric circles (Figure 2.29(f)). van den Elzen et al. represent Massive Sequence Views to such a polar representation and further improve their ordering techniques (Figure 2.30(l)). Polar representations can significantly reduce visual clutter since any arc is at maximum as long as half the circumference of a "time ring". van den Elzen et al. use the rays representing nodes to encode node attributes. Burch and Diehl (2008) show a radial ego network at the end of the each node beam, illustrated in Figure 2.30(k). Burch et al. also describe a polar representation for parallel edge splatting.

Timeline visualizations benefit from the clear mapping of time so space and they can effectively visualize changes in temporal clusters. On the downside, precise topological information is lost due to the linear ordering of nodes that is required, since only one spatial dimension remains to show them. Parallel edge platting and the designs by Reda et al. (2011) as well as Sallaberry et al. (2013) can suffer from crossing lines that result from nodes changing distant communities or edges between distant nodes. Cui et al. (2014) partially overcome this problem by creating a pixelbased flow map, rather than visualizing individual nodes and edges (Figure 7.3(c)).

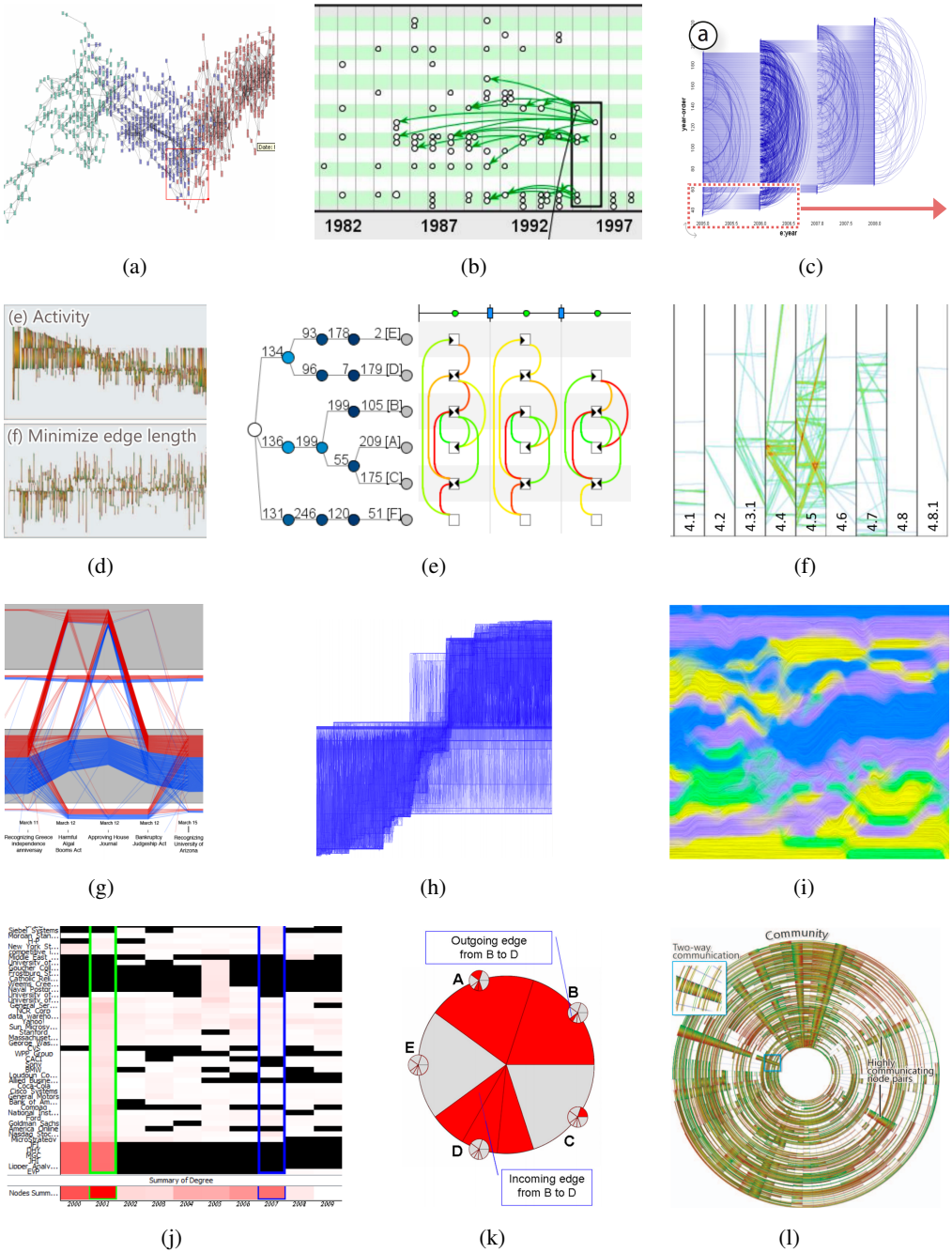
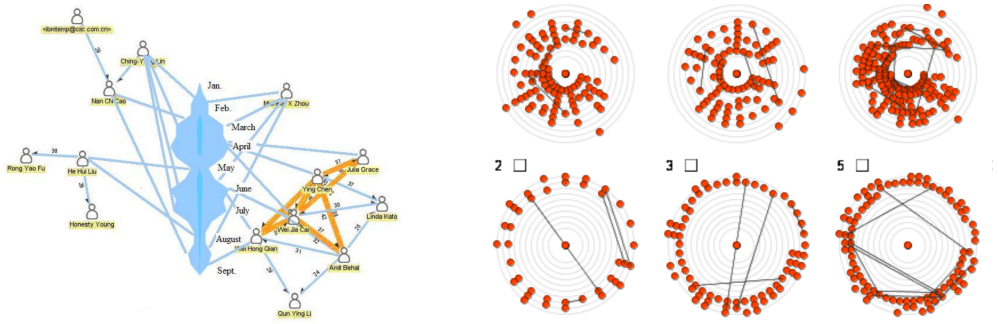


Figure 2.30.: *Examples of timeline visualizations for dynamic networks. Time runs from left to right, topology is abstracted in the remaining spatial dimension. (a) Falkowski et al., 2006, (b) Semantic Substrates, (Shneiderman & Aris, 2006), (c) GraphDice, Bezerianos, Chevalier, et al., 2010, (d) Massive Sequence views, van den Elzen et al., 2013, (e) TimeArcTrees, Greilich et al., 2009, (f) Parallel edge splatting, (Burch et al., 2011), (g) Reda et al., 2011, (h) Sallaberry et al., 2013, (i) GraphFlow, Cui et al., 2014, (j) NetVisia, (Gove et al., 2011), (k) TimeRadarTrees, Burch & Diehl, 2008, (l) Polar view of Massive Sequence Views, (van den Elzen et al., 2013).*



(a) (b)

Figure 2.31.: *Dynamic ego networks: (a) central node expanded to timeline (Shi, Wang, & Wen, 2011), (b) ego networks placed side-by-side (Farrugia, Hurley, & Quigley, 2011),*

2.6.6. Dynamic Ego Networks

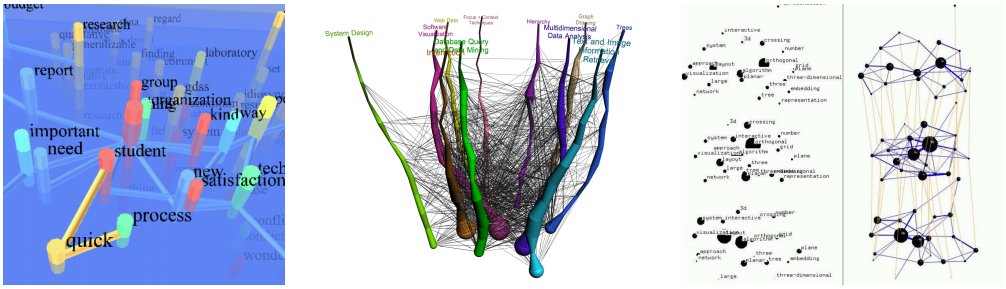
Dynamic ego networks capture changes to the neighborhood of a single node, called the *ego*. This reduction of the displayed information allows for some more freedom in arranging visual elements on the screen and encoding information about time and attributes.

Shi et al. (2011) use a timeline to represent the activity of the ego, while placing related nodes around the timeline. Whenever a node connects with the focus node, a link is drawn between the neighbor and the position on the timeline that corresponds to the time of connection. Edges between neighbors are shown regardless of the time point (Figure 2.31(a)). Farrugia et al. (2011) lay out the ego's neighborhood in a radial manner, while neighbors are placed on concentric circles, whose radius increase with time point of connection. This concise representation allows to visualize multiple ego networks at the same time (Figure 2.31(b)) and compare the individual connection behaviour.

While dynamic ego networks work well to observe connectivity changes in the neighborhood of an individual node, it does not allow to visualize higher-level graph structures such as clusters or paths.

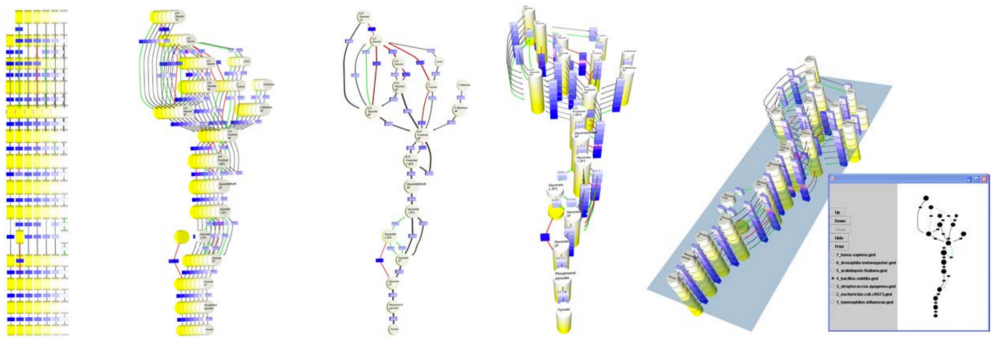
2.6.7. Space Time Cube Representations

Space time cubes have been used in many domains to visualize temporal data that intrinsically has to be shown in 2D, such as geographical data (Carpendale, Cowperthwaite, Tigges, Fall, & Fracchia, 1999; Kraak, 2003) or videos (Mackay & Beaudouin-Lafon, 1998; Fels, Lee, & Mase, 2000; Daniel & Chen, 2003). While two dimensions represent the spatial data, the third spatial dimension conveys time. Since network structure can be mapped to a 2D layout, approaches have been made to extrude node-link diagrams into 3-dimensional space. Dwyer (2004) defines this type of 3D visualisation as 2.5D,



(a) (b) (c)

Figure 2.32.: *Examples of space-time cube visualizations for dynamic networks. (a) Stacked node-link diagram with same layout for all time steps (Brandes & Corman, 2003), (b) stacked node-link diagram, with different layout per time steps, bending nodes to "worms" (Ahmed et al., 2005), (c) layers in GraphAEL (Erten et al., 2003)*

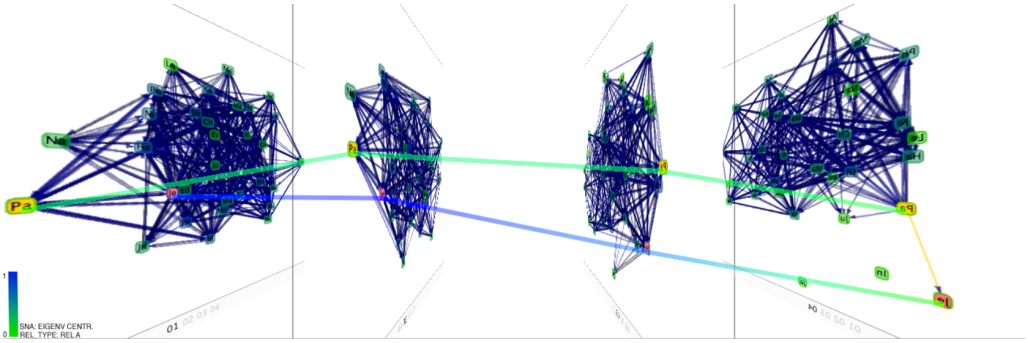


(a) (b)

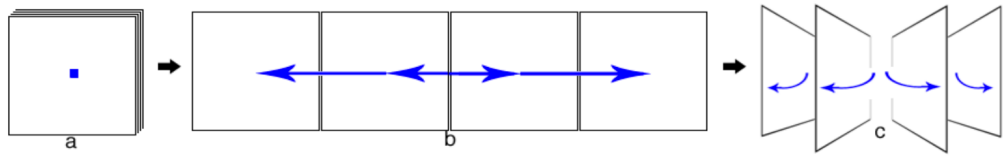
Figure 2.33.: *Views on the 3D map of different pathways in Wilmascope (Brandes, Dwyer, & Schreiber, 2004). (a) Orthogonal projections, (b) perspective projections with cutting plane and detail view of the pathway in the cutting plane.*

Nodes become columns (Brandes and Corman (2003), Figure 2.32(a)) or "worms" (Dwyer and Eades (2002), Figure 2.32(b)), while edges are shown as bridges between the extruded nodes. Worms are bent because node positions change between different layouts over time (see Section 2.7). Bending can show if two or more nodes are tightly connected, without showing edges explicitly. However, as nodes can be "arbitrary" curves in 3D space, tangled in many ways, the visualization can quickly get cluttered.

3D visualizations suffer from the common drawbacks of 2D screens, such as perspective distortion and scaling, reduced depth perception and visual occlusion. Space time cubes usually allow for free rotation to overcome the problem of occlusion. Furthermore, *cutting planes* help to highlight individual time steps (Brandes & Corman, 2003; Dwyer & Eades, 2002). In visually comparing biological pathways, stacked to form a space time cube, (Brandes et al., 2004) overcome some of the drawbacks of 3D visualizations by providing two fixed views; one that shows an aggregated view of pathways and another



(a)



(b)

Figure 2.34.: *(a) 3D view and (b) transitions between views in the system by Federico, Aigner, Miksch, Windhager, and Zenk (2011).*

view showing the space time cube "from the side" (Figure 2.33). The small window in Figure 2.33(b) shows the currently selected pathway in its 2D layout. GraphAEL (Erten et al., 2003) also combines several views on the dynamic network; besides temporal multiples, it provides animated sequences, difference graphs and a 3D view similar to Dwyer and Eades (2002) that shows edges connecting nodes in different time steps.

Similarly, Federico et al. (2011) propose a system that integrates views of multiple types: 2.5D view (space time cube, Figure 2.34(a)), juxtaposition (temporal multiples), superimposition (aggregation), and animated sequences. Animated transitions help users understand transitions between views (Figure 2.34(b)). Federico et al. implemented interaction support to show node *traces* (the positions of a node though time).

While static 2D network visualization is already challenging due to node overlap and link crossing, adding a third dimension usually increases the visual complexity. Yet, while they may be hard to read and less useful for a precise analysis, they have a great potential to explain a data set or visualization.

2.7. Dynamic Graph Layout

Most of the techniques in the last section require some sort of graph layout. Layouts for static graphs have been summarized in Section 2.4.1.

While aggregated visualizations of dynamic networks require only a single layout for the graph, animated sequences or temporal multiples can show a different layout for each time step (*local layout*). Local layouts can optimize the layout for each time step individually, improving readability of the topology. However, local layouts can change significantly between two time steps, depending on the number of nodes and edges being added or removed with respect to the previous layout. The problem of dynamic graph drawing is to find a local layout for each time step that minimizes changes in node positions, while allowing for a *good* layout at every time step. This problem is known as the *Mental map preservation problem* (Eades, Lai, Misue, & Sugiyama, 1991; Misue, Eades, Lai, & Sugiyama, 1995). In order to minimize visual changes between time steps, layout algorithms use information about node positions across time steps. Such algorithms use either previous time steps only (*online*), or previous and future time steps (*offline*). Conceptually, the mental map preservation problem exists as well for finding an optimal node ordering for matrices (Yi et al., 2010; Brandes & Nick, 2011) or vertical lines (Reda et al., 2011; Burch et al., 2011). Yet, to the best of our knowledge, no work has been done in this direction.

2.7.1. Mental Map Preservation

The *mental map* is described as the users' internal view of a visualization or network. It relates visual elements and patterns to their meaning and is required to understand the visualization, to remember it, and to use it for reasoning to solve tasks. Changes to the visualization, caused by interaction or system updates, require the user to update his mental map.

In order to convey changes across time steps and to help users updating their mental map, animated transitions have been proven useful (Heer & Robertson, 2007; Robertson, Fernandez, Fisher, Lee, & Stasko, 2008) (cf. Section 2.6.2). However, in the worst case, the visualization changes so much that users can not follow the animations and may lose their mental map. Loosing the mental map means not being able to relate the current (state of the) visualization, to the knowledge the user got from it previously. The user must read and understand the visualization again, while relating it to his new mental map. Independent from using animation or not, a stable layout which minimizes changes between time steps, is supposed to support users in maintaining their mental map, when navigating through time steps and working with the visualization.

Similar to aesthetic graph drawing criteria for static graphs, criteria for dynamic graph layouts exist. Misue et al. (1995) suggest (i) to preserve horizontal and vertical node ordering in succeeding layouts (*orthogonality*), to (ii) keep relative distances between neighboring nodes (*proximity*), and (iii) to preserve the dual graph, that is to avoid crossing lines that did not cross in previous layouts (*topology*). Based on these criteria, measures to quantify the *mental distance* between two layouts have been described (Bridgeman & Tamassia, 1998; Branke, 2001; Diehl & Görg, 2002).

Different studies investigated the effect of *mental map preservation* techniques on users' task performance, mainly while using animated sequences and/or temporal multiples. None of these studies found conclusive statistical evidence about which degree of stabilization is the best. Purchase, Hoggan, and Görg (2007) as well as Archambault, Purchase, and Pinaud (2011a) could not find any significant difference between three degrees of stabilization (low, medium, and high), although, depending on the task, either medium or high stabilization caused fewer errors. Purchase et al. suggests that mental map preservation is more important to track nodes, but is less required for tasks about edges. In a follow-up study, Purchase and Samra (2008) showed evidence that either high or low stabilization caused users to make less errors than when using a medium stabilization, with low stabilization causing significantly less errors than a high stabilization. Study participants reported that stable layouts support tracking individual nodes, and liked the fact that clusters were clearly visible and that low-degree nodes were shown on the margin of the layout. At the same time, they found stable layouts more cluttered and less clear. In contrast, users liked the more clear individual layouts and reported that it was fun to watch fluid animated node movements in the low stabilization conditions[3]. Purchase and Samra suggest that, rather than combining the advantages of high and low mental map preservation techniques, a medium stabilization would combine the worst of both extremes. Depending on the task, the one or the other should be preferred.

While the results are not so clear for the *efficiency* of mental map preservations, Archambault and Purchase (2012) showed that a stable layout improves *memorizablity* in dynamic graph layouts, and report on two features that are important for memorability. Memorizing *static features* such as prominent network motifs (cliques, users, etc) is supported by stable layouts. In contrast, *dynamic features* such as particular node movements or the order of changes, are supported by unstable layouts. Further evidence in favor of mental map preservation was reported in another study by Archambault and Purchase (2013). Users have been asked to find nodes in one time slice and to re-*find* them in another time slice, without being able to going back to the first time slice. The results showed a significant decrease in time and error rate for mental map preservation (i.e. stable layouts). These results could be seen as supportive for choosing a stable or locally optimal layout.

Yet, all these studies have only used a single metric for mental distance (node position preservation (Diehl & Görg, 2002)) but different layout and stabilization techniques, which may account for variations in results. For example, Purchase and Samra (2008), Archambault et al. (2011a), as well as Archambault and Purchase (2013) used the offline algorithm detailed in Erten et al. (2003), while Purchase et al. (2007) used an iterative online algorithm with differing tolerance in node movement. The remainder of this section discusses the most important techniques and their problems for creating layouts

[3]The fact that animations are engaging may account for better subjective results, but not as indicative of for better objective results.

for dynamic networks. Further techniques and references are summarized by Brandes and Mader (2012), Brandes, Freeman, and Wagner (2013), and Frishman and Tal (2007).

2.7.2. Online Algorithms

Online dynamic graph drawing algorithms create a layout for time step t_i based on the layouts of t_{i-x} whereby x can vary between 1 and the number of time steps (S. C. North, 1996; Huang, Eades, & Wang, 1998; Moody et al., 2005). In the most simple case, the layout from the previous time step is incrementally adjusted while taking the newly added nodes and edges into account. In order to minimize changes, new nodes are placed at the barycenters of their neighbors, i.e. minimizing distances to all the neighbors. For directed networks, Moody (2001) suggest to update node positions so that nodes move towards the barycenter of their successors. However, this could cause unnecessary movement.

Online algorithms are simple to implement by reusing existing algorithms and they do not require much computation. They are used in online scenarios (see Section 2.3.1) where the graph needs to be updated, such as when monitoring a network or when the user is allowed to manipulate the network's topology. However, the problem with this approach is that online algorithms can run into local optima and abrupt changes are necessary to keep the layout readable and ready for further changes.

2.7.3. Offline Dynamic Graph Algorithms

Given that all temporal information is present prior to a visual analysis, an offline algorithm can use information from all the time steps. According to Brandes et al. (2013), there are three approaches to offline dynamic graph drawing algorithms: *aggregation*, *anchoring*, and *linking*.

Aggregating is based on a single supergraph of the network, which is used to calculate a single **global layout** for all nodes that are part of the network, over time. Each node has a unique position that does not change between time steps (Diehl et al., 2001; Brandes & Corman, 2003; Dwyer & Eades, 2002). Such static layouts can be sufficient if the network grows, but lead to unusual results in cases where nodes get removed and the network changes a lot during time. (Purchase & Samra, 2008).

Anchoring uses node positions in past and future time steps, or the global layout, as reference to minimize changes between local layouts. Adjustments for a time step can be calculated (a) *independent* for each time step, taking only the difference to the global layout into account, (b) *predecessor dependent*, taking into account the previous and the global layout, (c) *context dependent* taking into account the previous, the global layout and the successor layout into account, and finally (d) *simultaneous adjustment*

which iteratively optimizes time steps. Simultaneous optimization is technically similar to linking techniques.

Linking is based on a supergraph that contains a copy of each node for every time step and introduces (invisible) *inter time step edges* between instances of the same node in adjacent time steps (Erten et al., 2003; Peterson, 2011). Inter time step edges are treated in almost the same manner as actual edges in a force-directed layout algorithm; they prevent nodes from moving too far. Linking does not require a global layout, but nodes can drift over time, which, depending on the case, can yield more optimal flexible layouts, but likely introduces more movement in the graph.

Comparing these three types of algorithms in a benchmark test, Brandes and Mader (2012) found *linking* to produce layouts with less overall differences between time steps. Since *linking* is performance intensive (each node has to be compared against itself, in all times) Brandes and Mader suggest *anchoring* as alternative. Results from studies on the mental map are not easy to interpret since they did not test for different layout algorithms. Quantifying the efficiency of layout algorithms and mental map preservation techniques is hard, given that difference metrics do not necessarily reflect user performance, but moreover, that users' performance depends on a multitude of (independent) factors: the size of graphs, their characteristics, the amount of changes between time steps, the tasks users perform and how the graph is eventually visualized.

Existing layout algorithms are useful only for visualizations that are based on node-link diagrams, but Section 2.6 shows that there are many visualization techniques for dynamic networks. Perhaps even more important than in static graph drawing, visualization of dynamic networks requires creative and purposeful visualization techniques, going beyond the calculation of a graph layout. Once a layout algorithm is chosen, the next question is *how* to present the data, using techniques such as those presented in Section 2.6. Yet, the choice of the visualization technique may have an influence on the layout algorithm to choose. The next section reports on controlled user studies that compare some of the visualization techniques reported in Section 2.6.

2.8. Comparative Studies

Few studies have been conducted that compare different ways of visualizing dynamic networks. Most of them compared animated sequences and temporal multiples ("*small multiples*" in the studies). The results suggest that temporal multiples are significantly faster than animated sequences on a number of tasks, but do not significantly decrease error rate (Farrugia & Quigley, 2011; Archambault et al., 2011a; Boyandin et al., 2012).

Farrugia and Quigley (2011) found temporal multiples to be *faster* and, in most cases, to be *more accurate* than animated sequences. However, several results were not significant. Asked for their preference, participants were much more positive about temporal multiples

than about animated sequences. As participants commented, animations were too slow for solving the time-critical tasks. They liked that temporal multiples allowed for quick access and the possibility to verify results. However, the temporal multiples conditions contained only four images (time steps). Farrugia and Quigley conclude that animated sequences may be the best choice for large networks, because they can use the entire screen and are independent from the number of time steps. They remain positive about animated sequences in general, but suggest better interaction and navigation methods to overcome the shortcomings of animated sequences. They argue that graph movies, such as created with SoNIA (Moody et al., 2005) (which have been used in the experiment) suffer from the inability to "jump" to a particular time point, as well as the ability to quickly preview.

Archambault et al. (2011a) also compared temporal multiples to animated sequences and, in accordance with Farrugia and Quigley, found temporal multiples to be *faster* than animated sequences, for all five tasks. There was no significant difference across error rate, except for tasks related to the appearance of nodes and edges; for these tasks, animated sequences significantly reduced error rate. Archambault et al. suggest to use temporal multiples for topology-based tasks (cf. Lee et al., 2006), and animated sequences for node and edge appearance and disappearance as well as in cases where accuracy is generally more important than task-completion time. However, similar to Farrugia and Quigley, data sets were rather small and consisted of 9 time steps with at most 60 nodes.

A third study, conducted by Boyandin et al. (2012), compared temporal multiples and animated sequences in an open-ended exploratory study on migration flows between countries. Boyandin et al. found that animated sequences helped observing local geographical changes and changes between subsequent years, while temporal multiples were good for long term observations. The authors suggest an integration of both techniques. Conditions in this study included far more nodes (200) and time steps (35) than the studies by Farrugia and Quigley (2011) and Archambault et al. (2011a). In the animation condition, users appreciated that the entire screen was available to show the visualization, while they found the temporal multiples too small.

Some contrary results to the above mentioned studies come from Archambault and Purchase (2013). Their study showed animated sequences to be generally faster and yield less errors than temporal multiples, but without statistical significance. The conditions included 6 time steps and up to 72 nodes.

Zaman, Kalra, and Stuerzlinger (2011) evaluated animations, side-by-side views (temporal multiples) and difference views for the comparison of two layered diagrams (directed acyclic graphs). Difference views showed both diagrams superimposed, highlighting nodes and edges that were in only one diagram. Zaman et al. found difference views to perform better than animations. Animations were also better than side-by-side views. One study condition combined difference views with animation, which resulted in better performance than animation or difference view individually.

(a) (b)

Figure 2.35.: *Experiment conditions in Robertson et al. (2008): (a) traces superimposed, (b) small multiples (one per country).*

Beyond the domain of networks, studies have compared temporal multiples, animated sequences and aggregated images for other data types. For example, Robertson et al. (2008) investigated the effectiveness of animated sequences for trend visualization in scatterplots. Scatterplots showed statistics for country statistics evolving over time, such as in Gapminder (Rossling, 2007). Robertson et al. compare animations to a static image with traces (Figure 2.35(a)) and to temporal multiples (Figure 2.35(b)). Each of the 74 temporal multiples shown in a scatterplot of a single country plus a trace for the countries' movement. The results of the controlled user study showed that animations performed significantly less accurately if users had no control over the animation pace and exact position in the entire movie (10 second duration for 25 years). Being able to control the animation raised task completion time over that of the other two techniques. Although temporal multiples performed better, especially for the condition with a larger data set, animation was the more exciting and encouraging to participants[4]. A more ample discussion of studies outside the scope of dynamic networks is found in Section 7.4.

Except for Farrugia and Quigley (2011), animations were generally rated higher in users' preference, although animations did not perform better in error rate and/or task completion time. According to Boyandin et al., animations were more appealing to users.

However, if some of the studies yield congruent conclusions for some conditions, but are contradictory for other conditions, each study is limited and in fact lists several exceptions for particular conditions. There are simply too many parameters that vary between studies. Data varied in terms of node count, and time steps, but also in edge attributes. For example Boyandin et al. (2012) included edge weight, while the others had no additional attributes encoded. Furthermore, data varies in density and amount of change, across studies. Tasks are generally hard to compare, especially since there is no common repository or taxonomy for tasks about dynamic networks. Each study used a proper implementation of the compared techniques: Boyandin et al. (2012) used

[4] "*Without control,*[the participants] *had to find their target very rapidly: 'It's hard to check all the flying balls!' However, participants in all groups did find flying balls to be exciting.*" (Robertson et al., 2008, p.7)

difference highlighting across all conditions, interactivity in the animation condition also varied. Several authors ask for improvement of animations (Farrugia & Quigley, 2011) and the integration of different techniques to balance individual drawbacks (Zaman et al., 2011; Boyandin et al., 2012). Finally, layouts were also different across studies, and as shown in the previous section, remain an unknown parameter to task performance.

In general, studies seem to show that for small networks (up to around 30 nodes) and with a few time steps (up to 4 time steps), small multiples seem to perform better overall. However, most authors suggest that animations are much better, although not perfect, for larger networks and longer time periods.

More studies are needed, investigating, for example, optimal animation speed, the effect of interaction in animated sequences and temporal multiples, data density and large networks, amount of change, etc. Furthermore, there are no studies comparing visualizations from the following categories discussed in Section 2.6: *nestings*, *temporal aggregation*, *those involving adjacency matrices*, and the different types of *timelines* (see Section 2.6). Yet, for the mentioned reasons (differences in data, implementation, task, etc.), it is unlikely that qualitative evaluations of individual techniques yield further insights, other than for particular cases. Moreover, many techniques may not be comparable at all, because they simply do not show the same information, but are rather complementary to one another.

2.9. Summary

Visualizing dynamic networks combines many different aspects, such as the notion and structure of time, structural changes inside the graph or on its attributes, visualization techniques that show topology, attributes and time, finding the right visual encodings that allow users to understand information in an unambiguous way, and provide appropriate interaction and navigation techniques. While much has been learned and designed in recent years, many open problems remain, such as:

- How to visualize large and dense dynamic graphs with thousands of nodes and hundreds of time steps?
- How to visualize changing edge weights?
- How to navigate efficiently between hundreds of time steps?
- How to track changes on higher-level graph structures (clusters, paths, etc.) over time?
- How to achieve a tradeoff for the mental map problem?
- Can performance of animated sequences be improved by interaction and better visual feedback?
- How can the drawbacks of the different techniques be balanced? and finally:
- Which tasks do users perform while exploring dynamic networks?

A potential design space for dynamic network visualizations is huge but less structured so far. A clear terminology to describe and discuss visualizations is missing, too. Individual visualizations differ in their implementation (interactive vs. non-interactive, color-coded vs. monochrome, attribute encoding vs. simple topology) and are hard to compare. Studies have evaluated mostly temporal multiples against animated sequences, but as data sets, tasks and other experiment conditions differed, the narrow evidence that the studies yield is hardly generalizable. While many visualizations rely on a single presentation (cf. Table 2.2), study authors suggest integration and improvement of techniques. In fact, a more sound exploration process would be too much constrained by a single technique.

This thesis adresses most of the above mentioned problems in an integrative manner. That is, ultimate solutions to an individual problem are hard to achieve and many solutions to these problems depend on the context they are employed in. For example, visualizing dense dynamic graphs require multiple views and interaction techniques, as well as efficient temporal navigation. On the other side, the individual advantages and drawbacks of certain visualization techniques heavily depend on the data characteristics.

Yet, in order to better understand *for what* we visualize dynamic networks, i.e. what these visualizations should support, the next chapter starts by laying out a taxonomy of tasks that users perform while exploring dynamic networks.

System/Name	Citation	Temporal Multiples	Animated Sequences	Aggregation	Nesting	Ego Networks	Timeline	Space-Time Cube
GraphAEL	Erten et al., 2003	x	x					x
Ploceus	Liu et al., 2011	x						
MatrixFlow	Perer & Sun, 2012	x						
–	Boyandin et al., 2012	x						
Maray	Friedrich & Eades, 2001		x					
Visone	Brandes & Wagner, 2003		x					
–	J.-W. Ahn et al., 2011		x					
SoNIA	Moody et al., 2005; Bender-deMoll & McFarland, 2005; McFarland & Bender-deMoll, n.d.		x					
DiffAni	Rufiange & McGuffin, 2013		x					
Information Visualization Spreadsheets	Chi et al., 1998			x				
–	Brandes et al., 2004			x			x	
–	Erten et al., 2004			x				
–	Andrews et al., 2009			x				
HoneyComb	van Ham et al., 2009			x				
Netvisia	Gove et al., 2011			x			x	
–	Hascoët & Dragicevic, 2012			x				
–	Hadlak et al., 2013				x			
TimeMatrix	Yi et al., 2010				x			
Gestalt Lines	Brandes & Nick, 2011				x			
–	Stein et al., 2010				x			
–	Farrugia et al., 2011					x		
1.5D	Shi et al., 2011					x	x	
–	Falkowski et al., 2006						x	
–	Sallaberry et al., 2013						x	
–	Reda et al., 2011						x	
Massive Parallel Sequence Views	van den Elzen et al., 2013						x	
GraphFlow	Cui et al., 2014						x	
Paralel Edge Splatting	Burch et al., 2011						x	
Semantic Substrates	Shneiderman & Aris, 2006						x	
GraphDice	Bezerianos, Chevalier, et al., 2010						x	
–	Diehl et al., 2001							x
–	Dwyer & Eades, 2002							x
–	Brandes & Corman, 2003						x	x
2.5D	Brandes et al., 2004			x			x	x
–	Gaertler & Wagner, 2006							x
–	Federico et al., 2011	x		x				x

Table 2.2.: *Representative list of visualizations of dynamic networks and the techniques they use.*

3. Towards a Task Taxonomy for Dynamic Networks

Contents

TASKS are important in creating and researching novel visualizations, because they describe which problems a visualization should help to solve. When designing a visualization, a designer should have a specific task in mind (Bertin, 1973); tasks are crucial for validation at various stages, as well as for a final evaluation (cf. Munzner, 2009 as well as Lam et al., 2012). For example Lam et al. (2012) describe the design process for visualizations in five stages; 1) *Pre-design*, 2) *design*, 3) *prototyping*, 4) *deployment*, and 5) *redesign*. Mistakes in each of these stages can impact the subsequent stages.

A list of tasks can be obtained by various means. For example, conducting interviews, using questionnaires, or observing practitioners' working process (Lam et al., 2012; Carpendale, 2008). These methods are time intensive and require practitioners' availability. Moreover, an unstructured list of tasks should then be structured in order to allow proper conclusions for design and evaluation. Structured *task taxonomies* are useful because they provide:

- task categories and grouping principles,
- the ability to evaluate existing visualizations, in checking which tasks they allow users to perform,
- a base for the definition of design guidelines and implications for interfaces and visualizations, as well as
- representative tasks for an evaluation.

Task taxonomies have been created for visualization tasks in general (e.g., Shneiderman, 1996 as well as Amar et al., 2005) and (static) networks (Lee et al., 2006). Recently J. Ahn et al. (2012) published a Technical Report describing an extensive taxonomy of tasks for dynamic networks. However, a simple description of the questions and tasks is missing. The questions we address in this chapter are:

- ▷ *How to describe and categorize tasks for dynamic networks?*
- ▷ *Which implications can we draw from the tasks for the visualization design?*

This chapter adapts and extends previous taxonomies on general tasks in information visualization (Amar et al., 2005), time and geography (Peuquet, 1994; MacEachren, 1995; N. Andrienko & Andrienko, 2004), as well as static networks (Amar et al., 2005). For the exploration of dynamic networks, we define three categories of tasks, corresponding to the three dimensions *Network elements* (Where?), *Time* (When?), and *Behaviors* (What?), then describe compound and eventually higher level tasks. While our taxonomy allows us to discuss the complexity and difficulty of tasks, tasks can be easy or hard, depending on the *visualization*. We conclude this chapter with a discussion of the relation between the tasks in our taxonomy and visual tasks, as well as list how our taxonomy can support design and evaluation of visualizations.

Our taxonomy informs the designs and evaluation of the visualizations presented in this dissertation.

3.1. Tasks for Information Visualization

This section gives an overview of the notion and purpose of tasks in information visualization, and lists the taxonomies that are important to understand the construction of our own.

The literature describes many different evaluation methods, ranging from very informal qualitative interviews, to long-term studies, and quantitative controlled user studies (cf. Plaisant, 2004; C. North, 2006; Carpendale, 2008 as well as Lam et al., 2012). Quantitative approaches, such as used in Human Computer Interaction (HCI) to assess task completion time and error rate, can assess a visualization's capability to support specific tasks, or to compare visualizations (Plaisant, 2004). However, quantification hardly captures a deeper understanding of the information and reporting of actual *insight*. Reporting on the users' insight is important to evaluate the understanding of complex and actual relevant information (C. North, 2006). Methods include scenarios (Lam et al., 2012), working with domain experts, posing open ended tasks, letting users explore the data using the visualizations, and preferably running long term observations.

	Low-level Tasks	**Higher-level Tasks**
Description	Analytic primitives (Amar et al., 2005)	Uncaptured by low and compound tasks (Amar et al., 2005; Lee et al., 2006)
Scope	Restricted set of data elements and/or attributes	Multiple elements or attributes
Answers	Precise (numbers, yes/no)	Complex answers, discussions
Domain knowledge	not required	required
Methodology	Simple or given	No methodology, user dependent
Time	Fast	Can take days and months (Carpendale, 2008)

Table 3.1.: *Contrasting low and higher-level tasks.*

3.1.1. High and Low-level Tasks

Understanding of task complexity helps in assessing which tasks to pose in order to get the right level of feedback in an evaluation; simpler and precise tasks allow for better quantification, complex tasks allow ecological validity. In quantitative evaluation, common measures for effectiveness are the time it takes a participant to perform a given task (*task completion time*), and the number of errors he produces (*error rate*). While task completion time is simple to assess and compare, time is measured equally, errors are harder to interpret since they depend on the tasks and data; some tasks may have only two possible answers, others require the user to choose from a scale or give free form answers. In order to make errors comparable across tasks, errors must be quantifiable.

The literature distinguishes between *low-level* tasks and *high-level* tasks (Amar et al., 2005; Lee et al., 2006; J. Ahn et al., 2012). Low-level tasks represent simple questions, described as *"analytic primitive"* (Amar et al., 2005). For example, the task *"Find Adjacent Nodes [...]"* from Lee et al. (2006) is considered low-level. Examples for higher-level tasks are *"Are there car aspects that Toyota has concentrated on?"* (Amar et al., 2005) or about the comparison of two networks or changes in dynamic networks (Lee et al., 2006).

Higher-level tasks require specific domain knowledge about the data, the domain. and involve human judgement. Human judgement is perhaps the most prominent aspect for differentiating between low-level and higher-level tasks; while low-level tasks describe a clear methodology, higher-level tasks may not follow any particular methodology. Table 3.1 summarizes the notion of low and higher-level tasks. However, the transition between both categories is not strict; tasks may be more *low-level* or more *high-level*.

Amar et al.'s taxonomy lists ten low-level analytical tasks:

1. **Retrieve Value:** Find the value for an attribute, associated with an element (e.g. *How long is the movie Gone with the Wind?*[1])
2. **Filter:** Find the elements whose attribute values satisfy certain filter criteria (e.g. *What comedies have won awards?*)
3. **Compute Derived Value:** Calculate derived measures from a set of values (e.g. *How many manufacturers of cars are there?*)
4. **Find Extremum:** Find elements with extreme attribute values (e.g. *What is the car with the highest MPG?*)
5. **Sort:** Rank data elements according to some metric (e.g. *Order the cars by weight.*)
6. **Determine Range:** Find the range of an attribute within a given set of data elements (e.g. *What is the range of film lengths?*)
7. **Characterize Distribution:** Describe the characteristics of a distribution of attribute values (e.g. *What is the age distribution of shoppers?*)
8. **Find Anomalies:** Find outliers in a data set which do not correspond to the general trend in the data (e.g. *Are there exceptions to the relationship between horsepower and acceleration?*)
9. **Cluster:** Group elements into groups according to attribute value similarity (e.g. *Are there groups of cereals w/ similar fat/calories/sugar?*)
10. **Correlate:** Define relations between the values of two attributes, across a set of data elements (e.g. *Is there a correlation between carbohydrates and fat?*)

Between low-level and higher-level tasks, users often perform *compound tasks*. Compound tasks can be divided into a series of primitives from the above list. For example, first retrieve particular *values* and then *sort*, or *compure derived values*.

The next section describes low and compound tasks for (static) networks.

3.1.2. Tasks for Networks by Lee et al., 2006

Lee et al. (2006) classify tasks on static networks, but do not explicitly include tasks for the exploration of dynamic networks. Exploration of dynamic networks is listed as the single higher-level task *"How has the graph changed over time?"*. This task is not a low-level task, since it involves multiple data elements, requires knowledge about changing networks, does not imply a particular methodology, and the questions are rather open (cf. Table 3.1).

In their taxonomy, Lee et al. define *graph specific elements: nodes*, *links*, *paths*, *graphs*, *connected components*, *clusters*, and *groups* (set of nodes related by attribute).

[1] Example tasks in this list are taken from Amar et al. (2005).

Lee et al. then define four groups of tasks:

1. **Topology based tasks** are performed on the network structure itself and involve for example, *"Find the set of nodes adjacent to a node?"*[2] or *"Given nodes, find a set of nodes that are connected to all of them."*
2. **Attribute based tasks** are about attributes associated with graph elements, such as *"Find the nodes having a specific attribute value."* or *"Which node is connected by a link having the largest/smallest value?"*
3. **Browsing tasks** are tasks related to following links and paths in the network: *"After they follow a path in the above task, they may want to see A's other friends"*
4. **Overview tasks** are related to the network in general (e.g. size) and the contained clusters and connected components: *"estimate the size of a network"*, *"find particular patterns and outliers"*

From J. Ahn et al.'s taxonomy we adapt the graph specific elements. However, the taxonomy does not provide the means to describe change over time explicitly.

3.1.3. Temporal Tasks by MacEachren

MacEachren (1995) describes seven tasks related to time, and the existence of geographic objects. We will reuse these tasks later in our own taxonomy:

- **Existence of Entity:** *Does an entity exist at a given time?* The answer to such a question is usually *yes* or *no* or may be *uncertain*.
- **Temporal location**: *When does an entity exist?* The answer to such questions is a particular instant (time point or interval).
- **Time interval**: *How long is the time span from beginning to end of an entity?* The answer to these types of question is a set of instants or a period.
- **Temporal Texture**: *How often does an entity occur?* The answer is a number or fraction that defines a frequency (e.g. five times per hour).
- **Rate of change**: *How fast is an entity changing or how much difference is there?* Answers include absolute or relative values about change, as well as less quantifiable values such as *very fast*, *faster than*, etc.
- **Sequence**: *In what order do entities appear?* The answer is an ordering which can include concurrency.
- **Synchronization**: *Do entities occur together?* Answers can include if two entities co-occur, and how much (Allen, 1984).

[2] Example tasks in this list are taken from Lee et al. (2006)

3.1.4. Task framework by Andrienko and Andrienko

N. Andrienko and Andrienko (2004) describe an extensive and detailed framework of tasks for the analysis of spatial and temporal analysis. In their framework, a task is defined by two parts, *constraints*, which define a set of known elements, and a *target*, which specifies the type of information that is wanted. Similar to Lee et al. and Bertin (1973), the framework differentiates into *elementary* tasks, which involve individual elements and their attributes, and *synoptic* (compound) tasks, which involve sets of objects. In elementary tasks, the target can be an attribute value (*characterisitc*), a location or a temporal value (*reference*). Targets on the synoptic level involve trends, distributions and variations (*behaviors*).

The framework of N. Andrienko and Andrienko (2006) further introduces a formal notion to describe all the possible relationships between constraints and target, listing many detailed tasks. An adaptation to dynamic networks would yield a taxonomy that we consider too complex and too hard to use, for our purposes. Nevertheless, the approach we adapt is similar but less formal, as detailed in the following.

3.1.5. Tasks for Spatio Temporal Data by Peuquet

Peuquet (1994) proposes a much simpler—although less expressive—task framework for spatio-temporal analysis. Her taxonomy describes tasks in terms of queries, according to three *representations*, each one tailored to representing a particular aspect of the data:

- ***Where*:** Describes spatial locations (woodlands, rivers, cities, etc.) and relations between them (administrative hierarchies, neighboring regions, etc.).
- ***What*:** Describes potentially movable objects (animals, storms, etc.) and their attributes, embedded in a knowledge representation, such as a taxonomy and other data models.
- ***When*:** Describes time points, sets of times and changes that happen to elements' positions or attributes.

Based on these representations, a system can receive queries and return values. Queries are created by defining known elements in *two* representations, and asking for the unknown one in the *remaining* representation:

- **What + When = Where:** Describe the location where a specific element is present at a given time.
- **When + Where = What:** Describe an element that is present at a certain location at a given time.
- **Where + What = When:** Describe a time when a certain element is present at a given location.

If an analyst has to find the time point when a certain element is present at a given location (*Where* + *What* = *When*), we can say that the information he is provided with, is the location (*Where*) and the element (*What*), while he has to search in time (*When*). For questions related to the What dimension (*When* + *Where* = *What*), the analyst must be provided with an appropriate view on the particular time and region, and has to describe the different elements (*What*).

In differentiating between *known* and *unknown* information, Peuquet's framework is similar to those by N. Andrienko and Andrienko and Bertin. Yet, we consider Peuquet's framework much simpler, and make it the basis for our taxonomy of dynamic network tasks. Yet, a direct application is not completely straightforward.

3.2. Task Dimensions for Dynamic Networks

In the following, we explain how we adapt Peuquet's approach to dynamic networks, taking into account the other taxonomies mentioned earlier; MacEachren (1995); Amar et al. (2005) and Lee et al. (2006). We first describe the units in each of the representations (we will call representations *dimensions*) and then systematically describe *queries*. A *task* is defined as consisting of two parts: a *query* (made out of known and unknown dimensions) and an *action* (i.e. *query* + *action* = *task*). While the query retrieves data, the *action* describes additional actions performed on the values, corresponding to Amar et al.'s tasks: retrieve, sort, characterize distribution, derive value, etc. For example, the task

How often is node n present in the network?

consists of a query that retrieves the instants the node is present, and an action that counts the number of instants. Yet, in order to properly describe queries for dynamic networks, we must re-define two of Peuquet's initial representations.

Most importantly, networks do not have fixed spatial (screen) positions; rather, nodes move in the abstract graphic space. Node positions are defined by the graph layout, taking into account their connectivity. Nodes and links in the networks are similar to Peuquet's moving elements in the *What* representation. However, referring to moving elements in space implies aspects such as: a) *proximity* and *distance*, b) *local* vs. *global*, c) *locations, paths* and *areas*, as well as d) *characteristics of spatial units*. Transposed to a network, these concepts are in fact represented by the network's topology. **Asking *Where* in networks, we ask about the graph elements and their attributes**; nodes, links, clusters, etc; Peuquet's question about *Where in space*, becomes *Where in the network* (topologically speaking).

Changes that happen to elements over time are captured by Peuquet's *When* dimension. In our taxonomy, **asking *What*, we capture questions that ask for the behavior of elements and their attributes.**

Our three dimensions WHERE, WHAT and WHEN, defined in the following, result from a direct extension of static networks (WHERE) into time (WHEN), while accounting for the behavior of elements in static networks (WHAT). Our notion of behavior is similar to that used by N. Andrienko and Andrienko (2006).

3.2.1. WHERE: Dimension of Network Elements

Our WHERE dimension describes the elements in a static network: *graph elements* and *attributes*. Graph elements are adapted from Lee et al. (2006)'s taxonomy and include the following graph *primitives*: *nodes*, *edges*, and an *attribute* associated with a particular node or edge, e.g., the weight of an edge *e* or the type of a node *n*.

To be able to describe higher-level changes in the network, we define *Compound graph elements*, which are made of multiple graph primitives:

- *dyads*: a pair of two nodes,
- *triads*: a set of three nodes,
- *paths*: a set connections and nodes between two arbitrary nodes,
- *motifs*: a specific topological configuration such as a cycle, star etc.,
- *clusters*: group of densely connected nodes,
- *cliques*: group of completely connected nodes,
- *groups*: any set of nodes, not necessarily defined by topology or connectedness,
- *subgraphs*: any subset of nodes and edges, etc.

The list of graph elements and attributes can be extensive, and depends on the domain that the network represents.

In our WHERE dimension, we include (domain) attributes of graph elements (e.g., edge weight, a node's group type) as well as measures about graph elements (e.g., degree, centrality, age). Attributes and graph measures are part of a static graph and are/can be defined *in every time point*. If a node or attribute values is not present in the network, then it is defined just as "not present".

3.2.2. WHEN: Dimension of Time

Our WHEN dimension captures temporal units and derived values. Units includes one primitive, a single *instant*. Compound units consist of (ordered) sets of instants:

- *pair of adjacent instants*,
- *pair of non-adjacent instants*,
- *set of adjacent instants* (*period*).
- *set of non-adjacent instants*,
- *set of equidistant instants* (recurrent or cyclic instants), etc.

Graph Element	Individual Events	Long-term changes
Node	appear / disappear connect to / disconnect from enter, leave, change, or connect to a cluster or group	get isolated, get integrated
Edge	appear / disappear connect two nodes	strengthen, weaken
Attribute value	increase, decrease, remain equal	increase, decrease, remain equal, pulse, converge, diverge, etc.
Cluster, Group, Connected Component	appear, disappear, dissolve get denser, sparser merge with other cluster, split grow, shrink exchange nodes attract, repel nodes	
Path	grow / shrink[3] break	
Graph	grow, shrink, become sparse, dense	

Table 3.2.: *Representative list of possible changes to primitive and compound graph elements.*

Using Amar et al.'s tasks (indicated in brackets), further values can be inferred from these temporal units. These inferred values correspond to questions described by other authors: Allen (1984); MacEachren (1995) and Moody et al. (2005):

- *how often* (derive value),
- *how long* (range between two instants),
- *in what order* (sort),
- *before or after* (sort),
- *during* (correlate),
- *beginning*,
- *ending*, etc.

3.2.3. WHAT: Dimension of Behavior

Our WHAT dimension captures short- as well as long-term events that happen to elements from the WHERE dimension (graph elements and their attributes) in the respective units of the WHEN dimension. WHEN and WHERE are orthogonal dimensions; WHAT captures WHERE $\times$ WHEN.

Low-level changes happen to primitives in a single or two adjacent instants, such as *appearance* and *disappearance* of a node or changes to one of its attributes (*increase* attribute a by value x). Low-level changes are responsible for *higher-level changes*, which:

- happen over a longer period of time,
- involve multiple graph elements, and/or
- events that happen on compound graph elements.

Table 3.2 lists low and higher-level changes that occur to graph elements over time, in a sense extending Table 2.1 in Section 2.3.3.

Some topological measures (e.g. group size) are implicitly mentioned in the table as *growing* and *shrinking* of a group. Changes to other measures (e.g. centrality) and domain attributes (e.g. edge weight, node type, etc.) are summarized as *attributes*. Attribute values are time-series with the general change *increase, decrease, pulse, converge, diverge*, etc. Changes further have characteristics such as *speed*, or *distribution* (among elements in the WHERE dimension).

3.3. 1-Dimensional Tasks

We now have all the definitions to describe tasks for dynamic networks: (i) a methodology to create queries, (ii) definitions of units in each of the three dimensions WHERE, WHEN, WHAT, and (iii) analytic actions from Amar et al.'s taxonomy.

In the following, we create *1-dimensional* tasks. These tasks involve two known and *one* unknown dimension which contains the answer and which is to be explored. We then refer to compound and higher-level tasks. Tasks in this section are created by systematically combining units from the three dimensions described in the previous section and have been complemented with tasks found in the literature about dynamic networks.

3.3.1. Where? — Network Tasks

The following tasks are of the form WHEN + WHAT = WHERE, which translates into *"Which element in the network is affected by a given event at a given time?"*

- *Are nodes n_1 and n_2 connected at time t?*
- *How many nodes disappear in time t?*
- *Which group is grows fastest?*
- *Which nodes keep the exact same neighbors between times t_1 and t_2?*
- *Which two nodes are connected only once?*
- *Which cluster is the most unstable over time?*
- *Which nodes remain in group g_1 from time t_1 to t_2.*
- *Which parts of the graph remain just for one time step?*
- *Which node increases its degree over time (Archambault et al., 2011a)*
- *Which set of three nodes appear together exactly once? (Archambault et al., 2011a)*
- *Which two edges appear together exactly once? (Archambault et al., 2011a)*

- *Which student was called the most (overall) during the 6 week term (Purchase & Samra, 2008)*
- *Which of these nodes has the most edges: [yellow, green, red, purple] (Saffrey & Purchase, 2008)*
- *Which of these groups of nodes has the most edges: [yellow, green, red, purple] (Saffrey & Purchase, 2008)*
- *Find a node that has only one connection on weeks 3, 4 and 5. (Farrugia & Quigley, 2011)*

3.3.2. When? — Temporal Tasks

Tasks of the WHEN dimension are of the form WHERE + WHAT = WHEN, which translates into *"When does a particular event happen to an element?"*.

- *When does node n disappear?*
- *When are nodes n_1 and n_2 connected?*
- *How long does it take until clusters c_1 and c_2 are completely merged?*
- *In what order do nodes n_1 until n_{10} appear?*
- *In which time slice is the shortest path between these two nodes: [yellow,green]? (Saffrey & Purchase, 2008)*
- *In which time slice is the shortest path between these two groups of nodes: [yellow,green]? (Saffrey & Purchase, 2008)*
- *For how many weeks does the connection between Node 1 and Node 6 last? (Farrugia & Quigley, 2011)*

3.3.3. What? — Behaviour Tasks

Tasks of the WHAT dimension are of the form WHERE + WHEN = WHAT, which translates into *"What happens to a particular element at a given time?"*.

- *How does the edge weight of edge e change over time?*
- *How does the degree of node n evolve over time?*
- *What happens to cluster c between times t_1 and t_2?*
- *Are nodes n_1 and n_2 connected at time t?*
- *Which two edges appear together exactly once? (Archambault et al., 2011a)*
- *How does the number of connections of Node 1 change in weeks 2, 3 and 4? (Farrugia & Quigley, 2011)*

Each time we pose a query, we restrict some dimensions, while returning a set of values that are used as input into our action. Recalling the task:

How often is node n present in the network?,

which is made of the query *When is node n present* (WHERE+WHAT=WHEN), the query returns a set *of arbitrary instants* which serve as input to Amar et al.'s task: *derive value* (counting time steps).

3.4. Compound Tasks

The above-mentioned 1-dimensional tasks can be used to describe compound tasks. *Compound tasks* consist of several low-level tasks to be performed in *sequence*, i.e. the answer to the first task, serves as input to the second one.

Examples of compound tasks are:

- *Which of the groups that node n is part of has the longest lifespan?*
- *In what order do the nodes in group g_1 join it?*
- *How big is group g when node n leaves it?*

The last task combines the tasks "When does n_1 leaves g?" and "Is n_2 part of g at time t?", whereby t is the answer to the first task.

3.5. Higher-level Tasks

Our framework allows us—to a certain extent—to discuss the complexity of a task. There may be (at least) two aspects that make one task more complex than another:

- number of sub-tasks in compound tasks, and
- the cognitive effort to perform the action.

Many tasks related to dynamic network exploration are hard to describe with low-level tasks. Here, we list tasks that we consider higher-level. *Higher-level* tasks still involve our three dimensions and Amar et al.'s tasks but in some cases, only *one* dimension is given, i.e. a single node, two time steps, etc.

Examples of higher-level tasks include:

- *Find similar time steps.*
- *Describe the distribution of edge density in time and topology.*
- *What are trends in the network?*
- *Which are the important time points?*
- *Which nodes behave ab-normally?*
- *Describe the formation of clusters.*
- *Describe the major changes between t_1 and t_2.*

3.6. From Data Tasks to Visual Tasks

So far, the described tasks are all in *data space*, i.e. they refer to objects in the data (the dynamic network). Those tasks are described independently from the actual visualization. A task can be low-level in data space, e.g. *Is node n present at time t?* but can be quite hard with certain visualizations. For example, consider an aggregated representation of a dynamic network (Section 2.6): assessing when a node is present and when not is *impossible*, if no additional visual clue is provided (e.g. color-coded time, labels, interaction). Visualizations are designed to *support* particular tasks, and can render a task that is difficult in data space easy in visual space. Answering whether the node n is present in t can be simple on a time line visualization (Section 2.6) if nodes are properly laid out and labeled.

Turning data-tasks into visual tasks requires users to map between both. This translation from network-oriented tasks into perception tasks and interactions is not straightforward: it requires some degree of familiarity and experience with the visual representation. For example, node-link diagrams represent topology, using points for nodes, lines for edges, and rely on a layout algorithm to place the nodes on the plane. Graph attributes are visualized using the visual attributes associated with the points and lines. A translation of network tasks into perception tasks may require some learning, leading to some acquired knowledge on the properties of the visual representation.

In particular, users familiar with network visualizations know that, in addition to reading lines to understand the topology of a graph, connected nodes in a force-based layout are closer-by than disconnected ones, with the exception of "bridge" nodes connecting two distant groups and that can be detected with their long connecting lines. This is a property enforced by most layout algorithms, although it is sometimes relaxed for small graphs where lines are easy to read, allowing nodes to be evenly spaced. Otherwise, this property is essential for larger graphs where lines should be drawn lightly to limit occlusion and cannot be read easily. Therefore, topology tasks can be performed by either reading the lines, if possible, or looking at the proximity of nodes. For larger graphs, densely connected nodes become clouds where bridges can be noticed with their long outstanding lines.

Most of the studies on graph readability have focused on relatively small graphs (¡ 50 nodes) where participants had to read individual lines or identify nodes to perform a tasks. Yet higher-level network-tasks, which involve tracking groups of nodes and estimate the type and quantity of change, require both proximity inspection and reading lines when necessary and possible.

3.7. Implications

We believe our taxonomy is useful, because it allows us to describe components of user interfaces and systems in terms of which dimensions and combinations thereof they cover; and thus, what low-level and compound tasks they support. The following high-level implications for the design of dynamic network visualization systems can be drawn from it.

When: Specific time steps must be easy to identify and reach, so that users can compare and analyze them in detail. Features that can meet these requirements include overviews of the network's time steps, mechanisms for the quick selection and filtering of those steps, mechanisms for temporal aggregation, and a flexible scheme to navigate from one step to another.

Where: Graph elements with particular properties must be easy to identify in the network's topology, and to track over time steps. The layout and rendering of the network should be parameterized carefully.

What: Understanding the nature of, and possibly quantifying, the changes that graph elements undergo requires that the corresponding attributes be easy to identify. Those should be emphasized.

Compound and Higher-level Tasks: Performing tasks in sequence can require switching to a more appropriate representation of the network while maintaining the users mental model of the data. It further implies interaction and navigation mechanisms.

Task Complexity: Although our taxonomy is yet in an initial state, we can sketch and discuss complexity of tasks in data-space. Important factors on task complexity may be the number of open dimensions, the number of given elements in each of the given dimensions, and the number of sub-tasks in a compound task. Eventually, complexity of a task also depends on the data itself, i.e. the number of elements that have to be searched; retrieving similar time steps in a graph with five time steps is much easier than in one with over a hundred.

Visual Tasks: Tasks that seem complex in data-space can be simple in visual-space. Especially those tasks involving multiple units from our dimensions, e.g. groups, time periods, etc., can be supported by pre-attentive mappings and well designed visualizations. For example, showing the differences between two time steps explicitly makes comparison tasks easier.

Evaluation: So far, the focus of the implications was on the design of visualizations, but what can we learn from the taxonomy about evaluating visualizations? Low-level tasks

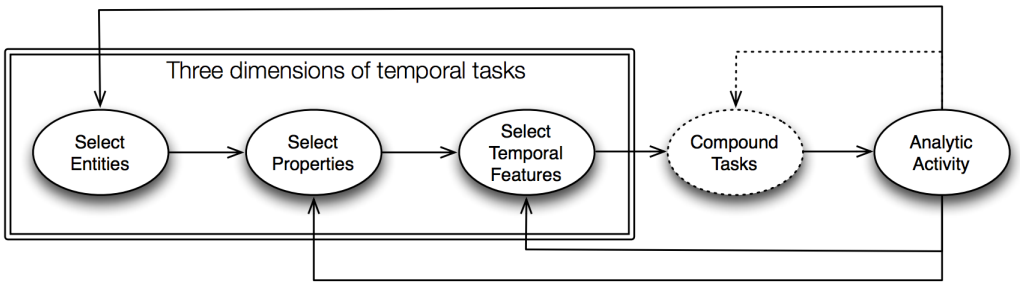


Figure 3.1.: *Pipeline model in the task taxonomy for dynamic networks by J. Ahn et al. (2012).*

and such that involve few elements are better suited to be used in controlled user studies, since quantitive measurements are more likely to be comparable. Higher-level tasks, which involve human judgement, may yield very different results across participants. However, carefully choosing tasks and components of queries may allow for evaluation of even more difficult tasks.

3.8. Comparison with the Taxonomy by Ahn et al. (2012)

While in the progress of publishing our taxonomy, J. Ahn et al. (2012) made a Technical Report publicly available, which describes a taxonomy of tasks for the analysis of dynamic networks. This section briefly compares their approach to ours.

Their taxonomy is based on three dimensions:

1. Network *Entities*, which fall into nodes, links, subgroups, or the entire network.
2. Network *Properties* such as attributes associated to nodes and edges, and which capture both *topological attributes* (e.g. node degree) and *domain attributes*.
3. *Temporal Features*, referring to the change in property values and captures:
    - *Temporal Features of individual events* such as *birth* or *death* (appearance, disappearance of network entities),
    - *Temporal Features of aggregated events—Shape of Change* spanning multiple time steps and describe growth, convergence, and stability,
    - *Temporal Features of aggregated events—Rate of Change* also spanning multiple time steps and describ speed and acceleration of change.

A task in their taxonomy is defined as a triple of the form *Entity-Property-Temporal Feature*. For reasons of complexity, the corresponding design space described in the report captures only two dimensions; Entities and Temporal Features. Tasks are described with a *pipeline model*, illustrated in Figure 3.1. The pipeline consists of the three steps: 1) *Select Entities*, 2) *select Properties*, and 3) *select Temporal Features* necessary for analysis. During analysis, users and analysts change tasks by adapting the stages of the

pipeline, i.e. change the graph elements of interest, change temporal feature to observe. Compound tasks are described as involving multiple tasks.

The basic building blocks—graph entities, attributes, temporal features—are similar in J. Ahn et al.'s taxonomy and in ours, although not grouped the same way. Their taxonomy is very detailed and lists many important aspects of temporal changes in dynamic networks. However, similar to N. Andrienko and Andrienko's taxonomy, we consider J. Ahn et al.'s taxonomy very complex to inform the design of visualizations, because of the many different concepts involved. With our taxonomy, we dso not aim for completeness in terms of tasks. Instead, we are seeking for a simple way to describe the main dimensions that an interface must allow to explore, and which complications arise from those.

3.9. Conclusion and Limitations

As the number of possible tasks is virtually infinite (Lee et al., 2006), we do not claim that our taxonomy is complete. Constructing task taxonomies is a complex process and ideally involves analyzing examples from the literature and talking to experts. Our taxonomy is based on the structural analysis of data. It presents an initial step towards a more comprehensive understanding for dynamic network exploration.

With our taxonomy we are able to:

- "Generate" and categorize various tasks for user evaluation,
- Inform the design of novel interfaces,
- Discuss the complexity of tasks in data-space,
- Categorize existing interfaces in how far they support the three dimensions. For example, an interface that does not allow for selecting an individual time point lacks support for all (low and high-level) tasks that require a single time step; either as given in the task (e.g. WHEN + WHERE = WHAT) or as answer (WHAT + WHERE = WHEN).

The taxonomy informs the design and evaluations of the visualizations developed in this dissertation. Yet, it is extensible in many ways. Future work consists of relating it to tasks in the literature, but more importantly to try to better describe types of changes and events that are important in dynamic network analysis (WHAT). A more comprehensive taxonomy does not necessarily capture more tasks or allow us to better judge the complexity of tasks, but to provide good rationales of when a particular visualization or technique is necessary, what they have to show and which interaction is necessary.

4. Animated Transitions and Temporal Navigation

Contents

VISUALIZATION systems for dynamic networks should provide effective visual representations and interactions to support tasks such as described in our taxonomy. Some tasks require exploration of events over time (WHEN), within topology (WHERE) as well as the type of changes (WHAT). A user's focus has to switch during various stages of the analysis or exploration process; a compound task may first require the user to find a particular time point, then describe the network topology, and eventually compare it to a particular other time point.

Many techniques have been described and evaluated in the literature (see Chapter 2). Sequences of the network, one time step at a time, provide particular advantages. Showing any time step individually reduces complexity of the overall network in an intuitive way and allows for detailed observation of the topology for every time step (WHERE). As changes are implicitly encoded as difference between time steps, the network visualization itself remains free to encode attributes about the topology (e.g. node degree), domain-specific attributes (e.g. node or edge type), or measures related to time (e.g. age of a node or edge). Users can easily track individual elements or subgraphs (WHEN + WHAT).

Network sequences are simple to integrate, and have already been integrated, into existing network visualization systems (e.g., Cytoscape (*Cytoscape: An Open Source Platform for Complex Network Analysis and Visualization*, 2013), Gephi (Bastian et al., 2009), Pajek (*Pajek - Program for Large Network Analysis*, 1997), Visone (Brandes & Wagner, 2003)), using animations between individual time steps to provide some cognitive support to indicate state changes (Tversky, Morrison, & Betrancourt, 2002).

Navigation between time steps is provided by common video controls and/or time sliders (e.g., TempoVis (J.-W. Ahn et al., 2011), Gephi) to navigate between time steps (*temporal navigation*).

Technically, animated sequences allow exploration of large networks—the entire screen is available for a single time step—and such with many time steps. However, their value tends to decrease as the number of elements that change between two steps increases. A further limitation comes from their inability, in their basic form, to transition between non-contiguous time steps. Especially for networks with many time steps, all animated transitions between intermediate steps have to be played. This makes the comparison of distant time steps difficult and actually increases users' cognitive load.

This chapter addresses the following research questions:

- ▷ *How can animations between time steps be improved to facilitate perception of changes?*
- ▷ *How can users navigate easily between time steps?*
- ▷ *Which complementary techniques and visualizations are necessary in a system for higher-level task support?*

This chapter presents an interactive visual interface, called *GraphDiaries*, which explores animated staged transitions with change highlighting between time steps, as well as adaptive and non-linear temporal navigation techniques. The design of GraphDiaries is motivated by the three dimensions in our taxonomy from Chapter 3 and combines various techniques in a consistent way; small multiples, difference highlighting, temporal aggregation, time sliders, interactive layout stabilization and a method to track nodes over time. It is extensible in many ways and provides a base for further research.

We evaluate our animated transitions and non-linear navigation with higher-level tasks. While studies on animations have so far focused on relatively small graphs where participants had to read individual nodes and lines to perform topology tasks, we are interested in tasks which involve tracking groups of nodes and estimate the type and quantity of change.

4.1. Animations in the User Interface

Comparing the results from the studies mentioned in Section 2.8 does not lead to a simple answer about the general usefulness of animations for dynamic networks. Although animations barely increased performance for most tasks, users often ranked them high in preference. User feedback also reveals some drawbacks of animations: distraction, longer run time, difficulties to find particular time steps, which in turn increases task completion time. These results altogether reveal that many important aspects of animations are still not well understood; pacing, staging, ordering of stages, graphical rendition of transitions (smooth/abrupt trajectory, fading, etc.). More experiments, especially ones involving

higher-level tasks and larger data sets, are required to better understand the challenges and find interaction techniques that efficiently support dynamic network exploration.

Mentioned across all studies, either implicitly or explicitly, is the lack of interactive control for animations. In many applications that display dynamic networks, animations are precomputed, and navigation in time is linear (e.g., Gephi (Bastian et al., 2009), TempoVis (J.-W. Ahn et al., 2011), Visone (Brandes & Wagner, 2003), Pohl, Reitz, and Birke (2008)).

Studies concluded that animations perform generally worse than small multiples, but the authors see more potential in using animations. This section reviews how animations are used in user interfaces and information visualization in general.

In general, animations have long been used to convey changes in user interfaces. For example, Chang and Ungar (1993) describe animations for the user interface, inspired by cartoon animations. These animations include slow-in slow-out movements, motion blur and dissolving windows, helping user to keep track of state changes in the interface. Section 2.8 already reported on studies that evaluated animations under different conditions.

4.1.1. Animations in Information Visualization

Heer and Robertson (2007) describe animated transitions between data graphics, such as line charts, pie charts and bar graphs. They report that users generally prefer slower animations. Animations helped users perceive changes much better, but the authors recommend to use *staged transitions* instead of parallel ones, even if they report that "the advantages are not overwhelming". Bezerianos, Chevalier, et al. (2010) employ animation to switch axes between between scatterplot views of graphs as first described in ScatterDice (Elmqvist, Dragicevic, & Fekete, 2008). While using animations to highlight changes within textual document histories, Chevalier, Dragicevic, Bezerianos, and Fekete (2010) avoid staged transitions to shorten transition time which, in turn, is an important factor for the efficient use of short-term memory when interpreting animations.

Staged animations have also been used for visualizing dynamic trees (Guilmaine, Viau, & McGuffin, 2012), e.g., when expanding subtrees in SpaceTree (Plaisant, Grosjean, & Bederson, 2002) or when navigating through the tree's changes over time (Card, Sun, Pendleton, Heer, & Bodnar, 2006). DOITrees (Heer & Card, 2004) also show animations when navigating the tree, but run the different types of animations (subtree expansion, layout adaption) in parallel. In addition, nodes that appear or disappear are briefly highlighted. Yee et al. (2001) employ animations to navigate in large graphs, while showing the neighborhood of only one node at a time.

4.1.2. Effectiveness of Animations

Animations as a means to convey changes in user interfaces has been extensively studied in psychology. Tversky et al. (2002) are critical about the effect of animations to communicate information, arguing that even if animations are intuitive in showing temporal changes (*Principle of Congruency*), they are hard to perceive and process by humans (*Principle of Apprehension*). The problem is to find an appropriate speed that allows the user to entirely "read" and understand the visualization; if animations are too short users can miss information, if they are too long, users have to wait. Generally, the less change happens, the faster a transition can be.

Tversky et al. contrast cases where animations have been effective and where not. Animations have been effective in such cases where (i) they conveyed *additional information* that was not shown in static variates of a graphics, for example, speed or changes on a finer level of granularity. Animations have further been considered useful in (ii) showing the trajectory of objects, for example though space, and (iii) are effective to support learning in school (in most cases). However, Tversky et al. argue that animation in the referred studies performed better because either animations showed information not visible in static graphics, or because they used better visualizations and were interactively controlled by the user. Tversky et al. conclude that interaction on its own is highly supportive for learning, since users can re-view animations, focus an particular parts and adapt the animation speed. Tversky et al. conclude that the (pure) effect of animations *per se* seems little or just not properly evaluated.

The results by Robertson et al. (2008) (see Section 2.8) support Tversky's findings. Both encourage animations for presentation (i.e. explain system in school classes), but discourage it for analysis. They considered it crucial that a presenter draws attention to (parts of) the animated data and *explains* what happens.

Summarizing the concerns about animations manifests that animations are desired where they show additional information and/or if they are interactive; although they seem to perform poorer, animations seem to be more engaging to users. This further suggests that animations are not appropriate to show changes across time steps in dynamic networks.

From the related work on the effectiveness of animations in user interfaces and from the studies in the graph literature, we conclude that animations are *supportive* in cases where there is interaction and temporal navigation. Animations *can* further show additional information about changes and events. The following sections describe our approach to an interface that enhances animated transitions by integrating them in a more general interface, complemented with other visualizations and improved interaction control.

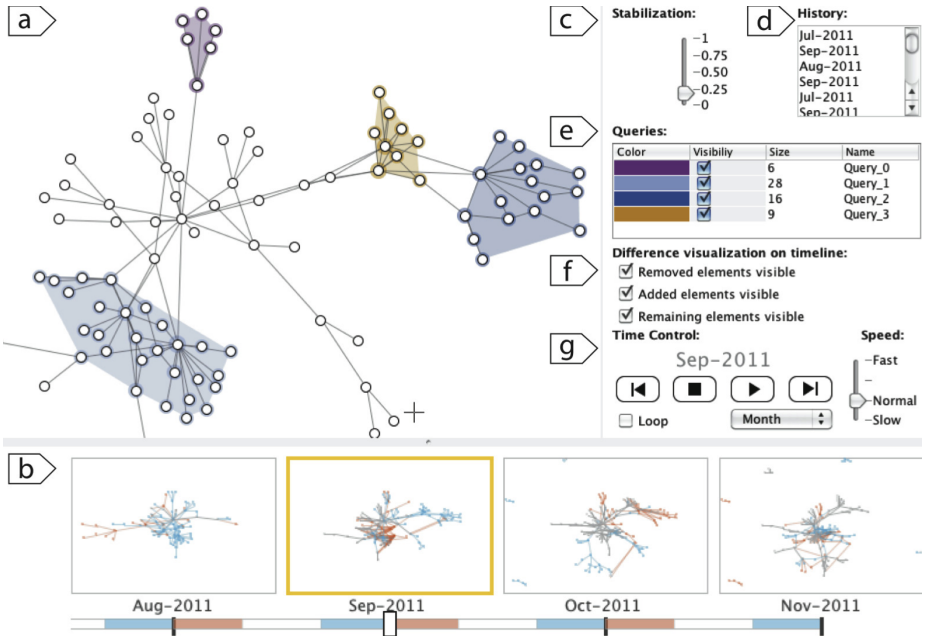


Figure 4.1.: *GraphDiaries interface: a) Network view, b) Timeline, c) Layout stabilization slider, d) Navigation history, e) Node queries, f) Panel to change visibility of red, blue or gray elements in the Timeline, g) Animation playback panel.*

4.2. Graph Diaries

GraphDiaries is an interface designed to help users answer questions related to the different dimensions of our taxonomy presented in Chapter 3 : WHERE, WHAT and WHEN. GraphDiaries employs *interactive, staged, animated transitions* that highlight changes from one time step to another, difference views and small multiples for temporal navigation. Users are able to control many of the transition parameters, such as pace, position, type and jump between arbitrary time steps. By integrating different techniques and interface components, we try to overcome some of the drawbacks of animations, as discussed in the last section.

4.2.1. Interface Components

The network view (Figure A.2(a)) shows the network as a node-link diagram at the time step currently selected in the Timeline (Figure A.2(b)). The network view focuses on answering questions about the WHERE and WHAT dimensions, i.e. where do changes happen, and what changes happen. The WHEN dimension is the primary focus of the thumbnails and the slider in the Timeline. Each thumbnail shows the network at a particular time step and is used for temporal navigation. *Difference highlighting* in the thumbnails and the network view highlight differences between any given step and the

one before, using the same visual encoding as in our staged transitions (Section 4.3): removed elements are colored red, new ones blue, remaining ones gray.

Additional interface components, shown in Figure A.2, provide further support to relate the taxonomy's three dimensions. The layout slider (c) controls layout stability (supporting element tracking and exploration within the WHERE), as further explained in Section 4.2.2. Navigation history (d) and the time control panel (g) provide higher-level playback and access to previously visited time steps (WHEN) in the network view. Conversely, options in (f) let users configure what information is shown (WHAT type of changes) in the timeline's thumbnails. Finally, the node query panel (e) lists node queries created by users, as detailed in Section 4.5.

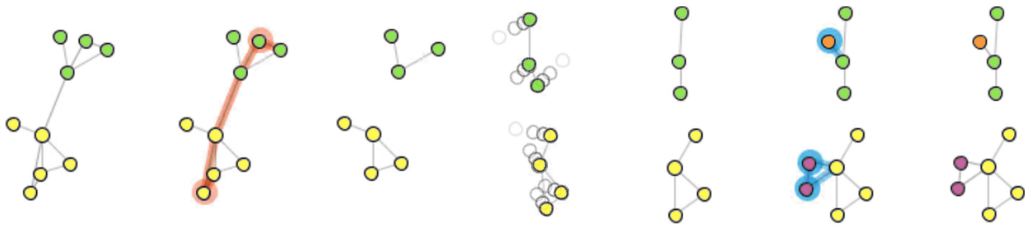
4.2.2. Dynamic Graph Layout

Layout stabilization (Figure A.2(c)) enables users to choose between a globally optimized layout for mental map preservation, a locally optimized layout for each time step, or any configuration in-between obtained by linear interpolation. Although a slider for partial stabilization has been used previously by Federico et al. (2011), our implementation is actually independent of the underlying graph layout. We were interested in how differently stable layouts combine with the different transitions, in order to support users in different tasks. Depending on the layout strategy, the same transition allow users to track different types of changes; combined with change highlighting, a stable layout draws users' attention to regions of the network that are changing—users can navigate quickly through time and to perceive the different amount of red and blue within the graph. While for a locally optimized layout, transitions support tracking changes in entities' positions, thus reflecting their new neighborhood (Section 4.3.3).

Layout Calculation

Global and locally-optimized layouts are computed as follows: the global layout is computed using the LinLog (Noack, 2007) algorithm on the whole time-aggregated graph, taking into account the number of edges between node pairs as edge weight. The locally-optimized layouts are computed for all time steps t_i, by running the faster Fruchterman-Reingold layout algorithm (Fruchterman & Reingold, 1991) on each time-step independently. For force-based algorithms that perform iterative improvements, instability is avoided by starting from an initial layout obtained by interpolating between the layout at the previous time step t_{i-1} and the global layout. Nodes that appear at this point are initially positioned at their coordinates in the global layout.

All layouts, global and local, are stored in memory, so that the exact same layout is reused when navigating back to the same time step. As the user manipulates the layout adaptation slider, the interpolated layout is calculated and then relaxed to remove overlaps.



(a) (b) (c) (d) (e) (f) (g)

Figure 4.2.: *Staged transitions with change highlighting (node fill colors describe arbitrary, domain-specific, attributes of those nodes): (a) initial state, (b) element removal (red halos), (c) remaining elements only, (d) layout adaption, (e) remaining elements at their new position, (f) element addition (blue halos), and (g) final state.*

We found an almost fully locally-optimized layout (80% local layout, in other words 20% stabilization) to be a good tradeoff for the various data sets we tested, which we use as default value.

Node Operations

While global and local layouts, as well as their interpolations, change the entire layout, users can pin individual nodes to make them stay in the same position. Nodes are pinned by a double left click and cause the global and local layouts to update accordingly. To preserve the mental map of the network, the original global and local layouts are stored when nodes are pinned, and re-stored when all nodes are un-pinned. A right click on a node highlights it within all small multiples and shows its trajectory in the graph view in light yellow.

4.3. Staged Animated Transitions

Staged animated transitions with change highlighting are designed to help users understand *what* changes occur in the network's topology and *where*, while navigating through time steps (Figure 4.2).

4.3.1. Design Goals

In order to address the problems of animations for graph visualizations (see Section 4.1) we define design goals for the transitions in GraphDiaries. In addition to common design goals for animations such as smoothness, aesthetics and intuitiveness (Chevalier et al., 2010), and in accordance with the goals for a good graph animation as defined by Friedrich and Eades (Friedrich & Eades, 2001), the design of our transitions is based on the following goals:

D1 **Separation of Concerns**—Staged transitions avoid overloading users with too much information at a time. It allows them to focus on each type of event in turn (WHAT). Low-level changes, which account for all types of higher-level changes, can be split in three stages: *node and edge removal*, *node repositioning* and *node and edge addition*.

D2 **Visualization independence**—The visual encoding of transitions must not interfere with the visual encoding of network data (node and edge shapes, color, visual elements etc.) or any user driven annotations (e.g., selection of nodes) . We want our transitions to be integrable into all kinds of existing node-link visualizations and visual mappings.

D3 **Controllability**—As implied by multiple studies (e.g., Farrugia & Quigley, 2011; Archambault et al., 2011a), users should be enabled to

(i) control the animation speed,
(ii) directly navigate between time slices,
(iii) freely navigate inside the frames of a transition, as well as
(iv) interrupt transitions at will.

Controllability is important for two reasons: it enables users to focus on graph elements (WHERE) and to understand complex changes (WHAT), possibly playing the transition back and forth multiple times at low speed; it also enables them to quickly browse through or skip transitions or particular stages within transitions of low interest to them (WHEN).

D4 **Ad-hoc Transitions**—As users interact with the visualization, they must be able to change other visualization parameters, such as the graph layout or the timeline's granularity, independently. Other than in existing "graph movies", any transition should be calculated on demand, taking into account the current state of the visualization (e.g., layout, visible graph elements, current visual mappings). This includes the ability to show transitions between non-adjacent time steps to allow the comparison of arbitrary time steps.

We explored various implementations of the above goals, iterating on the interaction design and fine-tuning the parameters through pilot tests. We compared the different options considered for both the staging of transitions and the interactions that control temporal navigation. The following sections describe our final design, relating to the features of the original design goals.

4.3.2. Transition Stages

Staged transitions in GraphDiaries can be triggered between any two time-steps t_i and t_j, not necessarily adjacent (D4). Stages correspond to the three types of low-level topological changes (D1), as illustrated in Figure 4.2.

1. **Remove Elements** (300 ms) — A red halo fades-in around each node and edge that is no longer present in t_j (Figure 4.2(b)). Edge halos fade-in slightly later than node halos to emphasize the perception of affected nodes in clusters and dense regions. Then, all elements involved in the removal fade-out along with the associated halos.
2. **Transform Layout** (600 ms) — Remaining nodes and connections get smoothly moved to their new position in the layout of t_j using a slow-in/slow-out pacing function (Figures 4.2(c),(d) and (e)). This stage has a longer duration to help users track node position changes. Changes to domain-specific attributes, encoded using, for example, node fill color or node size, also get animated during this stage.
3. **Add Elements** (300 ms) — The last stage adds new nodes and edges by fading them in, accompanied by blue halos that vanish thereafter (Figure 4.2(f)).

In cases where $t_i > t_j$ (user changes from a later to an earlier time step), nodes which are present in t_j but not in t_i are removed while highlighted in red, and nodes which are present in t_i but not in t_j are added while highlighted in blue. Thus, halo coloring *does not convey temporal order*, but type of change.

We tested multiple alternative designs for our transitions. We discarded the option of first showing element insertion, then changing to a new layout, and finally showing element removal, because it significantly increases the number of visual elements on screen during the transition. These elements must potentially be moved in the second stage, causing additional distraction. Furthermore, elements that are present in both time steps are not distinguishable. We also considered separating node and edge changes in two stages, but this increases transition time even more by making the transition appear less smooth and changing regions harder to identify. Furthermore, it introduces ambiguities such as the following: a node disappearing implies that its incident edges disappear as well; but when first removing edges and then nodes, users whose attention gets caught by a particular node might no longer know whether this node was connected or isolated before the staged transition started.

We also tried to run stages in parallel, either partially or completely, for example, repositioning nodes while removing or adding elements, or start fading-in new elements before the fade-out of removed elements ended. However, we found that this option contradicts D1 in the sense that users cannot focus on a particular type of change; staging allows them to anticipate when a certain type of change will happen. Overall, we observed that as networks become denser and as changes are more frequent, staging should be favored over shorter-but-more-confusing transitions that run all animations in parallel (fade-in of new nodes, fade-out of removed nodes, repositioning of remaining nodes).

(a) Changes at low zoom level (b) Changes at high zoom level

Figure 4.3.: *Size of node halos is independent from the zoom level, allowing for analysis at different levels of scale. (a) Using a low zoom level, change highlighting emphasizes changing subgraphs, while (b) a high zoom level reveals details.*

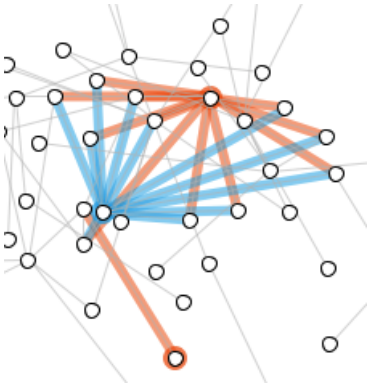


Figure 4.4.: *Direct difference view between two time steps, showing the replacement of a very central node.*

4.3.3. Change Highlighting

We use halos to highlight changing nodes and edges rather than coloring them directly, so as to avoid interfering with existing visual encodings (D2), instead making it possible to visually encode, for example, temporal network measurements such as dynamic centrality, or domain-specific data attributes (Lerman, Ghosh, & Kang, 2010; Federico, Pfeffer, Aigner, Miksch, & Zenk, 2012b). Figure 4.2 shows that halos are still visible when node fill color encodes a domain-specific data attribute. A recent study by Archambault, Purchase, and Pinaud (2011b) confirms that using color to highlight changes between two graphs improves users' performance, compared to a simple animation or no animation. Further evidence about the benefits of explicit change highlighting in comparing diagrams is found by Zaman et al. (2011). While there is no strong agreement in the community about which colors to use to encode those changes, we argue that red and blue are relevant

choices as they feature a significant contrast in hue and are readable by people impaired by color blindness.

Halos around nodes and edges have a constant, scale-independent thickness, which guarantees that changes will always be clearly visible, no matter the network's size and zoom level (Figure 4.3). Holding the *shift*-key while hovering any small multiple in the timeline view highlights the direct differences between the network in the Network view (Figure 4.4) and the one in the hovered thumbnail; blue elements are present only in the hovered thumbnail, red elements are only present in the current (reference) time step.

The duration of each stage of a transition was fine-tuned manually. While a total duration of 1.2 seconds might seem long, it is necessary to actually enable users to keep track of the complex changes that occur (D1). However, as users might not always be interested in all stages of a transition depending on the task at hand (depending on the WHAT component of the task), we enrich the animated transitions with interaction techniques that let users quickly skip or fast-forward them (D3), while navigating through time (WHEN).

4.4. Temporal Navigation

Users navigate through time using either the thumbnail previews, the time slider, or the animation control panel. Holding *shift* while moving the mouse wheel to zoom, scales the time line horizontally. The Time Control Panel mentioned earlier (Figure A.2(g)) provides standard playback controls, including playback speed, looping and temporal aggregation (hours, days, weeks..). Changing temporal granularity creates temporal aggregation of the network's steps. Thumbnails get updated and the network view shows the same transitions as described in Section 4.3.

We defined methods to navigate over time and interactively control staged transitions both across time steps (*inter time-step navigation*) and within transitions (*intra time-step navigation*). In the timeline, red, white and blue sections visually identify the three stages of a transition, thus facilitating intra time-step navigation when dragging the slider (Figure A.2(b) and Figure 4.5).

Inter Time-Step Navigation

Users move between adjacent time steps using the left and right arrow keys. To jump between non-adjacent time steps without going through the intermediate ones, users simply click on the corresponding thumbnail. In both cases, the main graph view gets smoothly animated according to the staged transition technique described earlier, providing details about *what* happens *where* in between the two time steps.

Intra Time-Step Navigation

Users can control a staged transition's unfolding in various ways, depending on whether they are trying to get an overview of changes through time (WHAT and WHEN), are tracking a particular element over time (WHERE and WHAT), or searching for a particular event (WHEN and WHERE): We provide four options for controlling the duration of and the position within a single transition so that users can adapt navigation to the current task, either overview, tracking or searching for a particular event related to a graph entity (WHAT) or a location (WHERE):

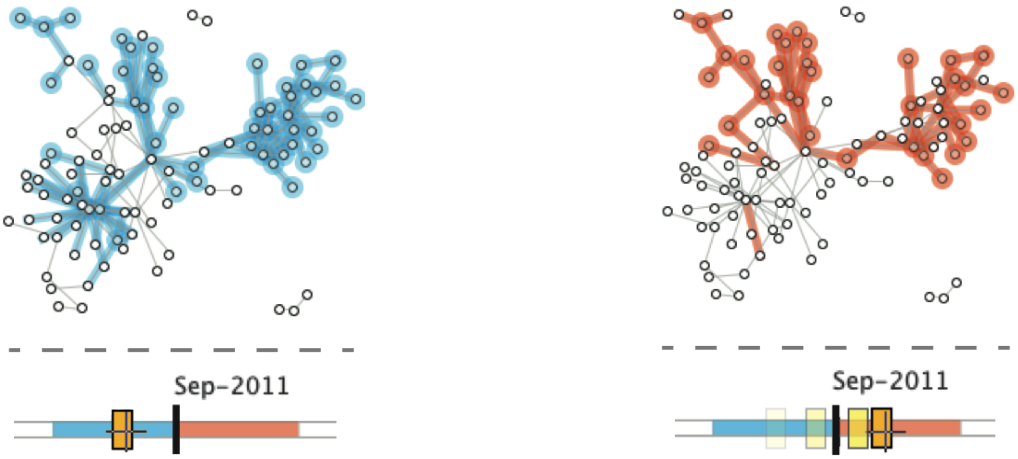
a) **Full Transition** — Pressing and holding down the left mouse button on a thumbnail, or keeping an arrow key depressed, runs the full staged transition.

b) **Interrupt and Finish** — Releasing the mouse button or key while the transition is running interrupts it. The remaining stages are **played fast-forward** (200 ms), all in parallel, in order to guarantee perceptual continuity and help users preserve their mental map of the network.

c) **Skip Animations** — Clicking on a thumbnail or hitting an arrow key (quick press/release) jumps to the target time step **without any animation or highlighting**, to allow browsing through steps very fast. This is useful when the details of changes between two particular steps are not so important or he uses a global layout to quickly see where the graph changes (WHAT is given, the user explores time and affected graph elements WHEN, WHAT).

d) **Interact** — Controlling the animation's pace with a time slider can be very effective when exploring transitions. We support this through direct manipulation of the time slider, with the red, white and blue zones between steps delimiting the three stages of the transition. Figure 4.5 illustrates the use of this strategy to identify elements that exist only during one time step: by quickly moving the slider back and forth between the blue slider part of the previous transition and the red part of the following one.

The slider requires the two compared time steps to be adjacent on the timeline. In order to use the slider on non-adjacent time steps, users can create a new timeline just below the default one (Figure 4.6(a)). Dragging a thumbnail from the default time line to the new timeline duplicated the time step. Users can drag several time steps and consequently use the slider to navigate between them. Difference highlighting on the new timeline layers show differences between the time steps on only that layer. Thumbnails are removed by right mouse click. If the graph has many time steps, thumbnails on the timeline get squeezed, using the mouse wheel (Figure 4.6(b)).



(a) Additions in September 2011 (b) Removals in October 2011

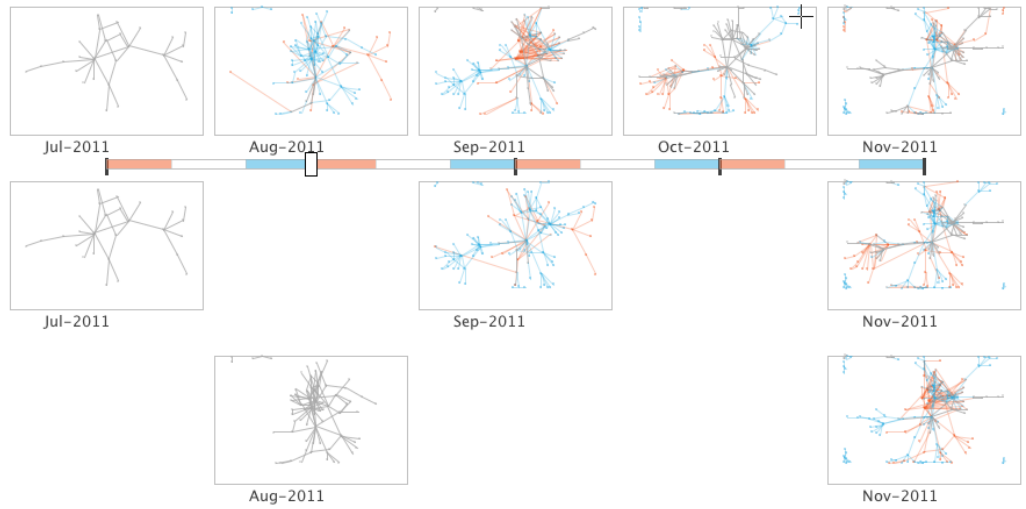
Figure 4.5.: *Dragging the yellow time cursor around the tick that indicates the time step for September 2011 (*Sep-2011*), shows which nodes and edges were (a) added from August to September (blue elements) and (b) which were removed from September to October (red elements). The example shows that a major part of new nodes added in September have been removed again in October*

4.5. Dynamic Node Queries

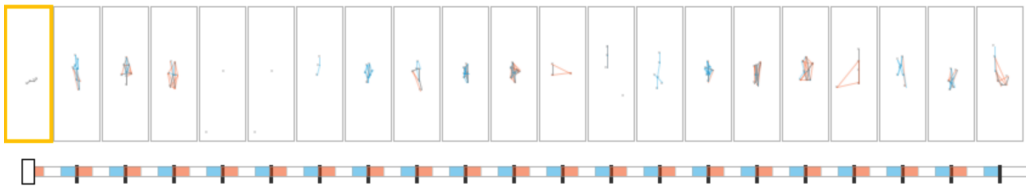
Changing layout and disappearing nodes can make it hard to follow specific sets of nodes and subgraphs. GraphDiaries provides a mechanism to highlight node sets over time. This feature, called *node queries*, is similar to the selection highlighting feature available in ScatterDice (Elmqvist, Dragicevic, & Fekete, 2008). Queries are created by lasso selection and are represented by a colored halo around the nodes that are part of the query, plus a convex hull polygon that encompasses all those nodes. Figure 4.7 shows a node query during an animated transition. Queries can be refined by the user at any time to include new nodes. Node queries are managed in the user interface panel depicted in Figure A.2(e).

4.6. Controlled User Study

To evaluate the potential benefits of staged transitions and associated interactive navigation techniques, such as the possibility to smoothly navigate between non-adjacent time steps, we conducted two controlled experiments. Evaluating all factors and combinations of techniques in GraphDiaries would require multiple user studies, each one looking at a subset of factors in isolation. The first experiment measured participants' performance on a set of three tasks, each one covering one of the dimensions of our taxonomy (WHERE,



(a)



(b)

Figure 4.6.: *Operations on the timeline: (a) Timeline with additional track on which the user placed selected time steps. Differences between thumbnails are computed on the basis of the time steps present in this track only. (b) Thumbnails in the timeline get squeezed to fit more timesteps.*

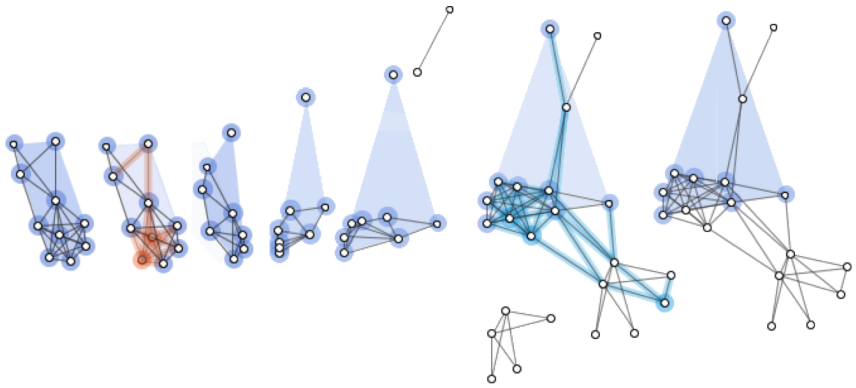


Figure 4.7.: *Node query during transition.*

WHAT, WHEN). The second experiment was a follow-up study to obtain additional empirical data about participants' ability to assess instability in dynamic networks.

Compared to previous evaluations of graph visualization techniques (e.g. Farrugia & Quigley, 2011; Archambault et al., 2011a), we favored higher-level tasks that involve

observation, tracking and comparison of attribute-based subgraphs, not necessarily corresponding to a single connected component. Our tasks possibly require non-linear navigation patterns, for example, revisitation of a given time step multiple times, compare non-adjacent time steps, and tracking different kinds of changes between steps. For graph navigation, we believe low-level tasks, such as tracking one specific node over time or detecting the presence of particular edges in two specific time steps, do not properly reflect realistic network exploration tasks and would not be very informative as to the techniques' efficiency. However, tasks that are too high-level by requiring extended graph knowledge or experience in mapping user tasks into visual tasks, are hard to control and to compare in a controlled user study, which aims in investigating the general understandability and benefits of technique.

4.6.1. Techniques

The primary goal of this study was to assess the potential benefits of techniques that support navigation in time and the impact of different visual feedback strategies to convey changes. For a first assessment of our design choices, we compared GraphDiaries to two baseline techniques: video animation and flip book. We compared conditions in which different time navigation capabilities were enabled. The interface components made available were the same across conditions: only the graph view (Figure A.2(a)), and the timeline with small multiples (Figure A.2(b)). The three conditions were as follows:

FB: **Flip Book** provided a static representation of the graph at each time-step, like an image viewer or a file explorer with content preview. Users could switch between any two images but there were no animation between time steps. Graphs were replaced instantaneously. To jump between time steps participants either clicked on the thumbnail in the timeline, or used arrow keys.

VA: **Video Animation** allowed participants to navigate using a video player metaphor, as in Gephi (Bastian et al., 2009) and TempoVis (J.-W. Ahn et al., 2011) Animations were shown between consecutive time steps only and showed all changes at the same time: added nodes and edges fading-in, removed ones fading-out, and all others moving to their position in the target step's layout. Animation time was 1 second. Participants could play back and pause, as when watching videos. As in the FB condition, participants could either click on the corresponding thumbnail in the timeline or use arrow keys to navigate between consecutive steps.

GD: **Graph Diaries** provided participants with the major techniques presented earlier in this chapter, extending the capabilities of VA: *staged animation* with *change highlighting* between any step, *inter and intra time step navigation*, and *difference visualization* on the thumbnails. Because of the staged nature of the transitions, default animation time was set to 1.2 seconds, as explained earlier. All other

features of GraphDiaries: layout stabilization, node queries, temporal aggregation, animation speed change and history, were removed.

All techniques used the layout strategy described in Section 4.2.2 with a stabilization of 20% (almost fully locally-optimized layout). Node positions were calculated and stored once for each data sample in order to ensure the exact same layout across techniques. Labels where shown on-demand when hovering nodes. All datasets fully fitted on screen at nominal scale. Panning and zooming were thus disabled to avoid noise in the experimental data due to uncontrolled differences in participants' spatial navigation strategies.

4.6.2. Tasks

Participants were asked to answer questions about a real-world co-authorship network made of more than 10,000 authors from 200 research groups between 2005 and 2009 (5 time steps). Nodes represent individual authors. In order to convey group membership, authors that belonged to the same research group shared the same color. For each task, we used data samples consisting of approximately 100 nodes in 7 groups (average, per time step). Group membership was not directly related to the network's topology. In a given time step, an edge links two nodes when the two authors have collaborated on at least one publication during the corresponding year. Participants had to answer the following questions:

T_{Size} **— In which year is the red group largest? (**WHEN**)** Participants had to navigate through all time steps and compare them, in order to find the time when the red group was largest. To input their answer, they had to press the space bar and select the correct year from a pop-up menu.

T_{Inst} **— Which group features the most changes over time? (**WHERE Groups exchanged nodes over time, for example, they lost some nodes and gained new ones. Participants had to observe all groups over the years, and eventually click any node from the group that featured the most changes.

T_{Trend} **— What is the trend of the red group? Does it grow, does it shrink, does it remain stable, or is it unstable? (**WHAT Participants had to spot a trend over the years. Groups that grew actually doubled in size. Those that shrank halved. Both did so in a non-monotonic way. Stable groups kept a set of constant core members that was larger than half the size of the group. Unstable groups did not feature any such set of stable core members. They could possibly gain, loose or exchange all of their members over time. To input their answer, participants had to press the space bar and select the correct answer from a pop-up menu.

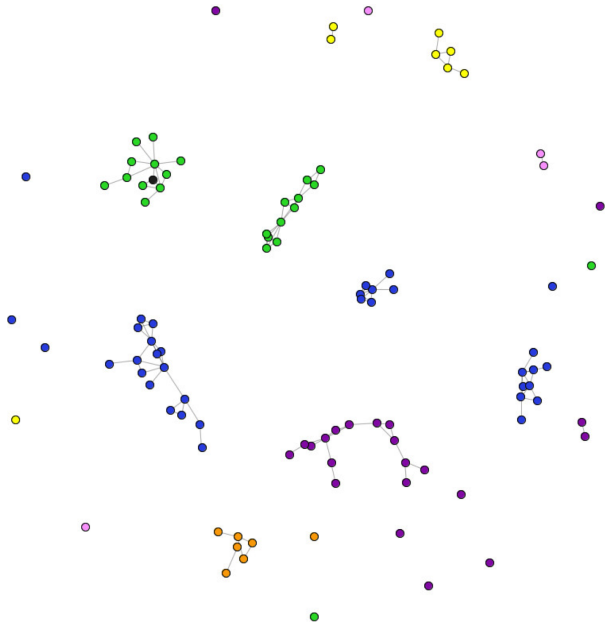


Figure 4.8.: *Example of the data set as used and laid out in the user study. Nodes are colored by research group. The actual background in the experiment was a very dark gray.*

4.6.3. Datasets

A major problem with these tasks is that difficulty can vary significantly with each dataset's complexity. Observed differences between techniques can actually stem from this variability if dataset complexity is not controlled and counterbalanced properly across conditions. But comparing the complexity of different datasets is difficult, especially if they have not been created artificially, carefully controlling their characteristics.

To guarantee equivalent conditions, we extracted and analyzed data samples (subgraphs) from our main dataset for each task, and reused them across all participants. Each task required special conditions and tuning to make sure that there would not be too much ambiguity with respect to what answer was the correct one.

We re-used the same datasets *across techniques*, thus allowing for a fair comparison between conditions. Each data sample was used in 3 trials, one per technique. However, simply reusing datasets across conditions would have been risky, as participants might have remembered answers or partial answers from previous trials, resulting in an uncontrolled learning effect. To minimize possible asymmetrical transfer between conditions, we mirrored and rotated the three instances of each network. In addition, each time a dataset appeared, node labels were anonymized by randomly assigning them popular English names at runtime.

The data samples were obtained as follows and were rendered as illustrated in Figure 4.8. Nodes were filled with the color of the corresponding research group, using the

Set1 9-color scheme from *ColorBrewer* (*ColorBrewer*, 2002) so that participants could identify groups pre-attentively.

T_{Size} — Per dataset, seven research groups were randomly extracted from the original network and one group was chosen to be the target one (colored red). The size of that group was analyzed over time and data samples were selected by hand in order to remove sources of ambiguity. A group, at the time step when it was at its largest size, always featured at least two more nodes than in any other time step.

T_{Inst} — All 200 groups from the original network were analyzed, based on particular types of changes: *size*, *nodeGain* (nodes added per time step), *nodeLoss* (nodes removed per time step), and number of *constantNodes*. Each dataset was composed of 6 groups with a low rate of change:

$$avg(nodeGain) < 3, avg(nodeLoss) < 3, avg(size) > 6.$$

The additional target group was extracted from the original network and featured a high rate of change:

$$avg(nodeGain) > 7, avg(nodeLoss) > 7, avg(size) < 15.$$

T_{Trend} — For each dataset, 7 groups were randomly extracted from the original network. One group was chosen to be the target one (colored red). Datasets were selected by hand to ensure that the target group featured a clear trend, either growing, shrinking, stable, or unstable.

4.6.4. Design and Apparatus

The first experiment followed a within-subject full-factorial design with the 3 earlier-mentioned techniques ($Tech \in \{\text{GD}, \text{VA}, \text{FB}\}$) and the 3 tasks described above ($Task \in \{T_{Size}, T_{Inst}, T_{Trend}\}$) as independent variables. The resulting 9 conditions were counter-balanced using a Latin square, blocking by *Tech*. For each condition, participants were presented with 4 training trials, followed by 5 actual measurement trials. Each trial used a different dataset. The presentation order of datasets was the same for all participants.

On average, the experiment lasted 70 minutes. It was divided into two sessions to avoid fatigue due to the high cognitive load involved in performing the tasks. Two techniques were tested in the first session (50 minutes). The remaining one was tested in the second session (20 minutes), which had to take place at least one hour after the first one. Participants were allowed to rest between each trial.

Participants were instructed to favor accuracy over speed, for example, to avoid making mistakes. Due to the complexity of the tasks, each trial was limited to 90 seconds. After 60 seconds, the screen flashed briefly, and a countdown for the remaining 30 seconds was

shown. Once an answer had been selected, both the right answer and the participant's answer were shown.

We asked eighteen volunteers (four female), ranging in age from 24 to 44 years old to participate in this experiment. All of them used computers daily, had normal or corrected-to-normal vision and were not color blind. The experiment was conducted using a 2.66Ghz iCore 7 MacBook Pro with 4GB of RAM and a monitor resolution of 1440×900. The interface was implemented in Java 6 using the ZVTM toolkit (Pietriga, 2005). Background was set to black to minimize visual fatigue. Participants interacted using a mouse and an external keyboard. During training, animation speed decreased from 2 seconds initially to the default duration of 1 second for VA and 1.2 seconds for GD.

The two main measures were error rate and task completion time. The timer started as soon as the dataset showed up on screen, and stopped when participants either clicked a node (for T_{Inst}) or hit the space bar (for the other two tasks). Error rates were computed differently for each task. In T_{Size}, time slices were ranked according to the size of the target (red) group. For T_{Inst}, groups were ranked according to their rate of change: $avg(nodeGain) + avg(nodeLoss)$. Error was equal to the position in this ranking. For T_{Trend}, the answer was either right or wrong as answers were nominal.

4.6.5. Hypotheses

Our hypotheses were as follows:

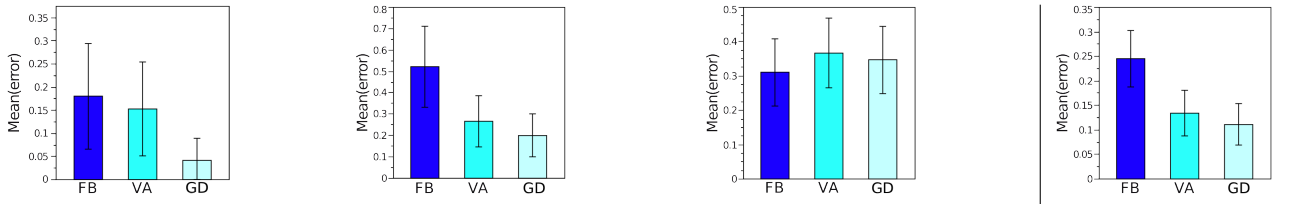
H_1 For all three tasks, error rate is lower when using staged transitions (GD), as this technique helps better keep track of changes between time steps.

H_2 For the same reason, completion time does not increase significantly when using staged transitions (GD), despite the longer duration of animations and their higher visual complexity.

H_3 Participants use features that enable them to transition between non-adjacent time steps, when available.

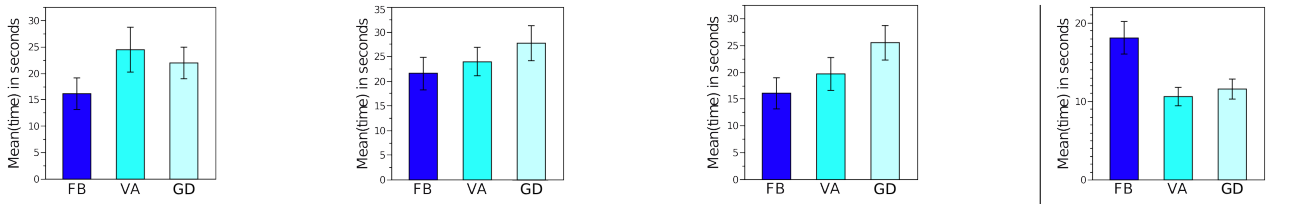
4.6.6. Results

A SHAPIRO-WILK test showed that measurements of time and error were not normally distributed. The measurements distribution could not be corrected using either a logarithmic or BOX-COX transformation. We thus performed a non-parametric MANN-WHITNEY-WILCOXON (Mann-Whitney U) test for pair-wise comparison between techniques, for each task. During the experiment, we realized that one particular dataset used for T_{Size} contained changes that were too hard to detect for participants. The difference between the largest two sizes was only 2 nodes, for an average group size of 18 nodes. We



a) Task = T_{Size} b) Task = T_{Inst} c) Task = T_{Trend} d) Task = T_{Change}

Figure 4.9.: Error *rate per* Tech × Task. *Error bars show the 95% confidence limit of the mean.*



a) Task = T_{Size} b) Task = T_{Inst} c) Task = T_{Trend} d) Task = T_{Change}

Figure 4.10.: Time *(seconds) per* Tech × Task. *Error bars show the 95% confidence interval of the mean.*

subsequently removed the corresponding trials from our analysis, as this noisy set would not have yielded meaningful results.

All analyses are performed by *Task*, as error rates are measured differently across tasks. T_{Size} (Figure 4.9-a): GD features a significantly ($p<0.04$) lower error rate (avg. 4%) than FB (avg. 18%). VA performed similar to FB (avg. 15%) with a near-significant difference compared to GD ($p<0.069$). There was no significant difference between FB and VA. T_{Inst} (Figure 4.9-b): GD features a significantly ($p<0.011$) lower error rate (avg. 21%) than FB (avg. 52%), with VA in-between (avg. 27%) and a near-significant difference ($p<0.072$) compared to FB. There was no significant difference between VA and GD. T_{Trend} (Figure 4.9-c): we did not observe any effect of *Tech* on *Error*. All three techniques feature relatively similar error rates.

T_{Size} (Figure 4.10-a): FB (avg. 16.2s) is significantly faster than both VA and GD (avg. 24.5s, $p<0.0001$ and 21.9s, $p<0.0001$, respectively). GD was not significantly faster than VA. T_{Inst} (Figure 4.10-b): FB features a significantly ($p<0.005$) lower time (avg. 21.6) than GD (avg. 28.4s), with VA in-between (avg. 23.9s) and not significantly different from the other two. T_{Trend} (Figure 4.10-c): FB features a significantly lower time (avg. 16.1s) than GD (avg. 26s, $p<0.001$) and VA (avg. 19.7s, $p<0.007$).GD is significantly slower ($p<0.004$) than VA.

4.6.7. Follow-up Experiment

T_{Trend} did not yield significant results in terms of error rate, which is our main measure of performance. Despite intense piloting, such higher-level and complex tasks are difficult

to control. We decided to redesign this task and run a follow-up study. We switched from four possible answers to two, and created the target group artificially, inserting ground truth in real-world data so as to better control this group's behavior. The new task was as follows:

T_{Change}—(WHAT) **How does the red clique behave? Is there a high turnover, for example, are there many nodes coming and leaving, over time? Or is there a larger constant core of members over all years?** The core was made of nodes that remained in the group from the beginning to the end. If the number of nodes in the core was larger than half the size of the group, the group was stable; otherwise it was unstable. Participants had to press the *space bar* and select the correct answer from a pop-up menu.

We extracted datasets from the co-authorship network we had used in the first experiment. For each dataset, 6 groups of nodes were randomly selected, and one group was artificially created according to predefined figures to guarantee its stability or instability (depending on the trial). This target group had between 13 and 17 nodes, including a constant core of nodes. Core nodes never left the group. Other nodes remained in the group for one to three years. In the *unstable group* condition, the target group's core size was lower than 50% of the overall group size. In the *stable group* condition, the core's size was above 50%. The difference in size between stable and unstable was either 30%, 15% or 7.5%, corresponding to three levels of difficulty. Although the last value seems fairly small, we did make sure that there was no ambiguity when performing the task visually. It is also important to bear in mind that the task was about observing changes of nodes, rather than changes in group size.

We asked the same 18 volunteers to participate in this experiment. Time between the first experiment and this follow-up study never exceeded two weeks. The only task tested was T_{Change}. We compared the same three techniques. We added a new feature to the environment that enabled participants to toggle visibility of all node labels at once. While this can potentially introduce noise in the data, participants needed a way to compare nodes across graphs. We kept track of the status of node labels (shown or hidden) in the experiment's logs.

Results for this task show that *Tech* has a significant effect on *Error*. FB features a significantly ($p<0.004$) higher error rate (avg. 25%) than VA and GD (avg. 13% and 11%, respectively), which are not significantly different from one another (Figure 4.9-d). *Tech* also has a significant effect on *Time*. As shown in Figure 4.10-d, FB features a significantly ($p<0.001$) higher task completion time (avg. 18.1*s*) than VA and GD (avg. 11.6*s* and 10.6*s*, respectively), which are not significantly different from one another.

4.6.8. User Strategies

Participants were asked to report their strategies and subjective preferences in a post-hoc questionnaire. Further information was retrieved by analyzing low-level interaction

logs: keyboard and mouse events, as well as somewhat higher-level information such as the order in which participants visited the five time steps for each trial, when were node labels displayed, how much was the time slider used. Examining these logs, we identified various common and alternative strategies for exploring dynamic networks. The main observation is that temporal navigation in the data follows Shneiderman's mantra (Shneiderman, 1996): users first wanted to have an overview of the data through time, and only then did they focus on particular time steps in more detail. While this observation holds for all tasks and techniques, other observations were made for specific tasks, as detailed below. We also observed that 27% of the participants never used the mouse. This means that they never used the slider, and never compared non-adjacent time steps (indeed, the mouse was necessary for both).

T_{Size} **— Find Time Step** While participants were able to quickly go from time step to time step with FB and GD, things were less straightforward with VA. They either watched the animation step by step, or dragged the time slider. Participants made extensive use of the non-adjacent time step comparison feature with GD, while they barely used it in conjunction with FB. While this feature was not available in VA, it is noteworthy that participants did not employ any feature for comparison between time steps with VA: in most cases, they simply navigated over time using video controls, not going back and forth.

T_{Inst} **— Find Network Entity** This task was about spotting nodes entering and leaving groups over time. In addition to the amount of color that was shown with groups, the middle step of staged transitions (GD) made it easy to identify how many (and which) nodes did not change. If a group was much smaller during this stage than at the beginning or end, it was probably unstable. However, changes had to be observed throughout the network. Participants employed very different strategies; FB allowed for quick navigation which led them to quickly visit all time steps in linear order and observe changes in the layout that might result from unstable groups. With VA, participants iterated less frequently over all time steps, except those participants who used the slider, the latter making navigation faster. 50% of all participants who used the mouse compared non-adjacent time steps in the FB and GD condition.

T_{Trend} **— Characterize change** This task was the most difficult for participants as observed from our post-hoc questionnaire, and we did not observe any statistical difference between conditions. However, we could observe that participants applied a strategy similar to that employed in T_{Inst}; they observed all time steps first and heavily switched between steps in the GD condition. Where it was possible to directly compare the first and last time steps (FB and GD) , participants could estimate the right answer when the group was growing or shrinking. However, due to the experimental conditions—clear differences in the data between growing, shrinking, unstable or stable—this strategy worked in the experiment.

T_{Change} **— Characterize change:** This task was the only one where enabling and disabling all node labels at once was allowed. Since this task was about tracking nodes joining and leaving a group, we expected the task to be very hard in the FB condition. However, despite the difficulty of the task, results indicate that very few participants (four) actually used labels in FB and in VA. Among them, all but one also displayed labels in the GD condition. This seems to indicate that having to read and memorize node labels was considered as inconvenient and unnecessary. As expected, many participants (80%) switched between non-adjacent time steps, when possible.

4.6.9. Discussion and Limitations

The goal of this study was to investigate the potential benefits of staged transitions and other features such as the possibility to smoothly navigate between non-adjacent time steps and to control the type and speed of transitions. We wanted to assess their impact on user comprehension of the higher-level changes that occur in dynamic networks.

Our results show that animations significantly decrease error rate, and suggest that staged transitions offer further improvements for some tasks, thus partially supporting H_1. Animations increase task completion time for tasks T_{Size}, T_{Inst}, T_{Trend}, but decrease it for T_{Change}, since participants spent more time memorizing node labels and comparing graph states. Comparing the first three tasks, the difference is significant in terms of statistical analysis, but its magnitude is relatively small. Thus, H_2 is supported. A lower error rate should be favored over a small decrease in task completion time in real work situations, and we interpret those results as generally favorable for GD. Quantitative results are in accordance with user preferences collected through the post-hoc questionnaire: GD was ranked as the best technique by 80% of the participants, and FB was ranked last by 72% of them. Our results are in accordance with Archambault et al. (2011a) who found that for tasks related to node and edge appearance, animations decrease the number of errors when compared to small multiples, but increase task completion time. Further studies (Zaman et al., 2011; Heer & Robertson, 2007) suggest that users prefer accurate techniques that they can trust to faster, but less reliable, ones.

Error: Results suggest that animations played a major role in participants' performance: the two techniques that relied on it, VA and GD, yielded consistently and significantly lower error rates than FB (Figure 4.9). Animations seemed to play a crucial role when tracking incoming and outgoing nodes. When animations were not available, labels sometimes helped to solve the task, but significantly increased cognitive load and fatigue according to several participants. When animations were available, the simple fading of nodes and edges provided by VA helped, but the animation was often not sufficient to help participants track more than a few changes in the limited screen area. Changes scattered throughout the screen were hard to track, as suggested by the results of T_{Inst}, in which the entire network had to be observed. GD provided an advantage in that respect, as it

used color to convey changes in a pre-attentive manner and enabled participants to spot and identify the kind of changes in a much larger area. Subjective feedback indicates that participants found highlighting *overall useful* (44%) or *useful* (36%). 66% of them indicated that it made them more confident about their answers.

The significantly lower error rate of GD for T_{Size} suggests that highlighting of changes (halos) played an important role when comparing two time steps. In addition, solving this task required comparing local maxima between non-adjacent time steps; this was made relatively straightforward by both FB and GD, but highlighting was only enabled in GD.

Time: Task completion time was slightly lower with GD than with VA for T_{Size}, and slightly higher for all other tasks (Figure 4.10). We tentatively attribute this difference to the fact that FB and GD enabled direct comparison of arbitrary time steps. FB was always the fastest technique, except for task T_{Change}. This is probably due to the fact that participants had to read and memorize labels with FB, for lack of a more efficient and less cognitively-demanding alternative. It is interesting to note that the average time spent solving a task with FB was far below the maximum time per trial (90 seconds). Participants seldom reached that limit, even if they were not particularly confident about their answer; they considered that spending more time on the task with FB was not worth it, as it was not going to provide more insight. Overall, FB featured lower task completion times mainly because participants gave up, not because they were confident about their answer, as confirmed by the higher error rates with this technique and our observation of participants' behavior during the experiment. This suggests that providing users with animated transitions while enabling them to skip these transitions is a good solution.

Strategies: Only five participants (27%) never used the mouse for interacting with the system; all others adopted non-linear time-navigation patterns in the FB and GD conditions. The reason could be that the interface did not required to pan and zoom in the network view, as well as the fairly limited amount of time-steps in the experiment so that no panning on the timeline was necessary either. Such capabilities would likely have involved the mouse as an interaction device. The most typical temporal navigation pattern was to get an overview by quickly looking at all time steps, then focus on a particular subset of—not necessarily contiguous—steps, switching back and forth to compare them. This supports H_3 and suggests that features enabling direct difference visualization between arbitrary time steps is useful, as those spare users from having to rely on their memory to compare distant time steps, thus lowering the associated cognitive load.

The slider enabled users to scrub and to explore elements that last for a single time step only (see Figure 4.5). As one participant reported with respect to VA, vanishing and occurring nodes faded out at the same moment and their translucency is not sufficient to differentiate both cases. Another method was to move the slider very fast through

across all time slices in both conditions VA and GD. Yet, highlighted nodes could better be estimated.

Tasks: The experimental tasks were chosen to represent the dimensions in our taxonomy on a higher level. However, we had to make some compromises to make those tasks amenable to a controlled experiment. Designing a study and operationalizing higher-level and cognitively-demanding tasks is very challenging, as acknowledged by Archambault et al. (2011a). We had to pilot and iterate on the design, datasets and tasks multiple times. This necessarily entails some limitations. Visually, our tasks mainly focused on the appearance and disappearance of nodes as opposed to links. As mentioned earlier, some topological tasks involving group membership turn into visual tasks of looking for geometrical proximity: nodes close to one another are more likely to be connected, especially when using a locally-optimized layout (see Figure 4.8). Tracking groups essentially comes down to tracking node movements *and* node appearance/disappearance. While staging separates both types of change for better perception, change highlighting results in a higher or lower concentration of red or blue halos in different regions of the representation. Further studies, however, should also include links, for instance asking questions about density and connected components; questions which require some domain expertise to answer, putting some constraints on the participant population.

Data: A limitation is the number of time steps associated with each dataset. Real-world data usually contains much more than five steps. Future studies should investigate the effect of this specific factor, but it already seems reasonable to claim that some features discussed earlier, such as the capability to directly compare and smoothly transition between non-adjacent time steps and quickly browse all time steps, should bring even stronger benefits in situations where the number of time steps is larger. Currently, we considered networks made of approximately 100 nodes but very large networks and the more complex tasks associated with them require aggregation and analytical capabilities that are both complementary to the work presented here and beyond the scope of the generic navigation tasks considered. Furthermore, larger data sets are likely to be more complex, which makes them hard to use in a *controlled* user study, and are more suited to qualitative evaluations.

Complementary Techniques: We did not fully isolate all factors. Our goal was to compare the transition and navigation techniques as complementary elements, not to isolate the effects and contributions of each single low-level interaction technique. We thus cannot say much about the respective contribution of, for example, small multiples and animation to performance improvements. Our results in terms of comparison between plain and staged animations are in accordance with those of Heer and Robertson's study about transitions in data graphics (Heer & Robertson, 2007). Archambault *et al.*'s study (Archambault & Purchase, 2013) suggests that layout stabilization depends very much on the task, and that small multiples and animations are rather complementary for

different tasks. Results from these studies and ours help gain a broader perspective, but further studies are required to gain a more comprehensive understanding of the various factors that come into play.

Finally, we did not test for the effect of the graph layout and different layout stabilizations. However, more extensive studies require more conditions and the usefulness of quantifying the individual effect for every single technique is questionable, since often the benefit of a single technique can hardly be defined, independently from others.

4.7. Conclusion

Showing multiple time steps of the network, one after the other, has clear advantages and disadvantages, as multiple studies have shown. However, the efficiency of animations for representative tasks from our task taxonomy (Chapter 3) can be improved by a) showing different changes at different moments (stages), b) highlight certain low-level changes, and c) complement the transitions with interaction. While evaluating GraphDiaries against techniques commonly found in visualization systems for temporal graph navigation, we observed a minor increase in task completion time, that is compensated by a significant decrease in error rate in favor of animated transitions. The latter improve the perception of changes and provide users with a rich set of exploration strategies.

Animated transitions should be part of any more general system, such as GraphDiaries tries to sketch, and be complemented with additional visualizations and navigation widgets. For the design and evaluation of new techniques, as well as their complementing, we have proposed a task taxonomy in Chapter 3. The goal of any novel technique, extension or integration should directly support higher-level tasks, such as exploring time, tracking elements and perceive complex changes happening to groups of nodes and their internal connectivity.

Future work in animations and integrating techniques should explore the issue of how to convey the temporal evolution of node and edge attributes (e.g. edge weight): we want to extend the scope of GraphDiaries towards the exploration of dynamic multivariate and hierarchical graphs, whose complexity will require new aggregation, navigation, comparison and tracking techniques. Still, GraphDiaries is meant to describe simple but useful techniques and to explore how different techniques support visual exploration and analysis. Techniques including temporal aggregation and enhancing the display and summarization in small multiples remain longer term research goals.

A more general problem is network density and the amount of change happening over time. GraphDiaries as well as most current visualizations and tools (Gephi, TempoVis, etc.) rely on node-link representations, which become less readable with the number of links crossing (Ghoniem et al., 2005). As for animations, they become hard to follow as the number of changing elements increases, especially since he has to keep track of

all the changes across time steps. The following chapters propose techniques to address these specific issues.

5. Comparison of Dense Weighted Networks

Contents

GRAPHDIARIES provides a platform to integrate different visualization techniques and to allow for flexible navigation between times steps of dynamic networks. The results of our experiment showed that we can effectively use the techniques integrated in GraphDiaries for the exploration of, for example, social networks. However, many networks are not as simple and easy to visualize, such as networks representing brain connectivity. Brain connectivity networks can reflect actual physical connections between *regions of interest* (ROI) (anatomical connectivity), aggregated functional correlation (functional connectivity) and actual signals (effective connectivity). These networks are usually (i) very dense or even fully connected, (ii) can dramatically change in a short amount of time, and (iii) edges are weighted and with weights changing over time.

Neuroscientists investigate brain connectivity to answer questions such as how brain connectivity changes throughout development, with aging, or with physical injury; what kind of connectivity patterns are required for a particular functionality; and how these connectivity patterns show variances across individuals and across certain anomaly conditions. A list of precise tasks is provided by my coauthors (Alper, Bach, Henry Riche, Isenberg, & Fekete, 2013) in the first section of this chapter. It shows that many of these tasks can be described as comparing two weighted graphs—either representing two different brains, or two different types of connectivity in the same brain, or the same brain at different time steps, relating to different conditions.

The questions addressed in this chapter are the following:

- ▷ *Which designs allow for effective comparison of two weighted graphs?*
- ▷ *Which techniques are most efficient?*

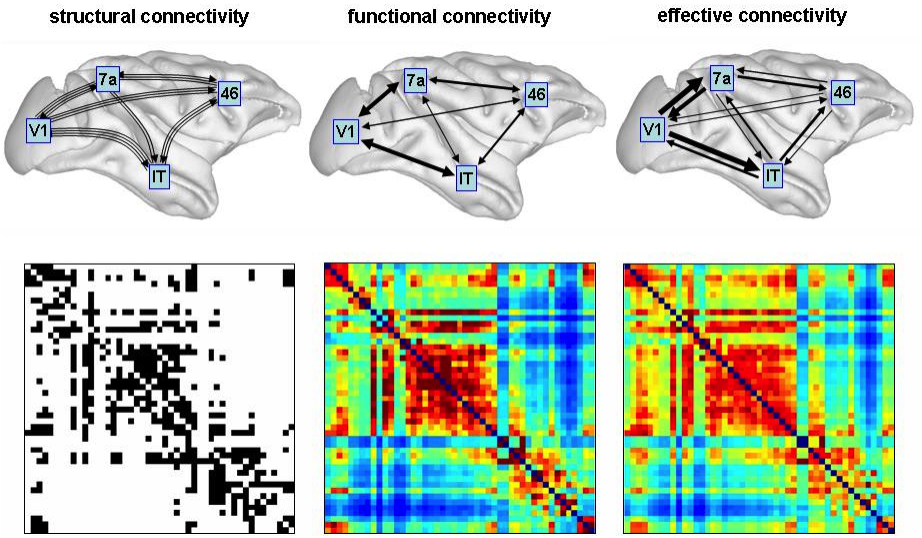


Figure 5.1.: *Three types of connectivity in the brain: (a) structural/anatomical connectivity, (b) functional connectivity, and (c) effective conncetivity. The figure shows schematic brain maps and adjacency matrices. Colors in the matrices indicate the weight of connection using color mapping (blue for low weight via green and yellow to red for high weight).*

In this chapter we develop several visual designs to allow for visual comparison of two dense weighted networks, using node-link diagrams and adjacency matrices. We start with an introduction to brain connectivity and list tasks that are performed by neuroscientists when investigating brain connectivity. We then discuss the strengths and weaknesses of existing visual designs for visualizing brain connectivity, weighted graphs, as well as visually comparing two graphs. After explaining our designs for node-link diagrams and matrices, we report on the results of a controlled user study that assesses the efficiency of our two most promising designs.

5.1. Brain Connectivity

Brain analysts investigate connectivity between *regions of interest* (ROI) in the brain in order to understand the brain's functionality and responsibility of particular regions. An ROI can be anything between a single neuron, neuron populations or anatomically-defined brain regions. These ROIs are related by different types of connectivity, each of which has particular characteristics, explained in the following and illustrated in Figure 5.1.

Anatomical connectivity refers to actual axonal fibers between neurons or fiber bundles between regions. Current diffusion weighted techniques detect fiber bundles on a coarse level. However, invasive tracing techniques can detect actual axonal connections. Changes in the anatomical connectivity are observed within hours or even days, reflecting

learning, rebuilding mechanisms after an injury (Cao & Slobounov, 2010) and even aging (Dosenbach et al., 2010).

The second type of connectivity is *functional connectivity*, which reflects statistical correlation and covariance in the activity of these regions; two regions that are always active at the same time show strong *correlation*, two regions that exhibit equal changes over time show strong *covariance*. Functional connectivity can be meaningful correlated in time intervals above 10 milliseconds and depends on the user's activity during this time.

The third kind of connectivity is called *effective connectivity* and reflects the actual exchange of signals between neurons and ROIs while a subject receives stimuli or performs cognitive tasks, such as observing a picture or reading a text. Signals are exchanged at a very high frequency and can be recorded during the entire time of an experiment. Signals are not exchanged symmetrically, thus the network is essentially a directed graph.

Functional connectivity and effective connectivity result in almost fully connected and weighted networks, where weight indicates correlation, covariance or strength of signal. Duration of time steps can be very low, e.g. 2 seconds, while experimental observations can take up to 5 minutes (see Section 6.6.3 in Chapter 6). Figure 5.1 shows all three types of connectivity in a schematic representation (top) and what the corresponding matrix looks like (bottom).

5.1.1. High-Level Brain Connectivity Analysis Tasks

Through a literature survey and a series of one-hour-long interviews with seven neuroscientists, we identified a list of major tasks in brain connectivity analysis where visualizations are already in use or can be used to aid and enhance the analysis:

1. *Identify network structures that are responsible for a specific cognitive function:* Neuroscientists examine how our brain carries out specific cognitive functions, identify the regions responsible or involved, and study their interaction with each other while executing these functions (Vidal et al., 2012). However, significant individual variations exist. These variations require the comparison of connectivity data of large subject groups to be able to identify common network structures.
2. *Identify effects of anatomical structure on functional connectivity:* Understanding the effect of anatomical structure on the formation of functional relations is a key challenge in neuroscience (Honey, Kötter, Breakspear, & Sporns, 2007; Bullmore & Sporns, 2009). For this purpose, scientists look for correlated patterns across anatomical and functional connectivity, requiring the simultaneous exploration of data for both types of connectivity to reveal complex mappings between ROIs.

3. *Identify alterations in brain connectivity:* Brain connectivity of an individual changes over time—through development and with aging (Dosenbach et al., 2010). Another instance of alteration in brain connectivity is the development of re-routings that restore functionality typically after a local injury (Cao & Slobounov, 2010). In addition, changes in white matter fiber connectivity following behavioral training of a complex skill have also been reported (Scholz, Klein, Behrens, & Johansen-Berg, 2009). In order to understand the temporal evolution of these changes, scientists need to compare connectivity data gathered at different instances in time.

4. *Identify the existence or the loss of patterns in brain connectivity that are associated with anomalous conditions:* Neuroscientists have already shown that certain cognitive anomalies are associated with the loss of specific connectivity patterns (Bassett et al., 2008; Sanz-Arigita et al., 2010). For instance in Alzheimer's disease, it is observed that functional connectivity does not exhibit small-world properties as much as a healthy brain (Sanz-Arigita et al., 2010). Identifying such differences in connectivity patterns is essential for the diagnosis and treatment of these anomalies, which can be done by comparing the connectivity of the anomalous brains to healthy ones.

5. *Identify deviation of an individual's connectivity from a population mean:* The aforementioned tasks often utilize an 'average' brain—the term 'average' encompassing connectivities that are common to all individuals. The average brain, in turn, can be used to characterize an individual's brain by studying the deviation of it from the average brain.

6. *Effective brain parcellation and multimodal connectivity analysis:* Identifying brain structures that act as a unit is an important problem. Automatic parcellation algorithms, based on statistical likelihood of synchronized activation, are employed to identify these units. A variety of statistical techniques along with multiple data sources exist, each yielding different parcellations. For example, an automatic parcellation based only on functional connectivity data can be refined by taking into account fiber connectivity (Li et al., 2012), or anatomy-based parcellation can be validated by functional connectivity relationships between the regions (Beckmann, Johansen-Berg, & Rushworth, 2009). Thus, different parcellation outcomes are needed to be compared to verify the robustness of a technique (Worsley, Chen, Lerch, & Evans, 2005).

7. *Identify effects of local injury:* In clinical applications it is important to predict the effect of a local injury. Local damage to an anatomical structure can have multiple implications on functional connectivity due to the loss of indirect as well as direct connections. Similarly, in image-guided neurosurgery (Archip et al., 2007), identifying optimal surgical paths that minimize damage induced by intervention is also important. In order to be able to make these decisions, neurologists look

at brain connectivity data in a spatial context and observe which anatomical and functional connections are associated with which specific brain locations.

This summary shows that a large number of neuroscientific problems can be facilitated with weighted graph comparison tools. Apart from the last task, for which connectivity data has to be presented in its spatial context (inside the brain volume), all other tasks can be conducted with non-spatial representations such as 2D node-link diagrams and adjacency matrices. Although it is essential for brain connectivity analysis to communicate spatial context of the data to ease the interpretation, non-spatial representations can prove more effective for communicating changes in complex connectivity data, as they relax a set of strong constraints on the visual representation (such as preserving the exact shape of connections in three dimensions). Hence, tools supporting brain connectivity analysis have to offer multiple visual representations—each of which is optimal for a specific set of tasks. In this chapter we focus on connectivity comparison tasks which, at an abstract level, can be expressed as comparison of two weighted graphs.

5.1.2. Use of Visualizations in Brain Connectivity Analysis

Defined as a correlation matrix between specific locations within the brain volume (ROIs), functional connectivity data is often visualized as 3D node-link diagrams reflecting the ROIs' spatial positions in the brain: ROIs are shown as nodes and weighted edges denote the pairwise strength of these node relationships (e.g., Figure 5.2(a)). Although the edges below a certain strength threshold can be hidden, the resulting images are still cluttered and suffer from the known side effects of 3D rendering such as occlusion (Gerhard et al., 2011; Dosenbach et al., 2010). To facilitate comparison, two connectivity datasets can be overlaid within the 3D brain (Worsley et al., 2005), with edge colors denoting correlation and anti-correlation between the end points. However, the clutter and complexity of the visual encoding in these spatial/volumetric representations makes it difficult to perform accurate weighted edge comparison tasks.

An alternative approach is to show the connectivity data as a 2D node-link diagram whose layout is calculated by multi-dimensional scaling or force-directed algorithms (Achard, Salvador, Whitcher, Suckling, & Bullmore, 2006) (e.g., Figure 5.2(c)). These layouts remedy some of the clutter problems by eliminating overlaps and reducing the number of long edges. They are thus adopted especially among those scientists who practice graph-theoretical analysis of brain connectivity.

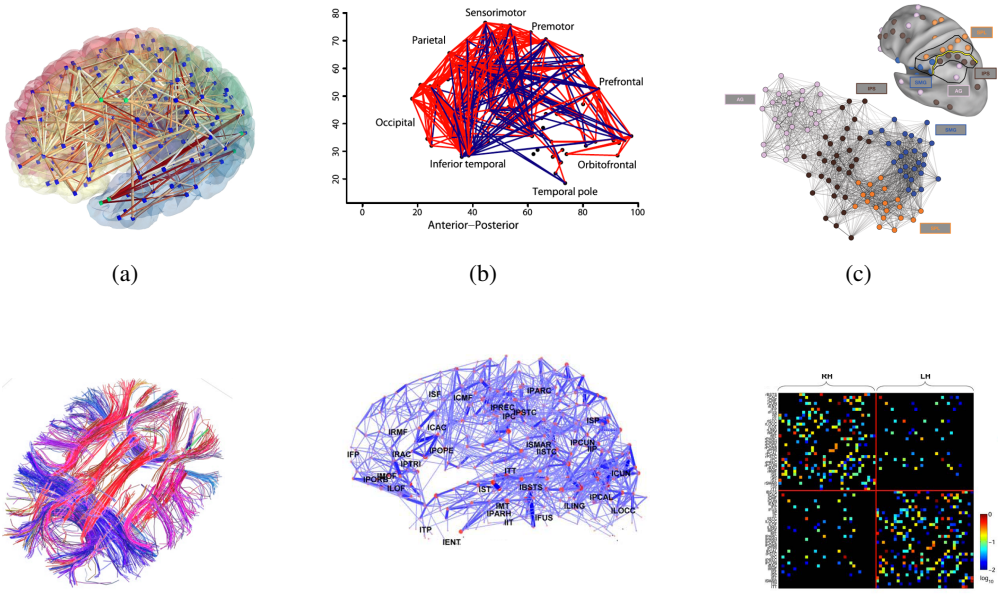
The spatial context of the data, however, is still crucial for an effective interpretation of the data by neuroscientists who are trained to reason with respect to spatial brain regions. Hence, node-link diagrams are often accompanied with a separate spatial rendering of the node positions, with nodes on the two representations being matched by region color and label (Nelson et al., 2010). To preserve the spatial character of the data to some extent, many 2D connectivity node-link diagrams adopt a biological layout where node

positions are projections on 2D planes defined by two of the standard anatomical axes (e.g., ventrodorsal, anteroposterior, left-right; e.g., Figure 5.2(b)) (Achard et al., 2006). These layouts communicate spatial properties of the data coarsely, helping scientists orient themselves in the graph, while enabling the use of color for encoding other information such as change across two states (Richiardi, Eryilmaz, Schwartz, Vuilleumier, & Van De Ville, 2011; Bassett et al., 2008).

Matrices of functional connectivity as an alternative representation are also popular (Achard et al., 2006; Gerhard et al., 2011; Sanz-Arigita et al., 2010) and are occasionally used in the form of small multiples to illustrate trends across different connectivity datasets (Bassett, Brown, Deshpande, Carlson, & Grafton, 2011; Kramer et al., 2011; Ginestet & Simmons, 2011). To support direct comparisons, correlation coefficients from multiple scan states can be shown within nested quadrants of a matrix cell (Thomason et al., 2011), conceptually similar to the designs by Stein et al. (2010) (cf. Section 2.6). However, this design makes it hard to focus on a single scan state.

Representing physical entities, fibertracts that constitute anatomical connectivity are more frequently visualized within the 3D brain volume (Gerhard et al., 2011) (e.g., Figure 5.2(d)). Research in fiber connectivity visualization focuses on their similarity clustering, bundling, and selection in 3D space (Moberts, Vilanova, & van Wijk, 2005; Sherbondy, Akers, Mackenzie, Dougherty, & Wandell, 2005) as well as on illustrative depiction (Everts, Bekker, Roerdink, & Isenberg, 2009). While non-spatial representations for fiber similarity and clustering are also used (Jianu, Demiralp, & Laidlaw, 2009), neuroscientists reported difficulties with interpreting these representations. Related to our work, anatomical connectivity can be reduced to a fiber density graph among ROIs and visualized as a 3D spatial node-link diagram combined with a matrix representation (Hagmann et al., 2008) (e.g., Figure 5.2(e)–(f)).

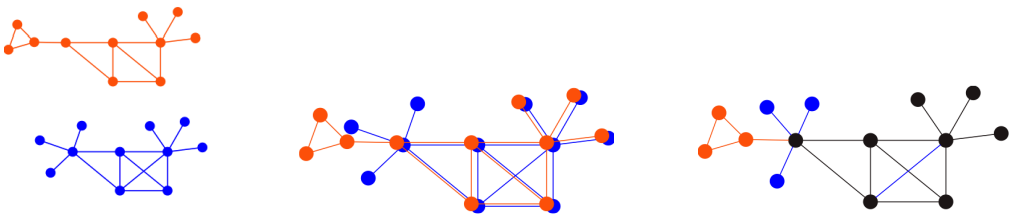
So far, the coordinated visualizations of structural and functional connectivity has received little attention, and existing visualizations focus on spatial representations rather than supporting abstract graph comparison tasks. Several tools, for example, ConnectomeViewer (Gerhard et al., 2011; Li et al., 2012) support visual brain connectivity analysis using spatial 3D node-link and matrix representations for functional connectivity and volumetric fibertract representations for anatomical connectivity. Although it is crucial to communicate the spatial context in brain connectivity analysis for several tasks, 2D non-spatial representations with flexible layouts are more suitable to communicate differences/changes in the connectivity data in an explicit way. To the best of our knowledge, none of the existing brain connectivity tools supports such a visual connectivity comparison.



(a) (b) (c)

(d) (e) (f)

Figure 5.2.: *Visualization techniques for brain connectivity. Left column, functional connectivity: (a) 3D spatial node-link diagram within the brain volume (courtesy of Erik Ziegler, Cyclotron Research Centre, Univ. of Liège, generated with ConnectomeViewer (Gerhard et al., 2011)), (b) 2D node-link diagram with biological layout (Achard et al., 2006), and (c) force-directed node-link accompanied with a spatial visualization showing actual positions of the nodes (Nelson et al., 2010). Right column, anatomical connectivity: (d) fibers within the 3D volume of the brain, (e) fiber density ROI graph as a spatial node-link diagram (Hagmann et al., 2008), and (f) matrix representation of fiber densities between ROIs (Hagmann et al., 2008). All images reproduced with authors' permissions.*



(a) Juxtaposition (b) Superimposition (c) Explicit encoding

Figure 5.3.: *Possible ways of comparing two node link diagrams as illustrated by Gleicher et al. (2011).*

5.2. Designing for Graph Comparison

5.2.1. Design Options

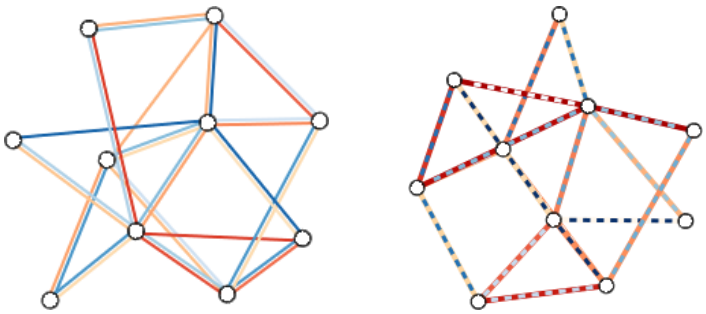
Comparing two graphs first and foremost requires communicating absence or presence of connections. For several domains, comparison techniques for unweighted graphs have been proposed, such as to characterize metabolic pathways (Brandes et al., 2004; Schreiber, 2003), business process models (Andrews et al., 2009) and software evolution (Collberg et al., 2003). Gleicher et al. (2011) categorise comparison techniques into three groups and illustrate them with node-link diagrams (Figure 5.3): (a) juxtaposed views where two graphs are presented side-by-side and often complemented with interactive techniques that highlight the matches between the two (Andrews et al., 2009), (b) superimposed or overlaid views with representations slightly offset, and (c) a single layout for both graphs with differing edges and nodes coded by color (Andrews et al., 2009; Hascoët & Dragicevic, 2012; Erten et al., 2003). According to Gleicher et al. (2011), several of these techniques can be used in conjunction in order to overcome individual weaknesses. To the best of our knowledge, adjacency matrices have not yet been used to compare two weighted or un-weighted graphs.

Only a single study, conducted by Zaman et al. (2011), compared juxtaposed views, direct differencing, animated transitions, and toggling (switch between graphs without animation). The results show that juxtaposition performed worst, while animation was best. Informal feedback that we gathered from our initial implementation, indicates that side-by-side views were indeed much slower and error-prone compared to the overlaid views—due to the fact that the distance to be covered even for simple comparison tasks is much higher in the side-by-side view compared to the overlaid views (especially for element lookup). The difference increases tremendously as the graphs get larger.

Explicit encodings may be the simplest to read since they encode actual increase or decrease in weight *where* they happen. However, the problem with changing edge weights is that we need information about both: *absolute* edge weight for the two graphs, and *relative* difference. A single visual variable in an explicit encoding could only either encode *absolute* or *relative* change, but hardly both.

Edge weight is usually mapped to the length of an edge with inverse proportions (Collberg et al., 2003), or mapped to line stroke width, or is encoded using color and translucency. In matrix visualizations, the weight of a connection is mapped either to the color of the corresponding cell or to the size of a glyph inside the cell (van Ham et al., 2009).

We finally opted for conceptual overlaid views, since both absolute values for edge weight can be shown and users can infer the difference based on visual contrast.



(a) (b)

Figure 5.4.: *Examples of designs for node-link for comparison of weighted graphs.*

5.2.2. Designs for Node-Link Diagrams

The design space for graph comparison, based on node-link diagrams, is limited to visual variables of the link lines: (i) color (intensity or hue), (ii) translucency, (iii) line width, and (iv) stroke patterns. Other variables such as curvature (Henry Riche et al., 2012) are less useful because the continuous variable of edge weight would be hard to read.

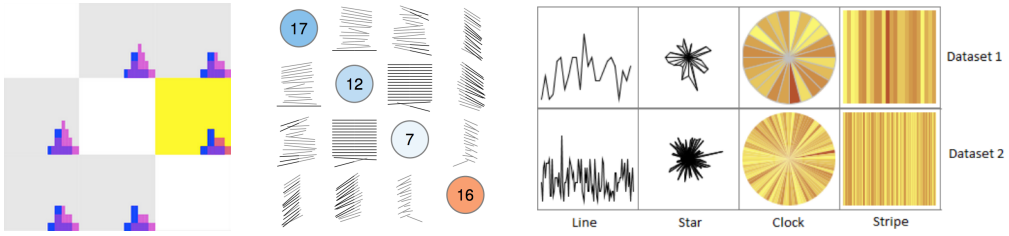
For our designs, we want to keep the visual variables used for this purpose at a minimum. However, the problem is to distinguish between both graphs in the overlaid view. Edge thickness to encode weight increases the space requirements and makes heavy edges very salient, biasing perception. Moreover, when node-link diagrams represent dense graphs, it is important to keep edge thickness at a minimum.

We examined two other options for overlaid edge weight encoding. First, we considered two constant-width parallel edges with two different colors (blue and red)—each encoding the edge weight from one graph (Figure 5.4(a)). Our second option was dashed lines for edges, while color for each graph alternated (Figure 5.4(b)). Initially, we varied dash-length according to edge weight in the corresponding graph, but eventually encoded edge weight by lightness, as in Figure 5.4(a).

We tried both alternatives with our datasets and found continuous parallel lines to be more legible, especially for dense graphs. Dashed lines caused much change of color through the entire visualization, making the overview seem cluttered and edges of one graph difficult to isolate.

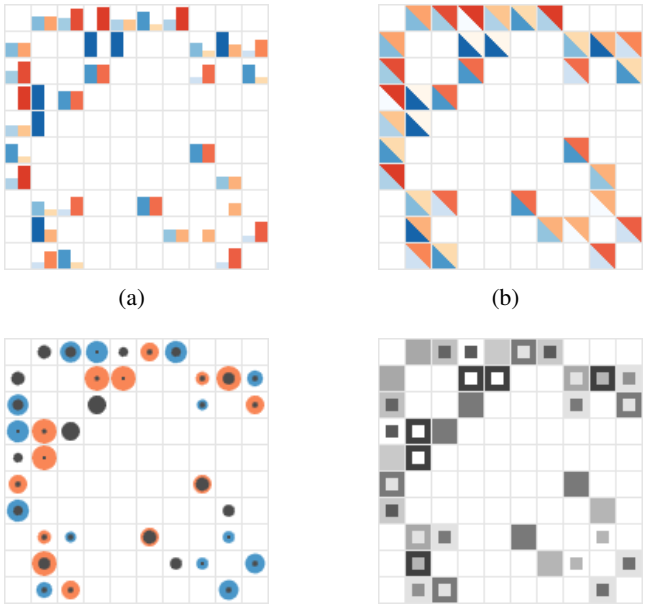
5.2.3. Designs for Matrices

Matrices provide space for much more complex glyphs inside cells and potentially encode information more efficiently. Different cell designs for edge weight comparison across multiple time steps in dynamic networks have been proposed. Yi et al. (2010) use bar charts (Figure 5.5(a)), while Brandes and Nick (2011) use *Gestaltlines* (Figure 5.6(b)) that encode edge weight using angle and length of vertical strokes (each stroke represents



(a) (b) (c)

Figure 5.5.: *Glyph designs to indicate changes in time series: (a) TimeMatrix (Yi et al., 2010), (b) Gestalt Lines (Brandes & Nick, 2011) (c) Time Series as small multiples (Fuchs, Fischer, Mansmann, Bertini, & Isenberg, 2013)*



(a) (b) (c) (d)

Figure 5.6.: *Examples of all visual encodings examined to facilitate the comparison of weighted graphs.*

one time step). Gestaltlines are powerful in showing general trend and outliers, but we consider even two strokes (one for each graph) too hard for comparison.

Similar to Yi et al. (2010), we examined color-coded bar charts shown within the cell to show the absolute weight from each graph (Figure 5.6(a)). However, this design produced vertical patterns and complicates comparison of time steps. We investigated alternative ways of dividing matrix cells into two regions and using a separate color scheme for each region to encode weight from the corresponding graph. Horizontal divisions produce the same problem and we tried diagonal divisions (Figure 5.6(b)) as used in (Bach et al., 2013). However, we found that even with only two graphs, all divisions produced

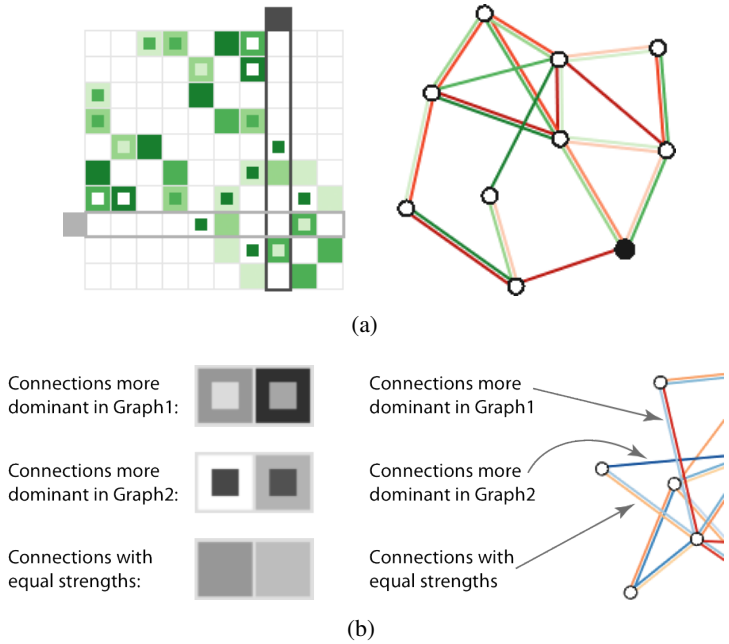


Figure 5.7.: *(a) Final encodings for node-link diagrams and matrices on the same data set. (b) Details of the selected edge weight encoding in matrices (left) and node-link diagrams (right), as appearing the the handout accompanying our study (The study used a red-green color encoding, while we ensured none of the participants was color blind).*

patterns interfering with the cell boundaries of the matrix, making it hard to distinguish individual cells.

Next, we tried to encode weight using scaled glyphs such as concentric circles (Figure 5.6(c)). Here the radius of the inner circle indicates the smaller value of both weights, while the outer circle indicates the larger value. A blue outer circle signifies *increase* in weight—from the first to the second graph—a red circle signifies decrease. We did not pursue this approach further because of the difficulty of identifying a single connectivity state. Besides, when the difference is minimal, the borders produced between inner and outer circles became illegible within the limited cell space.

Finally, we adopted an inner and outer squares division (Figure 5.6(d)), where the edge weight from each graph is mapped to the color density of the corresponding region. We considered this approach efficient because it did not obscure the matrix' grid structure. It also allowed us to reduce the number of visual variables, by using brightness alone to encode weight; graphs are simply differentiated by *inner* and *outer* rectangle. Color hue is free to potentially encode other data attributes. More important, brightness can be difficult to compare across color hues. As a natural outcome of this encoding, the amount of change is mapped to the contrast between inner and outer regions of a cell, easily enabling viewers to differentiate regions with high and low change in edge weight change.

Figure 5.7(a) summarizes the two final designs for node-link diagrams and matrices. Figure 5.7(b) appeared in the hand-out sheet accompanying our user study (see next section), and explaining both encodings to participants.

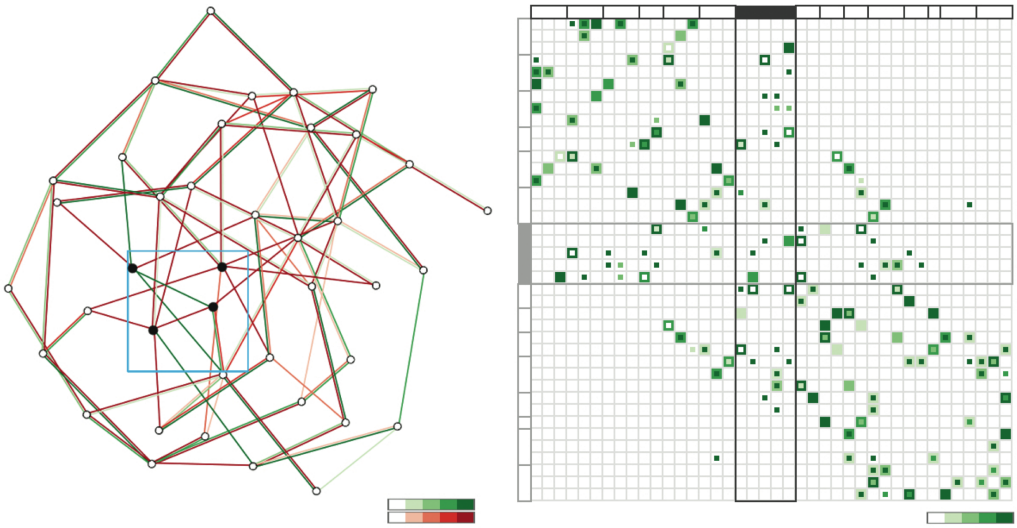
5.3. Controlled User Study

We decided to run a controlled user study to asses the efficiency (*accuracy*, *task completion time*) of our two final designs for node-link diagrams and matrices when comparing two graphs (G_1, G_2) with different edge weights. Our goal was to assess how both techniques scale with changing graph sizes and edge densities across different comparison tasks, to inform designs that would utilize both representations. Representative tasks for the study were derived from common tasks in analyzing differences in brain connectivity, as previously described.

5.3.1. Techniques

We used our visual encodings for overlaying two graphs in node-link and matrix visualizations. In all techniques, the entire information is shown in one full-screen window without a need to zoom or pan. This design eliminates any confounds due to navigation issues rather than reading of the visual representation to complete the task.

1. **Node-Link Diagram—Overlaid** (*NL*): A single node-link diagram shows edge weights in both graphs using two parallel edges. Connections that are present in G_1 are encoded in green colored edges, while connections that are present in G_2 are encoded with the red colored edges (Figure 5.8(a)). Thus, a single green edge represents an edge that was only present in G_1, while a single red edge represents an edge that was only present in G_2. The absolute edge weight is encoded using brightness as in the matrix condition, and we ensured that values for the same edge weight had the same brightness. Layout was a force-directed layout.
2. **Matrix—Overlaid** (*M*): A single adjacency matrix representation shows edge weights in both graphs. Each matrix cell (12×12 pixels) contains an inner region encoding the weight of the edge in G_1 and an outer region encoding the weight of the edge in G_2 (Figure 5.8(b)). The absolute weight is mapped to color brightness using a perceptually linear scaling. A cell with a light inner and dark outer regions indicates an edge with a high weight in G_1 and a low weight in G_2. Vice versa, a cell with a dark inner region and a light outer one indicates an edge with a high weight in G_2 and a low weight in G_1. Since one of our tasks required identifying a specific region, we reordered rows and columns of the matrix to ensure that nodes were placed in the same regions in both the matrix view and the node-link diagram. The tradeoff of this reordering is, however, that it did not ensure a close placement



(a) (b)

Figure 5.8.: *Example from our controlled study: (a) node-link and (b) matrix representation of the same data (small, dense) for the region identification.*

of items that need to be compared during other tasks, as other reordering algorithms could have.

5.3.2. Tasks

We identified three generic comparison tasks, which can be related to the taxonomy in Chapter 3. In fact, all tasks were compound tasks and the temporal WHEN dimension referred to these graphs. These tasks require users to assess changes in edge weights at different levels of detail: from a single element to the overview of a large portion of the data. Below, we describe each task and the optimal strategy for achieving it.

Assess weight change of a node's connections (*Trend*): *Given one highlighted node, does the overall edge weight to all of its neighbors* ***decrease*** *or* ***increase*** *from G_1 to G_2?* Participants completed this task in three steps: (1) they needed to identify all connections to the highlighted node (WHERE), (2) they needed to assess the change in weight for each connection (WHAT), and (3) they needed to estimate the aggregated change for all these connections. To avoid confounds in the study, we made sure that all trials exhibited a clear increase or decrease trend. To prevent participants from selecting one option at random, we offered an additional "I don't know" option and instructed them to select it if their confidence was low. We excluded these trials from the analysis.

Assess connectivity of common neighbors (*Connectivity*): *Given two highlighted nodes,* ***how many*** *of their common neighbors in G_1 are still common neighbors in G_2?* Participants completed this task in two steps: (1) they needed to find common neighbors, meaning the nodes that are connected to both of the highlighted nodes (WHERE) and

(2), among them, they needed to count how many are present in both graphs (WHAT). Participants selected an option from 0 to 6.

Identify the region with most changes (*Region***)**: *Identify the region showing the* ***most variation*** *between G_2 and G_1?* For this task we provided users with simple interaction tools to view regions. In the node-link case, we create a 4×4 grid laid over the node-link diagram and assigned each node to the region it fell into. As participants moved their mouse pointer on the diagram, the corresponding region boundary appeared in light blue and nodes of the region became highlighted (Figure 5.8(a)). In the matrix condition, we used the same 16 regions, ordering the nodes linearly according to the regions they belonged to. As participants moved their mouse pointer on the matrix, the region boundaries were highlighted (Figure 5.8(b)).

Participants completed this task by browsing each region successively (WHERE), estimating the region with highest edge weight variation (WHAT). To avoid confounds in the study, we ensured that all trials presented a region with discernably higher variation than the rest. Participants clicked on a region of their choice.

5.3.3. Data

We used synthetic data in order to control for data size, network density, and amount of change. We generated four types of uniform networks with either 40 (*Small*) or 80 (*Large*) number of nodes, and with either 5% (*Sparse*) or 10% (*Dense*) edge density. For each data type, we created four isomorphic networks per trial which were used across all tasks. We created five additional datasets for training with a *Small*, *Sparse* metric. The edge weights were assigned arbitrarily to each of the generated graphs, ranging from 0 to 1 in increments of 0.2. We created comparison graphs by copying the original weighted graphs and then randomly perturbing the weights of 70% of the edges in order to ensure edge weights that remain constant across the graphs.

5.3.4. Participants and Setting

11 participants (1 female) participated in the study with a mean age of 30.2 years. All participants had normal or corrected-to-normal vision, without color deficiency. Participants were graduate students or researchers, familiar with graphs, but not with matrices. The experiment was conducted in a quiet room during the day. The computer used for the study was a 2.4 GHz Dual-Core HP Z800 workstation equipped with a 30 inch screen with a 2500×1600 pixel resolution. The visualization area was restricted to a window of 1500×1350 pixels. Participants interacted with a mouse and keyboard to complete the tasks.

5.3.5. Experimental Procedure

We used a within-subject, full-factorial design: 2 *Techniques* × 3 *Tasks* × 2 *Sizes* × 2 *Densities*. Each condition was repeated four times. We counter-balanced the techniques (*M*, *NL*) using a Latin square. Tasks appeared always in a fixed order of increasing complexity (*Trend*, *Connectivity*, and *Region*). Datasets also appeared in a fixed order from simple (*Small* and *Sparse*) to more complex (*Large* and *Dense*). For each trial, we measured *accuracy* and *task completion time*.

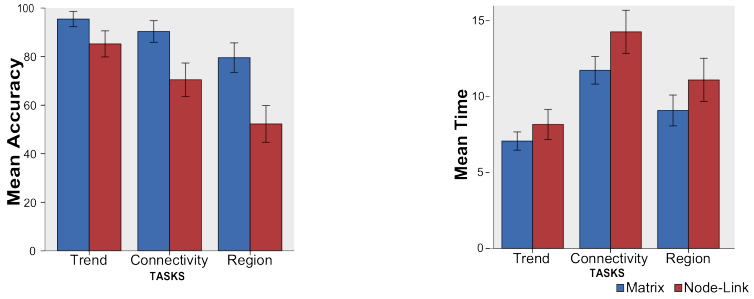
Before the controlled experiment, we instructed participants about the visualizations as well as the weight encoding used in each, making sure none of them had any vision problems. We asked them to complete trials as accurately and quickly as possible. Before each new technique and task, five training trials were performed. Participants completed the first two following the explanations of the instructor. They answered the remaining trials on their own unless they had further questions. Trials were not timed during the training. After training trials, participants completed the 16 timed trials (2 Size × 2 Density × 4 Repeat) required for each condition (technique × task). For all conditions, we collected a total of 96 trials per participant (excluding training).

To keep the experiment within a reasonable time, we limited the time per trial to 30 seconds (measured as a feasible value in a pilot study) and notified participants before the experiment. After 20 seconds, the screen flashed and a time counter appeared below the visualization for the remaining 10 seconds. To provide their answer, participants pressed the space bar to view the dialog box with the answers. After this point, the timer stopped and the visualization disappeared. Participants were instructed to take breaks if needed when no visualization was shown on the display. None of the participants took a noticable break nor reported any fatigue.

5.3.6. Hypotheses

Our hypotheses for the experiment were the following:

- **H1**—For the *Trend* task, we expect Matrix to outperform (*accuracy* and *completion time*) Node-Link for high-density datasets. We expect that occlusion problems in Node-Link would get severe in *Dense* cases, causing more errors. We also believe that the spatial encoding used in Matrix would prove easier to remember than the color encoding used in Node-Link, leading to faster answers.
- **H2**—For the *Connectivity* task, we expect Matrix not to outperform Node-Link because we believe that participants might need to compare two distant columns or rows in Matrix, whereas the force-directed layout in Node-Link ensures that connected nodes are in closer proximity.



(a) Mean accuracy in %

(b) Mean completion time in seconds

Figure 5.9.: *Task error and task completion time per task for matrix (blue) and node-link (red) techniques. Error bars represent +/– 2 standard errors.*

- **H3**—For the *Region* task, we expected Matrix to outperform Node-Link (*accuracy* and *completion time*). In Matrix, all connections of nodes in one region are contained within the region boundary. However, in Node-Link, participants had to consider links that are drawn across region boundaries, leading to more errors and slower answers.
- **H4**—Overall, we expected Node-Link to decrease in performance (*accuracy* and *completion time*) for *Dense* datasets, as edge density causes many edge crossings, decreasing the legibility of this representation.

5.3.7. Results

We used a repeated-measure analysis of variance (RM-ANOVA) to analyze the collected accuracy and time performance data. We performed the RM-ANOVA on the logarithm of the task times to normalize the skewed distribution, as is standard practice with reaction time data. The analysis of the time performance is reported for correct answers only.

Accuracy The accuracy results are summarized on the left side in Table 5.1 and are shown in Figure 5.9(a). We found a significant effect of accuracy for *Technique*

	Accuracy			**Time**		
	Matrix	Node-Link	p-value	Matrix	Node-Link	p-value
All tasks*	**88.5** (0.9)	69.3 (2.0)	$< .001$	**9.3** (0.3)	11.0 (0.4)	$< .001$
Trend *	**95.5** (1.2)	85.2 (4.3)	$< .05$	7.1 (0.3)	8.2 (0.5)	not significant
Connectivity *	**90.3** (1.0)	70.5 (2.5)	$< .0001$	**11.7** (0.5)	14.3 (0.7)	$< .0001$
Region *	**79.6** (2.1)	52.3 (4.5)	$< .0001$	**9.1** (0.5)	11.1 (0.7)	$< .01$

Table 5.1.: *Means of accuracy and time in percentages. Standard error is indicated in parentheses. Significant differences are indicated by *. More accurate results are highlighted in bold.*

($F_{(1,10)} = 55.32, p < .0001$) with a large effect size ($\eta_p^2 = .85$). Overall, Matrix is about 20% more accurate than Node-Link. The RM-ANOVA also revealed a significant effect of *Task* ($F_{(2,20)} = 28.67, p < .0001$) with large effect size ($\eta_p^2 = .74$), and a significant effect of the interaction *Task* × *Technique* ($F_{(2,20)} = 4.34, p < .05$) with a large effect size ($\eta_p = .30$). Pairwise comparisons revealed that Matrix is more accurate than Node-Link for all three tasks.

We also found a significant effect of *Size* ($F_{(1,10)} = 18.69, p < .01$) with a large effect size ($\eta_p^2 = .65$) and *Density* ($F_{(1,10)} = 61.00, p < .0001$) with a large effect size ($\eta_p^2 = .86$). As expected, accuracy decreases for *Large* or *Dense* networks. The interaction *Technique* × *Size* ($F_{(1,10)} = 6.5, p < .05$) is significant with a large effect size ($\eta_p^2 = .39$). Node-Link is affected by *Size*, losing about 25% accuracy in large datasets compared to small ones. In contrast, Matrix has less than 1% loss in accuracy. Pairwise comparisons indicate that Matrix significantly outperforms Node-Link for large networks across all three tasks. The results indicate that both *Techniques* are affected by *Density*, Matrix losing about 10% accuracy, Node-Link about 20%. Pairwise comparisons indicate that Matrix significantly outperforms Node-Link both for *Sparse* and *Dense* networks across all three tasks.

Completion Time We analyzed the completion time for correct answers only, using a mixed linear model capable of handling missing data cases. We excluded about 10% incorrect trials for Matrix and 30% for Node-Link (out of 528 total trials per technique).

The completion time results are summarized on the right side in Table 5.1 and are shown in Figure 5.9(b). We found a significant effect of time for *Technique* ($F_{(1,10)} = 28.40, p < .0001$). Overall, Matrix is 15% faster than Node-Link. We also found a significant effect of *Task* ($F_{(2,20)} = 183.25, p < .0001$) and *Technique* × *Task* ($F_{(2,20)} = 3.82, p < .05$). Pairwise comparisons reveal that Matrix is faster than Node-Link for *Connectivity* and *Region* tasks.

We also found a significant effect of *Size* ($F_{(1,10)} = 139.19, p < .0001$) and *Density* ($F_{(1,10)} = 30.94, p < .0001$). As expected, completion time increases for *Large* or *Dense* networks. The interaction *Technique* × *Size* ($F_{(1,10)} = 9.84, p < .01$) is significant. Node-Link is particularly affected by *Size*. For *Large* datasets, the completion time increases by about 60% for Node-Link and 40% for Matrix. For *Dense* datasets, the completion time increases by about 25% for both techniques.

User Preference Users rated their preference on a 5-point scale from -2 (strong preference for Node-Link) to +2 (strong preference for Matrix). We analyzed these ratings using a z-test. Results reported in Table 5.2 reveal that there is a significant difference in user preference between *Techniques*. Users preferred Matrix overall ($p < .0001$). In fact, 7 out of 11 participants ranked Matrix as the most effective representation (maximum rating of 2). Z-test also showed that the user preference was significantly different for the *Connectivity* ($p < .0001$) and the *Region* tasks ($p < .0001$). For these tasks, Matrix was preferred to Node-Link.

	Preference	p-value
Overall*	**1.25** (0.26)	$< .0001$
Trend	0.63 (0.38)	not significant
Connectivity *	**1.31** (0.23)	$< .0001$
Region *	**1.36** (0.21)	$< .0001$

Table 5.2.: *User preference (means of ratings from -2-strongly node-link, -1-somewhat node-link, 0-indifferent, 1-somewhat matrix, 2-strongly matrix), the standard error is indicated in parentheses. Significant differences in user preference are indicated by *.*

5.3.8. Discussion

The results of our controlled experiment indicate that matrix representations are more effective than node-link diagrams for encoding edge weights and performing comparison tasks. While we expected that it would be the case for the *Trend* (H1) and *Region* (H3) tasks in the *Dense* datasets, we were surprised to find significant differences in accuracy across all tasks for both sparse and dense networks. We did not expect a significant performance difference across techniques for *Connectivity* (H2). While we did not find any significant difference in completion time for correct answers, the results indicated that matrix outperformed node-link in accuracy, contradicting (H2).

Our results are in accordance with the study by Ghoniem et al. (2005). Besides that we compared particular *designs* for graph comparison on node-link diagrams and matrices, our tasks involved multiple graph elements; spotting a particular region in the graph, requires to entirely observe the representation and spot higher level visual patterns. However, we did not test for path finding tasks, since such tasks are generally hard to perform with adjacency matrices (Ghoniem et al., 2005).

Our results further show that participants unfamiliar with matrices were able to understand our cell encodings and to map abstract tasks to visual tasks (see Section 3.6). We originally thought that comparison in matrices may be error-prone since the rows or columns to be compared may be far away. Instead, we observed that the linear arrangement of edges (cells) in the matrix allowed people to inspect each of the candidate neighbors successively. In contrast, despite common neighbors being placed closer together in space in the node-link diagram, performance suffered from the less systematic manner to count them, leading to a decrease in accuracy.

While we exptected the decrease in performance for *dense* node-link diagrams (H4), we did not expect the strong performance decrease for large networks with node-link diagrams. We believe that this happened because the total number of edges increases quadratically with a linear increase in number of nodes, although the graph *density* remains the same. The total number of links shown in large sparse datasets, thus, is much higher than small sparse datasets. To offer a similar visual complexity in small and large graphs, we could envision to control the edge density per display area unit.

5.4. Conclusions

The findings of the study indicate that, for edge weight comparisons across two networks, node-link representations are more error-prone and less readable than matrices. The implications are three-fold.

Brain Analysis Scientists should adopt matrix representations to ensure better accuracy when performing comparison tasks. Our findings are encouraging to develop proper solutions that can combine matrices with spatial maps of the brain, which is important for brain analysis.

Network Density The first point to consider relates to the datasets' density. While we tested graphs with connection densities of 5% and 10%, most brain networks are fully connected graphs with a wide range of weights. For certain tasks, especially overview and topological exploration tasks (WHERE, WHAT), it is necessary to observe the full range of edges; the general distribution of weight and its differences, as well as clusters and regions of lower edge weight. Brain analysis currently use a visibility threshold on the edge weight, in order to make node-link diagrams readable. Yet, as Ghoniem et al. (2005) and our study suggest, the reasonable threshold for node-link diagrams is very high, i.e. few edges can remain visible.

Dynamic Networks Comparing graphs can help exploring dynamic networks by comparing individual time steps. The matrix design can be integrated into GraphDiaries to show directed differences between time steps in the network view. It remains to be investigated how many graphs can be directly compared with our glyph design.

***

Acknowledgements *We would like to thank Thomas Grabowski, Jr., MD, Dr. Tara Madhyashta, Dr. Kayo Inoue from The Integrated Brain Imaging Center (IBIC) at University of Washington, and to Dr. Arjun Bansal from Harvard Medical School for their valuable input that went into this work.*

This project has been started by my coauthors Basak Alper (University of California, Santa Barbara, CA), Nathalie Henry-Riche (Microsoft Research, Seattle, WA), Tobias Isenberg (INRIA, France), and Jean-Daniel Fekete (INRIA, France). I joined the project after the task analysis for brain analysis (Section 5.1). With Basak, I explored and developed the designs for both visualizations, as well as designed and conducted the controlled user study. I further was responsible for creating and adjusted the synthesized data (see Section A.5). This chapter slightly extends the original publication in (i) discussion of related work on graph comparision, (ii) the description of our designs, and (iii) the final conclusions.

6. Unfolding Networks with the Matrix Cube

Contents

DENSE dynamic networks with changing edge weight, such as brain connectivity networks, pose numerous challenges to visualization interfaces. The last chapter proposed *one* technique for *one* set of tasks (direct comparision of two time steps). Eventually, many different visualization and interaction techniques may be required to support the wide range of tasks and account for a network's complexity. For example, matrices are used to effectively visualize dense topologies; glyphs inside matrix cells (e.g. Brandes & Nick, 2011) or timelines can show temporal evolution of edge attributes; difference views allow for direct comparison, while animations help to track changes over a longer period, and allow a user to quickly jump between individual views of time steps.

Multiple techniques can always be combined and integrated into an interface in order to improve its power and task coverage, but at the cost of simplicity and usability. Highly specialized interfaces with powerful functions require learning and may only be manageable by experts, who have learned to *read* the visualizations and *use* different techniques, as well as aggregate information across visualizations. Generally speaking, the more powerful the interface becomes, the more effort is required by users to manage it—or, the less willing they are to learn and use it. Yet, we must build interfaces that remain simple while supporting a wide variety of tasks.

In this chapter, we address the following two research questions:

- ▷ *How to manage the tradeoff between power and simplicity in network visualization interfaces?*
- ▷ *How to support exploration of dense dynamic networks with changing edge attributes?*

To address these questions, we developed the *Matrix Cube*, an easy-to-understand visualization and interaction model to visualize and explore dynamic networks. Matrix Cubes result from stacking adjacency matrices for each time step in the network, forming a space-time cube (Hägerstrand, 1970; Kraak, 2003). 3D visualizations suffer from numerous problems, including perspective distortion, occlusion, and the need for specific interaction techniques (cf. Shneiderman, 2003; Robertson, Card, & Mackinlay, 1993). However, Matrix Cubes are not 3D visualizations; they serve as a model, a metaphor, and a design space to create and relate better readable 2D visualizations (*views*). Each view supports certain tasks by either showing the cube in a particular way, or by showing only particular parts of it. The views are conceptually—and visually—related by the cube, making it easy to navigate between views and understand *which* information they show and *how* they show it.

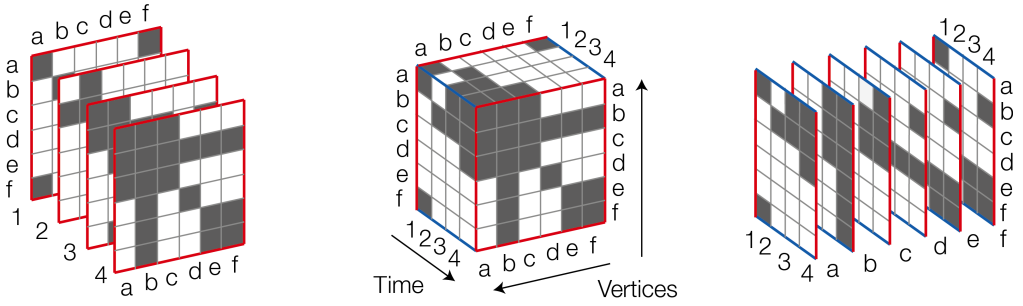
In order to create 2D visualizations from the cube, users interact with it. The Matrix Cube and associated interactions are inspired by the way people comprehend and manipulate physical cubes. Users can change their perspective on the data by rotating and projecting the 3D cube, or decomposing it using operations such as slicing and filtering. These operations lead to various 2D visualizations, that offer a set of coherent, easy-to-relate views on the data. The 3D view of the cube provides an overview of the data, as well as a visual metaphor for understanding transitions and pivoting between views. Meanwhile, the 2D views derived from the cube support more detailed analysis of specific aspects of the network and entities that constitute it.

In order to allow for interactions on the cube and employ visual mappings, we implemented a system called *Cubix*. Cubix supports several predefined views, quick navigation between them and several visual mappings, making it easy to handle the cube and explore the data. Cubix further features animated transitions that show how the cube is rotated and decomposed.

6.1. The Matrix Cube

The *Matrix Cube* is a representation of dynamic networks with associated time series to edges (e.g. edge weight). Let $G = (N, E, T, W)$ be a dynamic network with a set of nodes N, a set of edges E between nodes in N, and a set of time steps T. W is a function that maps a weight to every edge, for every time.

The Matrix Cube is created by stacking adjacency matrices in chronological order, one for each time step, as illustrated in Figures 6.1(a) and 6.1(b). The cube consists of two dimensions corresponding to the network nodes (*node dimension*) and one dimension containing the network's time steps (*time dimension*). In our representation, node dimensions are colored red, while the time dimension is blue. For the sake of simplicity and demonstration, here we focus on undirected networks, where matrices are symmetric to



(a) Adjacency Matrices (b) Matrix Cube (c) Node Slices

Figure 6.1.: *The Matrix Cube. (a) Each time step of the network (1,2,3,4), is represented as an adjacency matrix. (b) The Matrix Cube results from stacking those matrices. Red edges of the cube hold nodes and correspond to the rows and columns of the constituent adjacency matrices; blue edges of the cube hold time steps. (c) Slicing the cube along one of the node dimensions yields node slices.*

the diagonal. An extension towards directed networks is straightforward and is briefly discussed in the end of this section.

6.1.1. Parts of the Cube

This section further describes the three major parts of a Matrix Cube and what information they encode: *cells*, *slices* and *vectors*.

Cells A *cell* $c_{ijt} \in (N \times N \times T)$ in the Matrix Cube exists for each edge e_{ijt} between nodes i and j at time t.

Slices A *slice* in the cube is a 2D array of cells. There are two types of orthogonal slices in the Matrix Cube:

- **Time slices** T_t exist along the cube's time dimension and correspond to adjacency matrices for any time step t (Figure 6.1(a)). Time slices have the dimensions $N \times N$.
- **Node Slices** N_i exist along each of the cube's node dimensions and represent the dynamic ego network of node i. Node slices have the dimensions $N \times T$ (Figure 6.1(c)). The cube contains one node slice per node in the network. Rows in a node slice correspond to all nodes in N while columns correspond to time steps in T. A cell in the node slice for node i shows all edges between i and the nodes in the slice's rows, for any time.

Vectors Cells are 0-dimensional and slices are 2D structures in the cube. 1-dimensional structures are called *vectors* and contain a 1-dimensional array of cells. The Matrix Cube contains two types of vectors which are orthogonal:

- **Time vectors** contain all cells representing an edge between the same node pair over time; $\alpha_{ij} = [c_{ij0}, c_{ij1}, ...c_{ijT}]$. A time vector hence describes the evolution of the connectivity between two nodes. Empty places in a time vector indicate that an edge is not present at that moment.
- **Neighborhood vectors** contain all the cells representing incident edges of a node; $\beta_{it} = [c_{i0t}, c_{i1t}, ...c_{iNt}]$ A neighborhood vector β_{it} corresponds to a row or a column in the adjacency matrix (time slice) for time t. It contains all incident edges for a node i at a time t.

6.1.2. Matrix Reordering

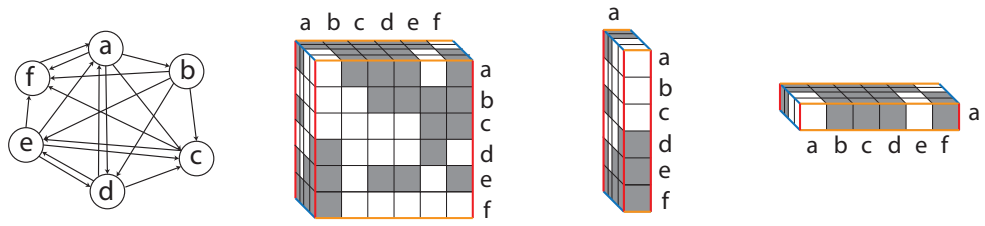
General methods for matrix reordering have been mentioned in Section 2.4.1. Since the Matrix Cube is a static structure it avoids the problem of stabilizing the mental map for each time point. If individual time slices were to be reordered, consistency of time vectors and node slices would be violated. Similar to the global layout in dynamic node-link diagrams, the Matrix Cube has one global ordering, which is the same for all time slices.

Brandes and Nick (2011) use a block modeling approach (Doreian, Batagelj, & Ferligoj, 2005) to reorder a matrix that visualizes a directed weighted dynamic network (Figure 2.28(c), Section 2.6). Block modeling requires a weight w_{ij} for any pair of nodes i and j. In the case of a directed networks, Brandes and Nick calculate w_{ij} as the sum over the individual weights w^t_{ij} (weight w from node i to node j in time $t \in T$, with $w^t_{ij} > 0$):

$$w_{ij} = \sum_{t=1}^{T} \sqrt{w^t_{ij} \cdot w^t_{ji}}$$

Cubix currently supports the following reordering methods:

1. **alphanumerical** on the node names to help finding nodes,
2. **reverse Cuthill-McKee** (Cuthill & McKee, 1969): an algorithm that tries to minimize the matrix bandwidth and which yields good results for sparse matrices. It starts with a low-connected node and subsequently choses neighbors with the highest number of common neighbors.
3. **Traveling salesman resolver**: models the linear matrix ordering as finding the shortest path between points with a defined distance. Distances between nodes are the euclidean distances between row vectors. To find the shortest path, we use the implementation of Lin-Kernighan heuristic (*LKH*, 2012), described by Helsgaun, 2000).
4. **Optimal leaf ordering** first requires a hierarchical clustering that results in a linear order of the leaves (nodes). Then leaves in the clustering are ordered to minimize the distance d_{ij} between any two adjacent nodes. Clustering in Cubix is calculated using the HAC java implementation (*HAC - A Java class library for hierarchical agglomerative clustering*, 2013). First, it calculates a binary clustering, based on



(a) Node-link diagram (b) Matrix Cube (c) In-slice (d) Out-slice

Figure 6.2.: *Illustration of (b) Matrix Cube for directed networks (a), in row-major notation (edges point from rows to column nodes). (c) Vertical node slices contain incoming edges (In-slices), (d) horizontal node slices contain outgoing edges (Out-slices)*

the euclidean distances between nodes (rows/columns). Then, the clusters are ordered to minimize the distance between adjacent leaves in the clusters. This optimization is done with the Bar-Joseph algorithm for ordering binary hierarchical clusters (Bar-Joseph et al., 2003).

Since we do not yet visualize directed networks with the Matrix Cube, for the distance w_{ij} between two nodes i and j we simply accumulate the absolute values of their weights over time:

$$w_{ij} = \frac{1}{|T|} \cdot \sum_{t=1}^{T} w_{ij}^{t}$$

The values of w_{ij} are then useds to calculate the euclidean distance d_{ij}:

$$d_{ij} = \sqrt{\sum_{n=1}^{N} (w_{in} - w_{jn})^2}$$

6.1.3. Directed Networks

For directed networks that allow only one edge between two nodes i and j, that edge can appear twice in each time slice; as cell c_{ijt} and as cell c_{jit}, while edge direction must be encoded visually (e.g. color). For such networks, no extensions are required to the previous definitions of the Matrix Cube. Section 6.6.3 illustrates such a network.

For directed networks that allow for bi-directed edges, our definitions can be easily extended. Since an edge can only appear as one cell in a time slice, matrices can be read from rows to columns, for example (row-major notation). Consequently, we would differentiate between horizontal and vertical node slices in the cube. Vertical slices contain all incoming edges and horizontal slicess contain all outgoing edges. There would

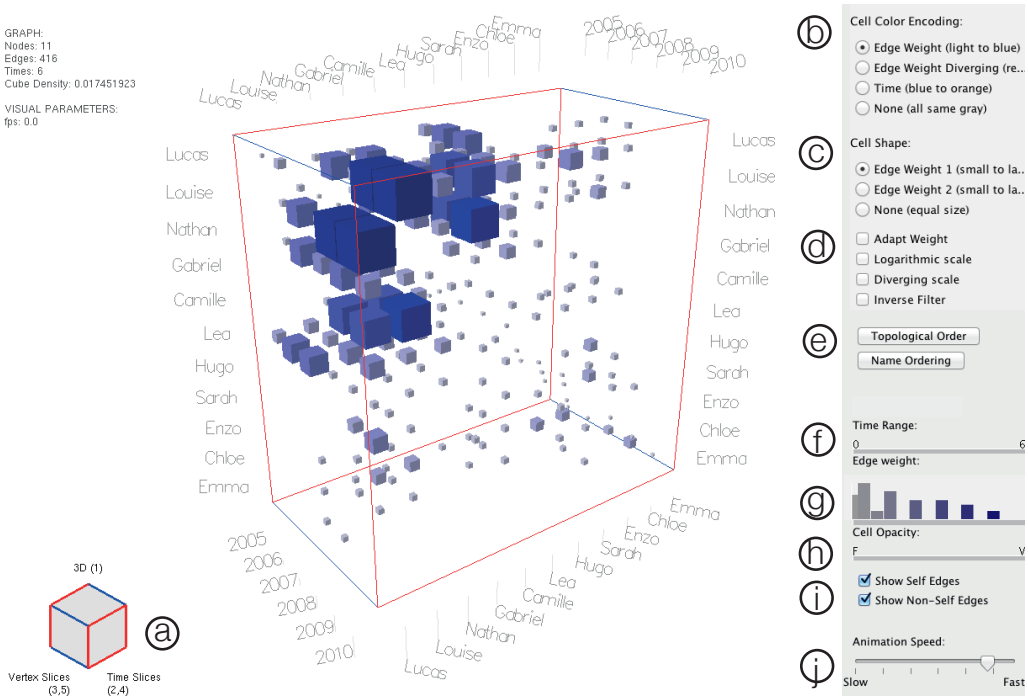


Figure 6.3.: *Cubix UI screenshot. a) Cubelet Widget, b) Cell color encoding options, c) Cell shape encoding options, d) Additional cell options, e) Matrix ordering options, f) Time range slider, g) Cell weight filter with histogram indicating edge weight distribution, h) Cell opacity slider for filtered (F) and visible (V) cells. i) General edge visibility options, j) transition speed slider.*

further be two time vectors between any node pair (one for each direction) and two types of neighborhood vectors; one comprising incoming edges at a specific time, and one comprising outgoing edges at a specific time. These additional slices and vectors describe additional views in our design space.

6.2. Cubix

The Matrix Cube is a rich model to describe dynamic networks. Cubix is a system visualizing dynamic networks with the Matrix Cube. Its interface is shown in Figure 6.3, with the Matrix Cube in the center. Users interactively manipulate the cube by applying *operations* to it. Different operations can be described to (i) derive meaningful 2D views that emphasize particular dimensions of the data, (ii) filter cells, and (iii) apply different visual mappings to cells' color and size. This section summarizes the main concepts of Cubix, while a walkthrough follows in the next section (Section 6.3), illustrating how Cubix supports the exploration of dynamic networks.

6.2.1. Design Principles

Cubix is based on the following design principles:

D1: **3D as a pivot** — As already stated in the introduction of this chapter, 3D representations are hard to visualize on two-dimensional screens. Cubix is about applying *operations* on the cube that result in 2D visualizations. Still, explicitly showing the 3D cube has advantages: (i) it explicitly conveys the cube-based model to the user, (ii) it provides an overview of the data along all dimensions, and (iii) it serves as a pivot visualization when switching between views.

D2: **Limited number of views** — The possible number of operations on the Matrix Cube as well as the views that can be derived by decomposing the cube and arranging its parts is potentially huge. In Cubix, the number of operations and views is constrained in order to keep navigation simple and manageable.

D3: **Easy navigation** — Some views require multiple operations. For instance, laying out slices side-by-side requires specifying the slicing direction (time slices or node slices), and moving them to their appropriate position in 3D space. With the *Cubelet* ⓐ[1], we provide a widget for quick navigation between most predefined views, which are also accessible via keyboard shortcuts.

D4: **Animated transitions** — Changes between views on the cube are smoothly animated, providing a good level of perceptual continuity during transitions, which helps users understand the relationship between views as well as track elements across them.

6.2.2. Views and Operations

In order to support exploration and create 2D views, Cubix currently supports several *operations*, illustrated in Figure 6.4. Most of them lead to particular *views* on the data, others are used to adjust the cube's appearance. Operations are performed on the cube or its slices and change the geometry of the cube. Operations are grouped into six major categories:

1. **Extraction** includes the extraction of individual cells, time and neighborhood vectors as well as both types of slices; time and node slice.
2. **Projection** flattens the cube along one dimension; time flattening or node flattening. Vectors in each direction get flattened into a single visible cell.
3. **Shift** includes the translation of time and node slices along all three spatial dimensions.
4. **Rotation** includes the rotation of the cube along all three spatial dimensions and the rotation of individual slices along the vertical spatial axis.
5. **Filtering** is the filtering of cells/edges according to their weight.

[1] Circled letters refer to interface components in Figure 6.3

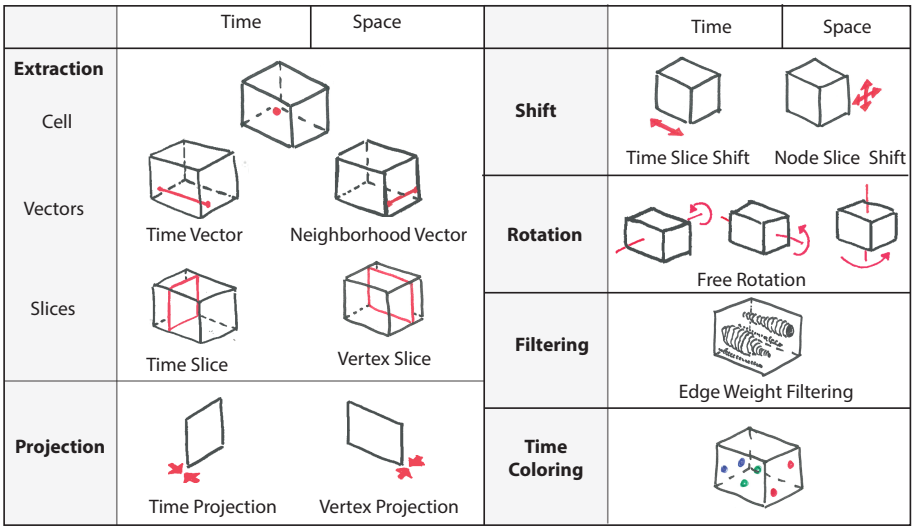


Figure 6.4.: *Operations on the Matrix Cube, supported in Cubix.*

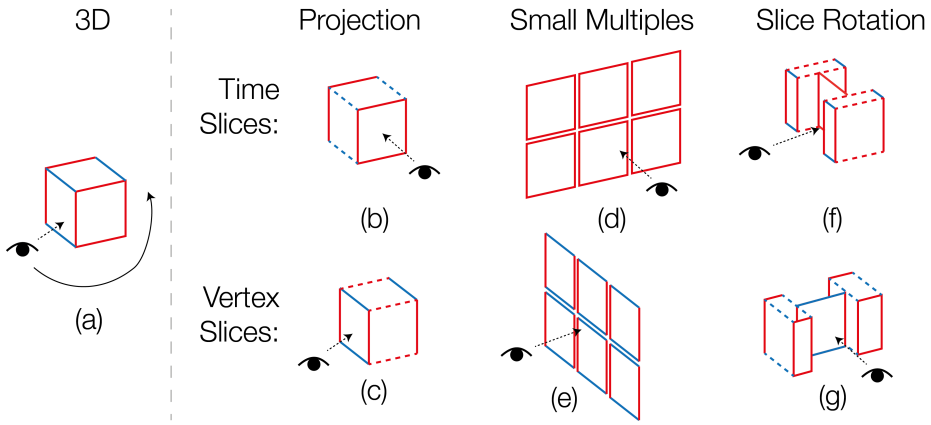


Figure 6.5.: *Cubix View design space. Columns indicate operations applied to the cube. Rows indicate operations applied to time (red × red) and node slices (blue × red), respectively. (a) 3D view, (b) time-projection view, (c) node-projection view, (d) time small multiples, (e) node small multiples, (f) time-slice-rotation, and (g) node-slice-rotation.*

6. **Time Coloring** colors cells according to where they are located in time. All cells of the same time slice are get the same color assigned.

Some of these operations are used to create the *views* (D2) in Cubix, others are applied interactively by the user. Figure 6.5 shows the set of, each of which can be further adjusted. View names have been chosen to consistently to describe a *view design space*.

(a) **3D view:** The Matrix Cube is shown in 3D space, using a perspective projection. The virtual camera can be everywhere around the cube (Figure 6.5(a)). The 3D view is the default view and does not require any particular operation. It can be

(a) (b) (c)

Figure 6.6.: *Different states of the Cubelet widget indicating the current view of the Matrix Cube. (a) The $N \times N$ face is shaded and selected slices highlighted, (b) slicing along the time dimension with selected slice highlighted, and (c) slicing along the node dimension. Red slices indicate slices the user has selected, while the others are hidden.*

adjusted by *rotation*, *filtering*, *slice*, *vector* and *cell extraction*. Extraction and filter operations are not exclusive but complement each other.

(b) **Time-projection view:** The virtual camera orthogonally faces a cube's side so that the entire cube is *projected* onto the $N \times N$ plane (Figure 6.5(b)). Using orthogonal projection, temporal information is generally lost to the benefit of showing clearly the network topology.

(c) **Node-projection view:** The virtual camera orthogonally faces a cube's side so that the entire cube is *projected* onto the $N \times T$ plane (Figure 6.5(c)). Using orthogonal projection, topological information is generally lost to the benefit of giving a glimpse of the overall temporal evolution of the network.

(d) **Time small multiples:** This view *extracts* all time slices and *shifts* them so that they are placed side by side on the screen (Figure 6.5(d)). This view shows the network's state at any point in time (in T).

(e) **Node small multiples:** This view *extracts* all node slices and *shifts* them so that they are placed side by side on the screen (Figure 6.5(e)). It shows the evolution of every node over time.

(f) **Time-slice-rotation:** This view is a focus+context view that *rotates* one or more time slices in the context of the node projection view (Figure 6.5(e)).

(g) **Node-slice-rotation:** This view is a focus+context view that *rotates* one ore more node slices in the context of the time projection view (Figure 6.5(f)).

6.2.3. View Navigation with the Cubelet Widget

To obtain a view, users do not need to perform all the necessary operations manually (D3). Quick navigation is enabled by an interactive overview widget called the *Cubelet* ⓐ (Figure 6.6). The Cubelet is a static 2D-isometric abstraction of the Matrix Cube that indicates the current view on the Matrix Cube. Our Cubelet is somewhat similar to the ViewCube (Khan, Mordatch, Fitzmaurice, Matejka, & Kurtenbach, 2008) but provides different feedback about the current views. It uses the same color coding for the cube's dimensions as the actual Matrix Cube: blue for time, red for nodes. The Cubelet's left face

corresponds to node slices (red × blue), the right face to time slices (red × red). Clicking on one of these faces triggers an animated transitions to the corresponding projection view, and shades the clicked face of the Cubelet (cf. Figure 6.6(a)). Dragging the mouse on the cube's faces transitions to the corresponding small-multiples view. When this happens, the Cubelet changes its appearance, to make it look like if it were sliced. Clicking on the Cubelet's top face animates back to the 3D pivot (Figures 6.6(b) and 6.6(c)).

Red segments in the cube indicate slices that have been selected manually by the user, while all other slices are invisible.

6.3. Walkthrough

This section illustrates Cubix using the example of a scientific collaboration network, referring to the tasks in the task taxonomy in Chapter 3. The network contains one time step per year. In this network, nodes correspond to researchers, and connections between nodes indicate a co-authorship relation on at least one publication during that year. Self-edges, which are cells located on the cube's diagonal, indicate individual publications of the corresponding author. Edge weight represents the number of papers between any two authors for a given year.

6.3.1. Matrix Cube Overview

After loading the data into Cubix, the cube appears as in Figure 6.3. Its axes are labeled with author names and dates, respectively. The initial questions we can answer from this first overview include all three dimensions of our taxonomy in Chapter 3 (WHEN, WHERE WHAT):

Q1) **Size**: How big is the network, i.e. how many nodes and time steps? (WHERE)
Q2) **Density:** How dense is the network approximately? (WHERE, WHEN)
Q3) **Clusters:** How are edges distributed in time and topology? (WHERE, WHEN)
Q4) **Attribute distribution:** How are edge attributes distributed in time and topology? (WHERE, WHEN)
Q5) **Trends:** Is there a clear trend over time and where are the anomalies? (WHAT)

The network shown contains 11 nodes and 6 time steps (Q1), and appears of medium density (Q2). By default, cell size and color indicate the cell weight (number of publications per year) encoded redundantly. Color ranges from light gray for low edge weight, to dark blue for high edge weight (*value encoding*). Edges with high values stand out across the entire cube while smaller cells make it possible to see inside the cube.

Cells appear in the entire cube with a denser cluster in the "upper left" corner (Q3). To better show edge weights in dense areas, we switch to an alternative value encoding ⓑ (circled letters refer to interface components in Figure 6.3) that maps edge weights

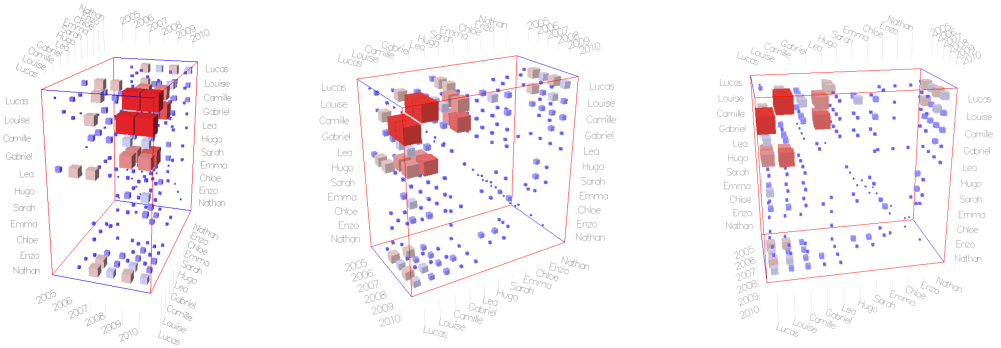


Figure 6.7.: *Matrix Cube rotated. Cell size and color indicate edge weight; blue indicates low weight, red indicates high weight.*

from blue (low) via gray to red (high)(Q4). This color scale was chosen because the two extreme values (blue and red) have equal lightness and gray in the middle is a fairly neutral value. In order to avoid occlusion, users can freely rotate the cube and obtain an overview of the data (Figure 6.7).

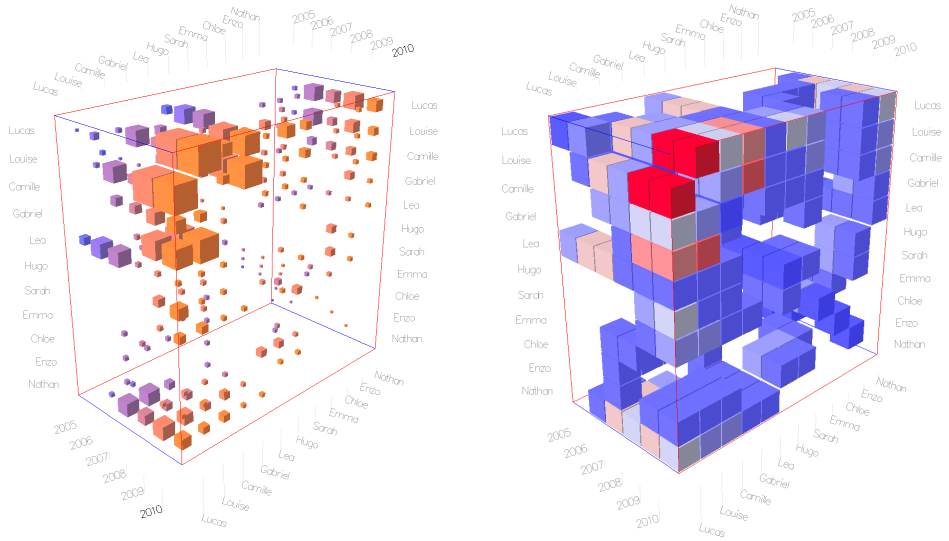
As long as the user smoothly rotates the cube with the mouse, he perceives the cube in 3D and can very well differentiate between cells in the foreground and in the background. However, as he stops rotating the cube, the 3D effect vanishes and the cube "collapses" into a 2D image. To help the user further getting an idea of the distribution of cells over time, we switch to time-color encoding, using the radio buttons in ⓑ, and obtain Figure 6.8(a). For the *time encoding* we use a gradient ranging from blue (early time steps) via violet to orange (later time steps). This mapping was chosen because it ranges from a dark and fairly cold color to a light and warm color. The additional step of violet is to obtain more shades. In addition to that, I had to make sure that color scales (time encoding, value encodings) are sufficiently different from each other, so that they can be easily distinguished (during the exploration process, but also on screenshots). Time encoding, shown in Figure 6.8(a), shows that the cube contains much more and larger bright and orange cells than dark and blue ones. Hence, the network is growing (Q5).

Small cells can bias the way we perceive network density (Figure 6.7). Figure 6.8(b) shows a mapping where all cells have equal size ⓒ and entirely fill their "slot" in the cube. In fact, the entire cube contains 208 cells, which equals a density of 34%.

6.3.2. The Network's Topology

After that initial overview, we want to gain information about the network topology (WHERE):

Q1) **Clusters and central actors:** Which are the main clusters, central actors and outliers?



(a) Time encoding for cell color, cells encode edge weight.

(b) Value encoding for cell color, cells have equal size.

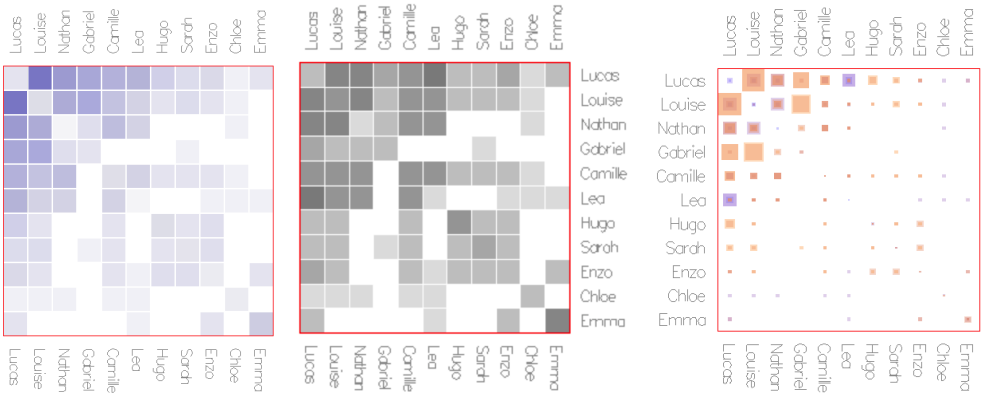
Figure 6.8.: *Collaboration network in 3D view with (a) time encoding and (b) value encoding.*

Q2) **Aggregated number of edges:** Which regions are connected in most of the times?
Q3) **Aggregated edge weight:** Which regions are connected by higher edge weight?
Q4) **Stability/Instability:** Which regions are affected by change, which regions remain stable?

Clicking on the Cubelet's right face ⓐ rotates the cube to the time-projection view shown in Figure 6.9(a). Cubix switches from a perspective projection to an orthogonal projection and all cells in the same time vector become superimposed. We adapt cell translucency ⓗ so that cell values get aggregated. Since cells all have the same size, we see the large cluster involving most of the authors, as well as a second smaller cluster in the lower right corner (Q1) containing low-weighted edges (Q3). Both clusters are highly interlinked involving Lucas, Louise, Nathan and Camille.

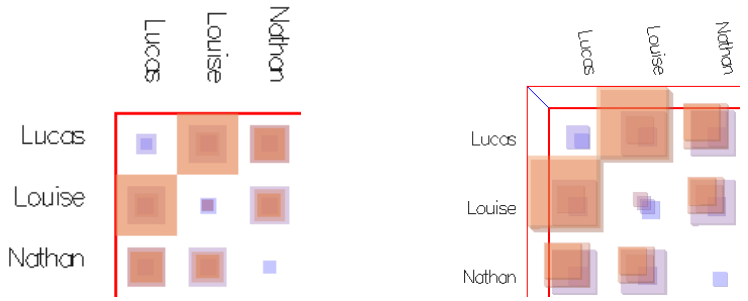
In order to know in how many years a collaboration has happened, we make all cells the same color (*None*-option checked in ⓑ), which results in the view shown in Figure 6.9(b)). Now only the number of cells in time vector are responsible for the darkness value of the cells. While most papers have been written between Louise and Lucas, the *longest* period of collaboration is between Lucas and Lea (Q2).

We are now interested in instability in the topology and switch back to the time encoding while mapping edge weight to cell size (Figure 6.9(c)). The major cluster contains large orange cells indicating growing edge weight. The pair Lea-Lucas show decreasing edge weight, because the corresponding time vector contains a large purple



(a) Value encoding (b) No color encoding (c) Temporal encoding

Figure 6.9.: *Cell color and size mappings in Time-projection view. (a) Cell color is mapped to time, and cell size to edge weight. (b) Constant cell size, with color mapped to edge weight shows accumulated edge weight over time. (c) Constant cell size and same color (gray) for all cells gives an idea of the number of edges in time vectors, independent of edge weight.*



(a) Superimposed (b) Shifted

Figure 6.10.: *Superimposition of translucent cells in projected views using temporal encoding. (a) exact superimposition, (b) slightly shifting time slices gives a quick preview of previously-hidden cells.*

cell (Q4). Gabriel's connections seem exclusively recent, since they do not overlap with any purple cells.

Figure 6.10 shows a close-up on the authors involved in the major cluster. The amount of orange indicates strong and recent connectivity between all of the cluster's members. Each of them has some individual publications long ago (purple cells), except Louise whose time vector is darker (more superimposing cells). Also visible is that the most publications have been co-authored by Louise and Lucas. Yet, a slight shift occurs, from Lucas and Louise collaborating with Nathan (purple framed cells Figure 6.10(a)), to collaborating more mutually (orange framed cells in Figure 6.10(a)).

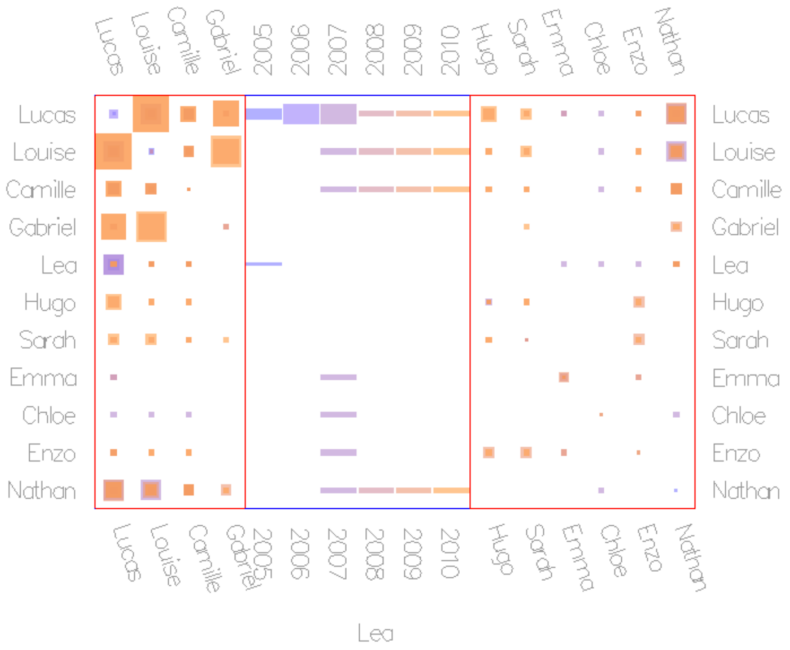


Figure 6.11.: *Rotation of the node slice corresponding to Lea. Cells inside the rotated slices all have the same length, while their height is still mapped to edge weight.*

6.3.3. Previews

Since time has been (almost) completely omitted from the topology view, many open questions remain:

Q1) **Edge weight evolution:** How does the edge weight between a particular node pair evolves? (WHAT)
Q2) **Part of subgraph:** When is a particular node part of a cluster? (WHEN)

An easy way to get a better impression of time in the topology view, is to slightly tilt the cube causing time slices to shift (SHIFT+drag mouse (panning)). Figure 6.10(b) shows how cells in the time vectors appear and are better visible. The figure shows better the different cell sizes: cells sizes (edge weights) decrease between Nathan and Louise, while increasing largely between Louise and Lucas (cf. Figure 6.10(a)). Edge weights remain stable for Louise's individual publications and between Nathan and Lucas (Q1).

From the topology view, we noticed that Lea stopped collaborating with Lucas at some point in time. In order to know when this happened, we *rotate* Lea's node slice by 90 degrees and yield Figure 6.11. Rows in the rotated slice align horizontally with those of the projected slices, while columns now indicate time. To better compare cells sizes along time, cells only vary in height, but have the same lengths to link cells in the same time vector. Lea's slice reveals that she still collaborates with other authors (Q2) in the dense cluster as well as Nathan (bottom row). However, she was most productive in 2007.

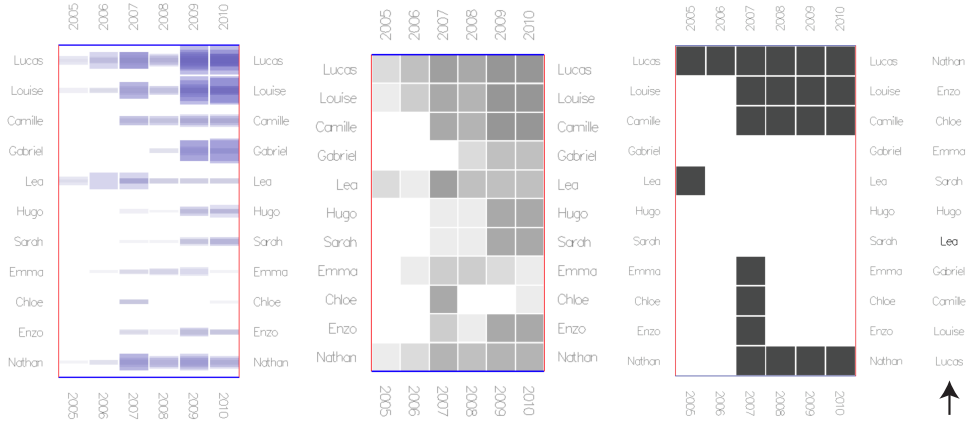


Figure 6.12.: *Temporal trends in the node-projection view using different cell size encodings. Time runs from left to right.*

6.3.4. Temporal Trends

Now, we are interested in the general temporal evolution of the network:

Q1) **Evolution of density:** How is the trend in density? (WHAT)
Q2) **Node degree evolution:** How does the degree of individual nodes evolve? (WHAT)
Q3) **Regional density:** Which regions get denser/sparser over time? (WHERE)
Q4) **Activity:** How "pro-active" are nodes, i.e. does their degree increase or decrease? (WHAT)

Clicking on the Cubelet's left side first rotates Lea's slice back in-line and then shows an animation that rotates the Matrix Cube by 90 degrees to the right, resulting in the view in Figure 6.12(a). We switch to value encoding since time encoding is redundant, while keeping the *cell-size-to-height* mapping. In this *node projection view*, we see a heat-map with one column per time step, time running from left to right. Rows still correspond to the network's nodes. In this view, cells in the same neighborhood vector superimpose and yield the aggregated connectivity per node for every year.

The general trend in this research group is towards more publications and collaboration (Q1) indicated by darker cells due to superimposition of translucent elements, except for 2008. Even the large cluster, involving Lucas, Louise, and Nathan grew steadily, except in 2008 (Q3, Q4).

When comparing rows, we compare the connectedness of individual nodes over time. To avoid bias by self edges, they can be hidden manually (checkbox in ⓘ). We see that four authors collaborated throughout the entire time span (Lucas, Louise, Lea, Nathan). Several others (e..g Camille, Gabriel, Hugo) seem to have joined the group in 2007 (Q3).

6.3.5. Individual Connections

After obtaining a literally "superficial" overview over the network's topology and its general temporal trends, many questions require a detailed analysis of the cube's interior.

Q1) **Individual connections:** When does a node collaborate with whom? (WHEN)
Q2) **Stable connections:** Which nodes are always connected to the same nodes? (WHEN, WHEN)
Q3) **Unstable connections:** Which nodes have connections with many different nodes, at different times? (WHEN, WHAT)
Q4) **Evolution of weight for an individual connection:** How does the connectivity between an individual node pair evolves? (changes in the same time vector) (WHAT, WHEN)
Q5) **Trends in connectivity:** Which trends in connection weight exist? (Comparison of time vectors) (WHAT, WHEN)

From Figure 6.12(b), we see Lea publishing only with Lucas until 2007. If we switch to an equal encoding for all cells ⓑⓒ, we see that Lea in 2007 has one of the highest degrees in the entire network. In order to know with whom she collaborated, we have two options. (1) Her slice can be rotated vertically, similar to Figure 6.11. (2) More elegantly, we can hide all other slices. Therefore, projection views in Cubix provide a list with labels from the hidden dimension on the right side of the cube, indicated by the arrow in Figure 6.12(c). Hovering Lea's label keeps only her slice in the cube and reveals her collaborators in 2007. Note that cell color and size have been set to be equal to obtain the view in Figure 6.12(b), except that cells have been rendered opaque. Lea's slice further reveals that she collaborated with almost all people in the data set (Q1), indicated by only three completely empty rows. The row for Lea shows one cell, which indicates the one individual publication Lea had in 2006 (showing self-edges again ⓘ).

In order to compare multiple node slices, we could rotate as many of them as necessary, while remaining in the time-projection view. Eventually, all slices would be rotated and would face the observer. A shorter way to achieve that view is to drag the mouse on the left face of the Cubelet. An animation is shown decomposing the cube and laying out slices as in the node-small multiples view in Figure 6.13.

The overview on all node slices reveals who has been most active and for what period of time. Slices for Hugo, Sarah and Enzo show very similar patterns. They started collaborating recently and collaborate with the same people (Q1). The same is true for Lucas, Louise and Camille. Most collaborations are stable (Q2), indicated as almost no row shows holes, meaning that once a pair have started collaborating, they continue or stop for ever (Q2,Q3). There are only two node pairs not following this pattern: Enzo—Lucas and and Enzo—Chloe. In fact, having a closer look to Enzo's slice, shows that he did collaborate a lot in 2007. It might be that he was an intern in 2007 and officially joined the group later.

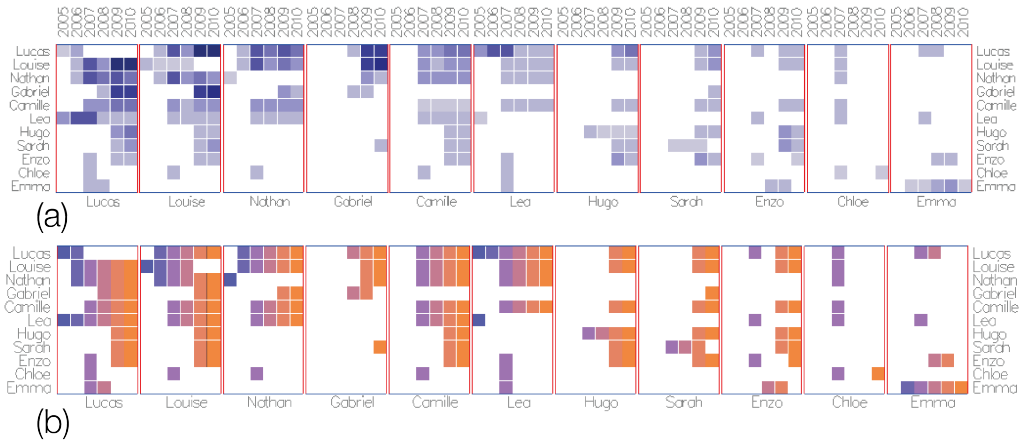


Figure 6.13.: *Node Small multiples view illustrating both color mappings. (a) Cells are colored according to edge weight, highlighting differences in weight across slices. (b) Mapping color to time-index facilitates topological comparisons across slices.*

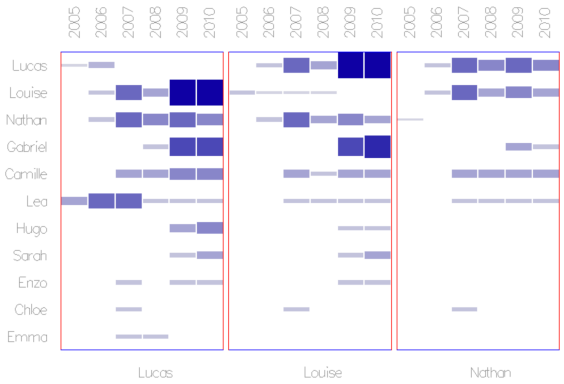


Figure 6.14.: *Three Node slices using value encoding. Cell size also encodes edge weight, while cell width is equal, linking cells in the same time vector visually.*

Figure 6.14 shows more detail for Lucas', Luise's and Nathan's slices. Comparing the evolution of connectivity (time vectors) shows that Nathan's collaborations are constant but slightly decreasing, while Louise yields the most increase in collaborations. She co-authors most of her publications with Lucas (Q4). Camille and Nathan are constant collaborators.

Switching to the time-index color mapping (Figure 6.13(b) makes it easier to compare ego-network topologies for a given time step. The general trend of collaboration is towards more collaboration and number of papers (Q5).

6.3.6. Topology of Individual Time Steps

At this point of the exploration, we know about the network's clusters, its central actors, important years, as well as how individual authors collaborate over the years. Finally, we want to compare individual time steps, to answer the following questions:

Q1) **Compare times:** How much do time slices differ between each other? (WHEN, WHERE)
Q2) **Evolutionary process:** Can the network evolution process be divided into periods and states? (WHEN, WHERE, WHAT)
Q3) **Recurring Patterns:** Are there typical topological patterns that reoccur over time? (WHEN, WHERE, WHAT)

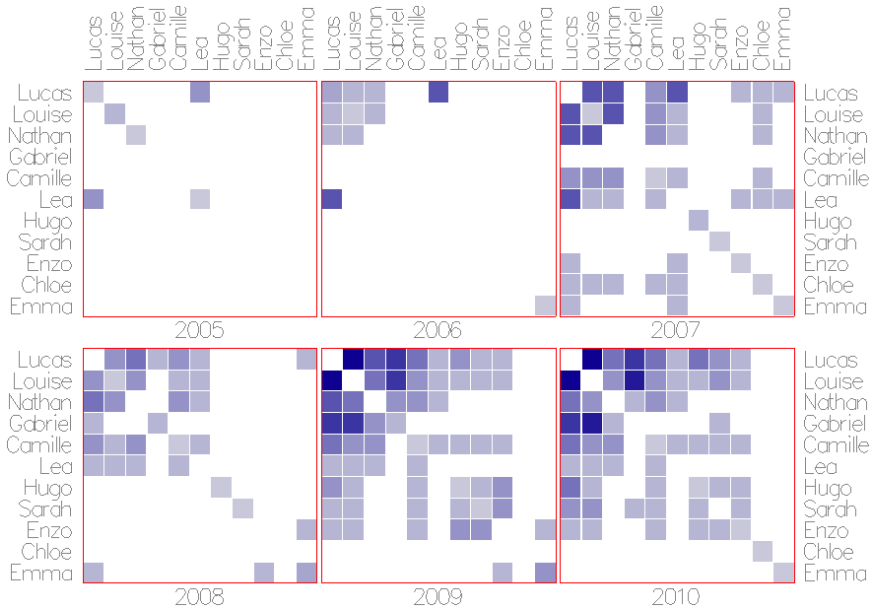
In summary, Cubix provides five ways to show a single or a few individual time or node slices:

1. hover the label of a slice in the 3D view (Figure 6.16(a)), or in the corresponding legend in the projection views (Figure 6.12(c)),
2. click the slice label to make the slice permanently visible, but hide the others,
3. select a time range on the time-ranger slider ⓕ (for time slices only),
4. layout all slices side-by-slide Figure 6.15, or
5. rotate an individual slice inside the cube (cf. Figure 6.11).

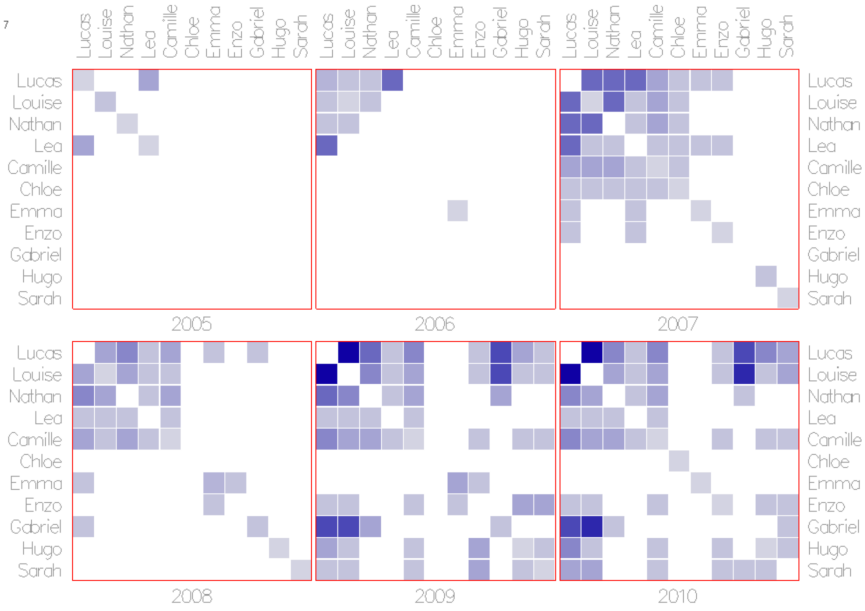
In Figure 6.16(a), the label for slice 2009 has been hovered, causing all other slices to be hidden. In order not to lose context completely, hidden slices are "ghosted", rather than completely removed; the left handle of the opacity slider ⓗ, labeled "F", is used to set translucency for *filtered* slices, while the right handle sets translucency for *visible* slices.

We switch to the time-projection view and, using the list of all times on the right side, leafing though slices by sliding over the labels with their mouse. One or several slices can be selected to compare two time steps. Figure 6.12 showed 2008 as an outlier, against the general trend towards more connections and higher edge weight in the network. By selecting the two years 2007 and 2009, and switching to a time encoding, Figure 6.16(b) reveals the direct differences between the years preceding and succeeding 2008 (2007=purple, 2009=orange). Most connections in 2008 are not present in 2007 and vice versa, except for some pairs in the cluster on the top left (Q1).

To see the different stages of topology evolution at-a-glance, dragging the mouse on the Cubelet's right face triggers an animation that decomposes the cube into individual time slices. Using color encoding for cells, Figure 6.15 shows four characteristic topological configurations (Q2) that explain the different patterns observed in Figure 6.12. 2005 and 2006 indicate a primary period in the research group, with only a few people contributing. 2007 then involved several new people, including Lea as a major central actor. Some of these people vanished again one year later (2008), partially reappearing in 2009. In 2008

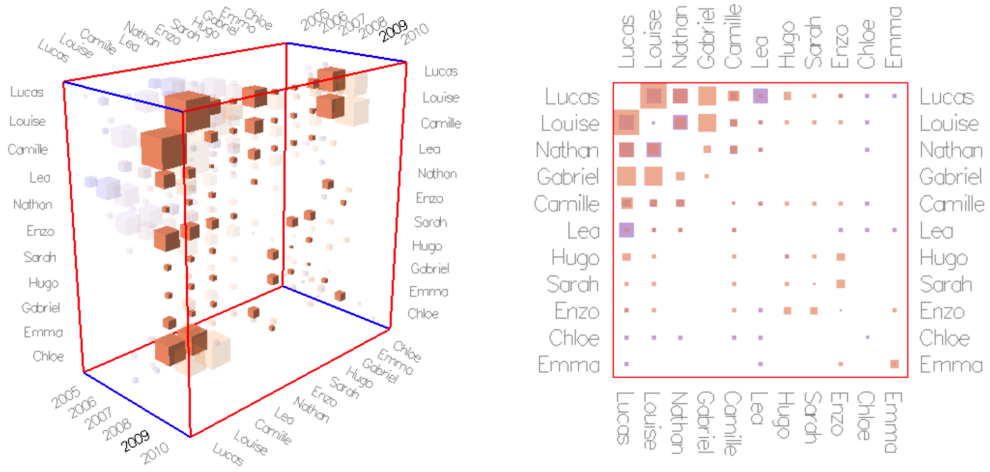


(a) Global ordering



(b) Ordering according to 2007

Figure 6.15.: *Time slices juxtaposed in the time-slices view. Darker cells correspond to more publications between people. Ordering by different time slices (a,b) reveals different patterns.*



(a) (b)

Figure 6.16.: *Exploring individual time slices; (a) Individual slice (2009) is hovered in the 3D view as the user hovers over the corresponding slice label, (b) selection of two slices (2007, 2009) for direct comparison in the time-projection view. In (b) purple cells belong to 2007, while orange cells belong to 2009.*

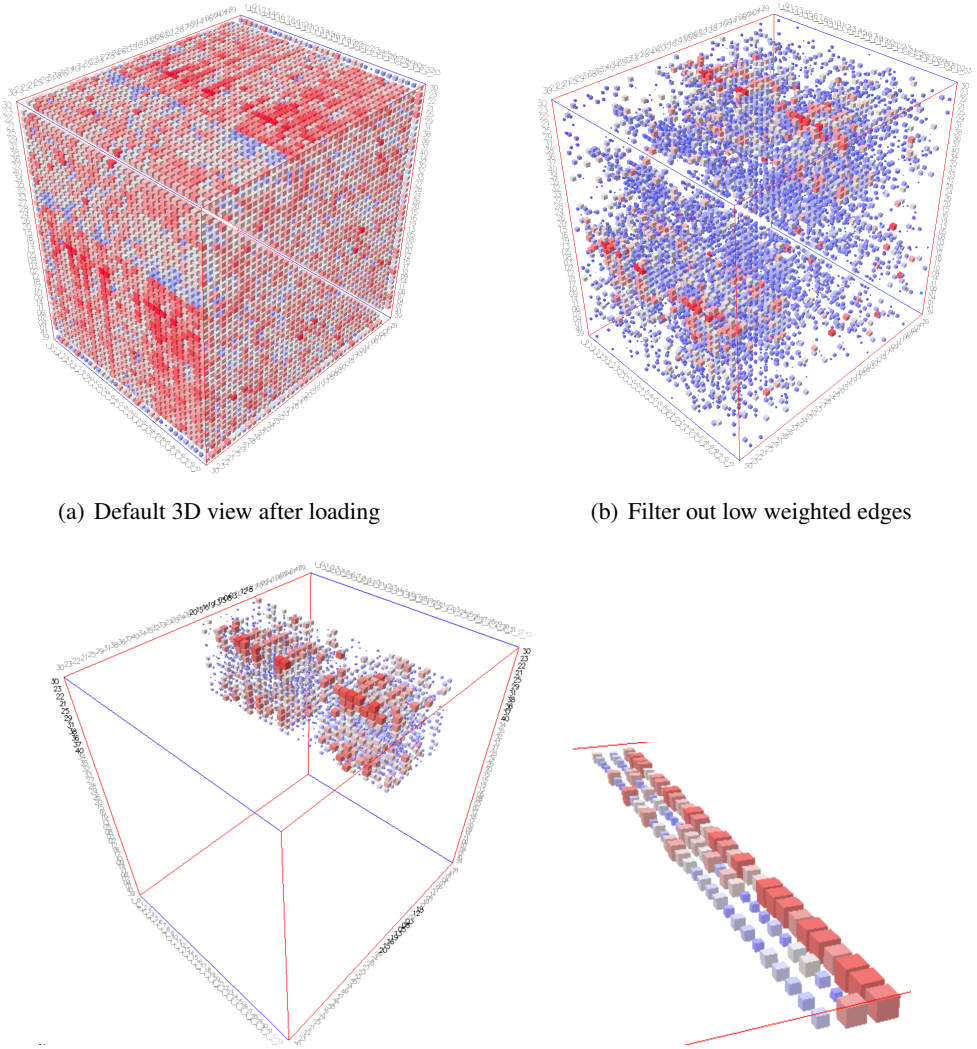
only the group's core existed. One year later, the group again gained new members and kept stable in 2010 (Q3).

Note that, due to the global matrix ordering, outlier topologies such as in 2007 stick out visually due to a more fragmented cell layout. In larger networks, fragmented time slices are likely to appear more often and reduce legibility. When only a subset of time steps are selected (using time slider or mouse selection), users can reorder the entire cube ⓓ, by taking only the selected time steps into account, potentially yielding more meaningful visual patterns. After reordering, users show again all slices; time slices that are more fragmented indicate time steps with a more different topology than the selected ones.

This rough overview over the collaboration network, already revealed many trends, patterns as well as detailed information. The group's density is steadily growing, as well as the numbers of papers co-authored between any two authors. New people seem to have arrived during the years, being highly connected to the one central core, which likely included the group leader(s). However, collaborations are generally equally distributed among all group members.

6.4. Cell Filtering

The example network from the walkthrough scenario was small and relatively sparse, compared to the majority of networks. Figure 6.17(a) shows a almost complete network



(a) Default 3D view after loading

(b) Filter out low weighted edges

(c) Filter node slices (horizontally and vertically)

(d) Filter time vectors

Figure 6.17.: *Effect of cell filtering in an almost complete network.*

of communication between antennas (see Section 8.4). Signal strength is indicated as edge weight.

Cubix offers multiple ways to filter cells that let users specify which ones should be shown or hidden in the cube (and thus across all views).

- The **edge weight range** slider makes it possible to hide edges whose weight is below or above some threshold, which is especially useful when the cube is complete. Selecting the *Adapt Weight* check box ⓓ maps the weight of visible cells into the range of minimal and maximal cell weight. As an effect, the low weighted visible cells become even smaller (making the cube appear sparser) and allow to perceive more details in the edge weight distribution of the selected range (Figure

6.17(b)). As an option the mapping from weight to cell size may be changed from a linear mapping to a logarithmic one (*Logarithmic Scale* ⓓ).
A further option (*Inverse Filter* ⓓ) allows to invert the edge weight slider, i.e. the part selected by the range is removed from the visualization. This function is particularly useful to investigate extreme edge weights (see Section 6.6.3).

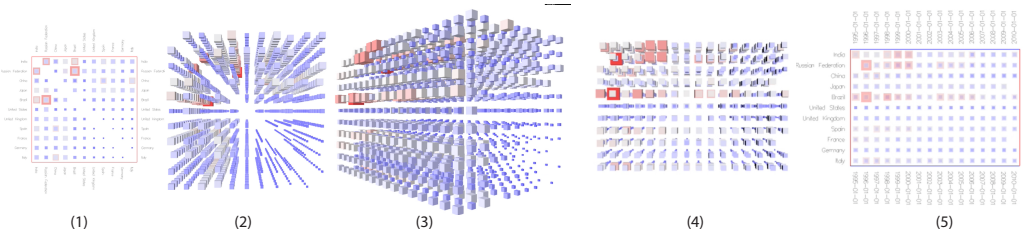
- **Lasso selection** works in any view and leaves only the selected cells and those behind. Figure 6.17(c) shows an isolated block selected while in the node-projection view.
- **Selecting one or more slices**, by right clicking slice labels in the 3D view, automatically hides all others. If both horizontal and vertical slices are selected, only the cells at the intersection of all selected slices remain visible. Hovering over a hidden slice's label temporarily reveals that slice. Figure 6.17(d) shows several vertical and horizontal node slice selected, leaving only the cells in the intersection visible.
- **Clicking a cell** once hides all slices but those that contain this cell. Clicking it twice hides all cells but those that belong to the same time and neighborhood vectors.

6.5. Animated View Transitions

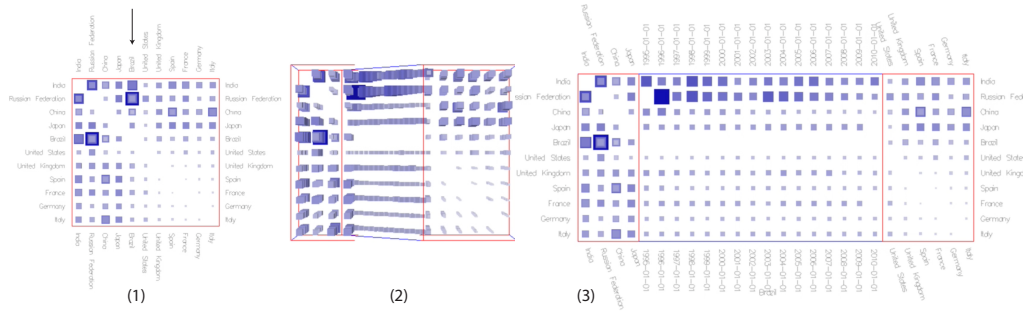
To help users understand the relationship between views on the Matrix Cube, transitions between individual views are smoothly animated transitions (Robertson et al., 1993). Those transitions are designed to (a) help users identify the current direction the cube is sliced (by time or by nodes slices) and (b) support users' interpretation of the new view. Each animation within a transition changes one particular element or parameter of the representation. There are animations to move the camera, animate between perspective and orthogonal projection, move or rotate slices, and fade labels in or out. Animations on the same element (e.g., a slice, or the camera) are played sequentially, while animations on different elements are played in parallel, for the most part.

The three following types of transitions exist between individual views of the cube.

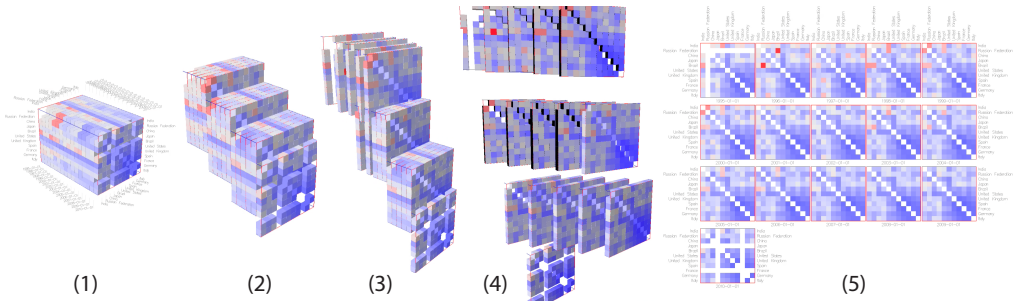
- **Rotation + Projection:** When switching from the 3D view to one of the projected views, the camera is moved, and then the projection is changed from perspective to orthogonal. Likewise, switching from time-projection to Node-projection views (Figure 6.18(a)), (1-2) the projection is changed from orthogonal to perspective, then (3) the camera is rotated around the cube, and finally (4-5) the projection is changed back to the orthogonal projection. Labels remain in their 3D position but are adjusted and rotated to always face the user. The total duration of this animation is 1 second.
- **Slice Rotation:** When rotating a slice in the Time-projection or Node-projection view (Figure 6.18(b)), first (1-2) the projection changes to perspective, then (2) the



(a) Cube rotation from Time-projection view (1) to Node-projection view (5).



(b) Rotating node slice "Brazil" (arrow) in the Time-projection (1).



(c) Cube decomposition from 3D view (1) to Time small multiples view (5).

Figure 6.18.: *View transitions in cubix.*

slice is rotated, all other slices being pushed to the sides to make room for it, and finally (3) the projection changes back to orthogonal. The default duration of this animation is 0.5 second.

- **Composition and decomposition of the cube:** Decomposing the entire cube and laying out all of its slices in a grid pattern is the most complex transition. We tested several possibilities including an animation that manages slices as if they were pages on a stack of paper and, starting with the far most one, slices are moved to their new position using a slow-in / slow-out pacing function. This transition turned out out cause a lot of visual noise, especially if the transition has to handle many slices by remaining within a short duration (maximum of 1.5 seconds). We finally decided on the staged transition shown in Figure 6.18(c). (1) From the 3D view, (2) slices in the cube move in together to their final vertical-position in the small multiples grid (5), as if the cube were sliced only partially. Then (3-4) slices

get moved to their final horizontal-position, starting with the top slices slightly in advance to produce a fluent animation.

There are some views between which no direct transition exists, such as between a node small multiples view and a time small multiples view. In cases that require the cube to be sliced, independently whether it is already sliced, a first transition leads to the 3D view, and a second transition leads to the target view. For instance, this happens when switching from Node to time slices and vice versa. A slice rotation view first rotates all its rotated slices back, yielding a corresponding superimposition view, before starting the transition to the destination view. The 3D cube acts as a pivot, providing clues about the relationship between the views. While novice users might prefer slow, detailed transitions, experts might prefer to navigate quickly between views. Animation speed can be adapted using a slider ⓗ.

6.6. Validation with Experts

Visualizing dense, weighted dynamic networks is a problem in multiple application domains. For a first evaluation of Matrix Cubes, we contacted two researchers that face the issue of visualizing dynamic networks: an astronomer and a neurologist. We wanted to focus on the overall usefulness of Matrix Cubes and usability of Cubix through an open-ended evaluation using real data, so as to answer questions such as: Is the metaphor understandable? How well do experts manage to manipulate the cube and interpret the views? Can they gain insights about their data from this visualization?

6.6.1. General Methodology

Both experts were approached independently and asked if they would be interested in trying to visualize their data with Cubix. We demonstrated Cubix using the co-authorship network described earlier in this paper. In both cases, the experts' feeling was that the approach had potential, and they agreed to provide us with actual data. Once their data had been imported in Cubix, we met again with each of the two experts for a longer session, walking them through the different views and features based on a sample of their own data. Experts then took control, looking at their data in more detail by themselves under our guidance, as detailed below.

6.6.2. ALMA

The Atacama Large Millimeter/submillimeter Array (Pietriga et al., 2012) is a state-of-the-art radio-telescope composed of 66 high-precision *antennas* currently under construction in the Chilean Andes. Observations are based on the principle of interferometry: a source in the sky is observed by a subset of 12-to-50 of those antennas, called an *array*.

In a given array, all antennas are connected to the other ones in a pairwise manner, thus forming a fully-connected network. Edges of this network, called *baselines*, have several quantitative time-dependent variables associated with them, which astronomers are interested in looking at.

Astronomers currently use a tool developed in-house to plot these individual variables as scatterplots or line charts. While these charts may be able to produce legible visualizations for antenna-based variables, they do not scale to the number of baselines in an array (typically 1225 for 50 antennas). Still, astronomers need to either monitor in real-time or analyze a posteriori those data over numerous time steps.

Following the general methodology described above, we approached an ALMA astronomer. The session consisted of two parts. We used one of the four baseline-based variables available in the data he had provided (see below) for the walkthrough. Then the astronomer took control of the computer, and visualized all four datasets, commenting on his findings and insights, and asking for help to get to a particular view when stuck (he was told that he could freely ask for our assistance). These two phases lasted 22 and 45 minutes, respectively. The entire session lasted 1.5 hours, including preliminary explanations and post-hoc discussions. We recorded the session and took pictures of meaningful views on the data for analysis once back in our lab.

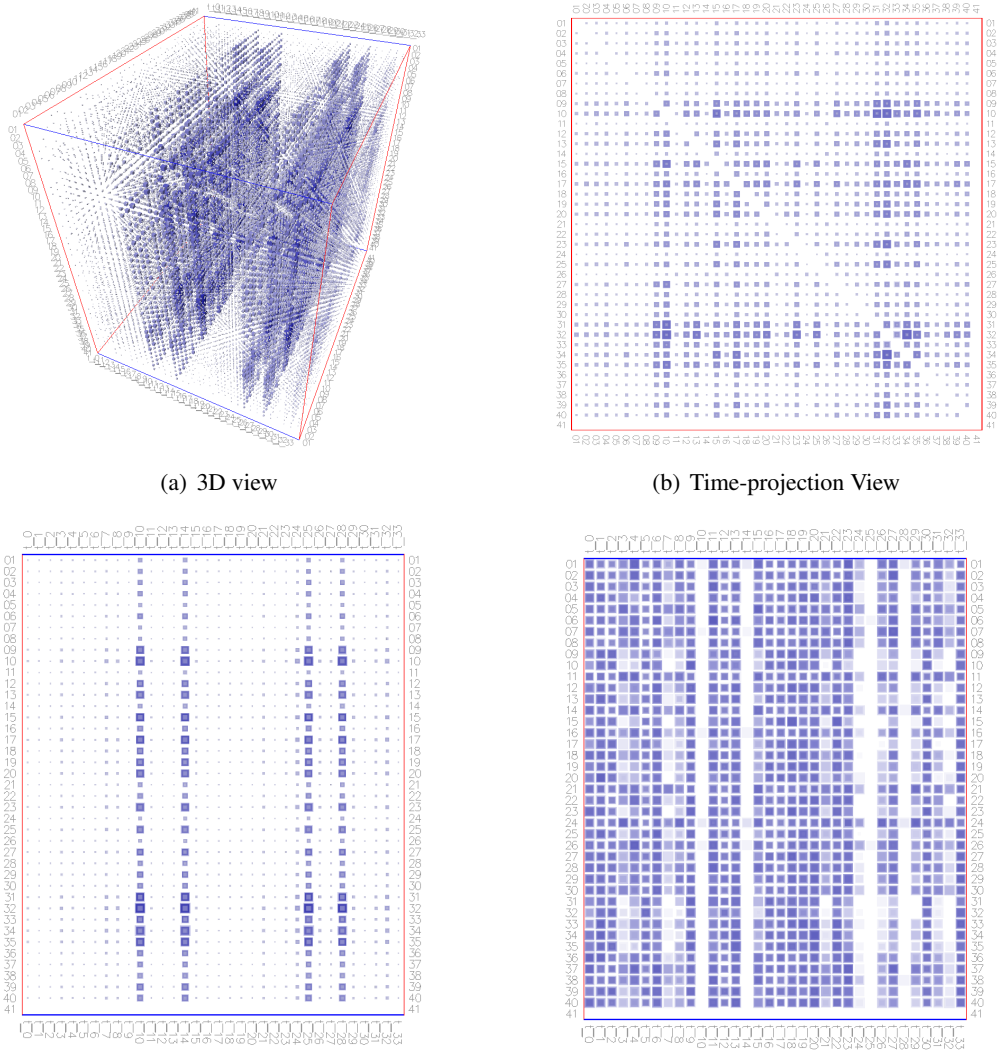
Data

The dataset corresponds to a calibration, used to compute very precisely the (x,y,z) geographical position of antennas. For this, antennas point at a series of quasars that act as reference sources. Quasars are pointed at, one after another, by all antennas simultaneously. Each quasar represents one of the 34 time steps in the data. The data consists of four variables, each visualized independently in a separate cube: DELAY, Amplitude (AMP), Amplitude RMS (RMS), and signal-to-noise ratio (SNR). All are measured for 42 antennas, i.e., 861 baselines, for one particular baseband and one polarization only. Each cube thus consists of $42 \times 42 \times 34 \approx 60{,}000$ cells.

Usage Scenario

We first provide a summary of the visual exploration process over the entire session, focusing on the main steps related to the key findings which are discussed afterwards.

RMS We used the RMS data to give a walkthrough to the astronomer (Figure 6.19). Straight from the 3D overview, four time slices stood out, featuring larger values than other time steps, as illustrated in Figure 6.19(a). After briefly showing the time-projection view (Figure 6.19(b)), we quickly switched to the node-projection view at the expert's request, as he was interested in getting a better look at the evolution of values over time (Figure 6.19(c)). From this view, he confirmed his intuition that the four time slices



(a) 3D view

(b) Time-projection View

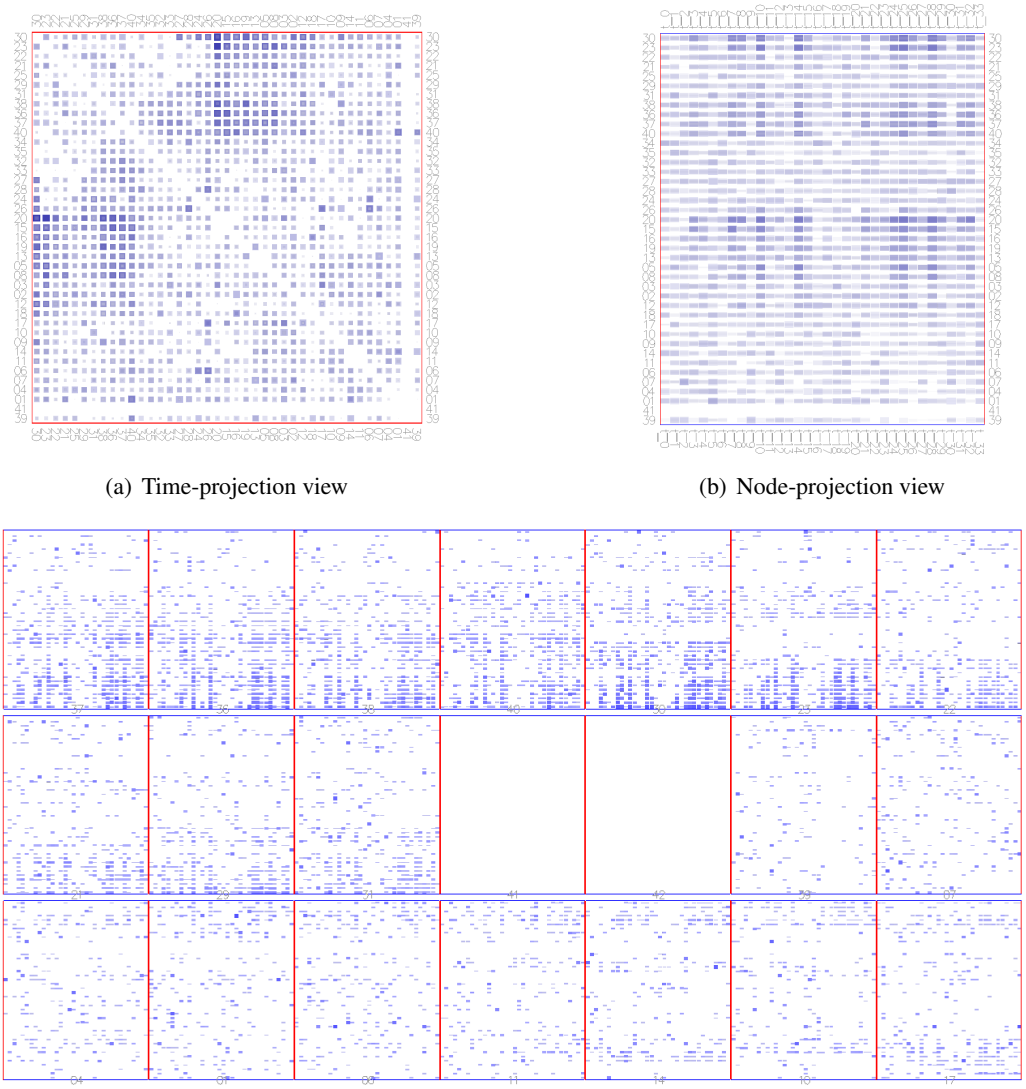
(c) Node-projection View

(d) Node-projection View

Figure 6.19.: *ALMA* RMS *data set. Antennas are ordered by name. (a) with four time slices seem to stick out 3D View, (b) Time-projection view, (c) Node-projection view, and (d) again Node-projection view while high weighted edges are filtered out.*

spotted in the 3D overview were quasars that were too weak and should probably be removed from the catalogue of sources used for calibration. Based on the fact that always two of the four columns in the node-projection view look exactly the same, he inferred that these four time steps might actually correspond to the same quasar observed 4 times, or to 2 quasars observed twice each.

At the expert's request, we filtered out higher values corresponding to the weak sources, so as to be able to better compare the cells from other time steps visually (the value-to-color mapping adjusts cell color dynamically as higher-value cells get filtered out). The result looked homogeneous (Figure 6.19(d)), leading the astronomer to the conclusion that



(a) Time-projection view

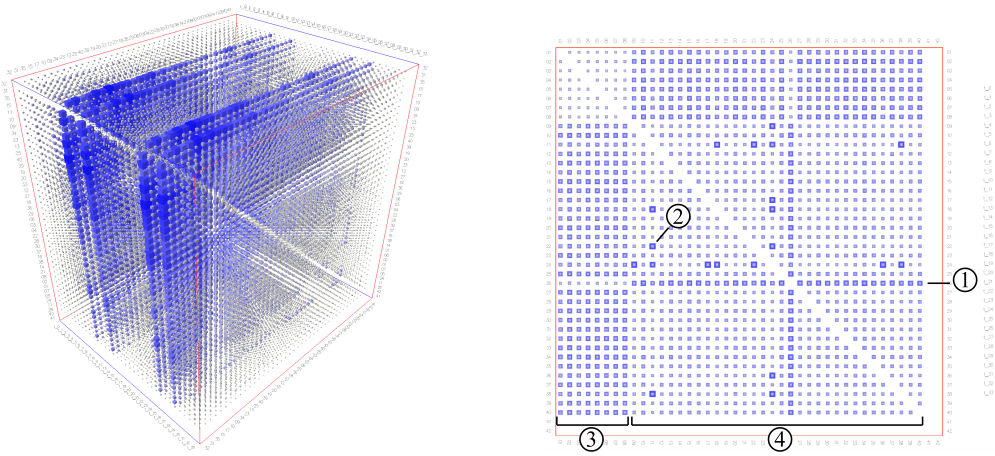
(b) Node-projection view

(c) Node small muliples

Figure 6.20.: *ALMA* DELAY *data set in different views. (a) Node-projection view, (b) Time-projection view, (c) node slices as small multiples. View cropped to show only 21 out of 42 slices. Similar patterns (reversed) appear in the lower half.*

there was no antenna-based problem, which would have appeared visually as one-or-more entire rows of cells with high values. We then finished the walkthrough by explaining both time and node small-multiples views, illustrating features such as brushing.

DELAY Now familiar with Cubix's main features, the astronomer started exploring the data on his own, switching to DELAY Figure 6.17(a). In this data, he was looking for baselines whose delay varies over time, as this was not supposed to happen (delay should be more or less constant). He switched to the node-projection view Figure 6.20(a), which shows time steps as columns. However, as all node slices were superimposed, it was



(a) Overview in 3D view (b) Time Projection View

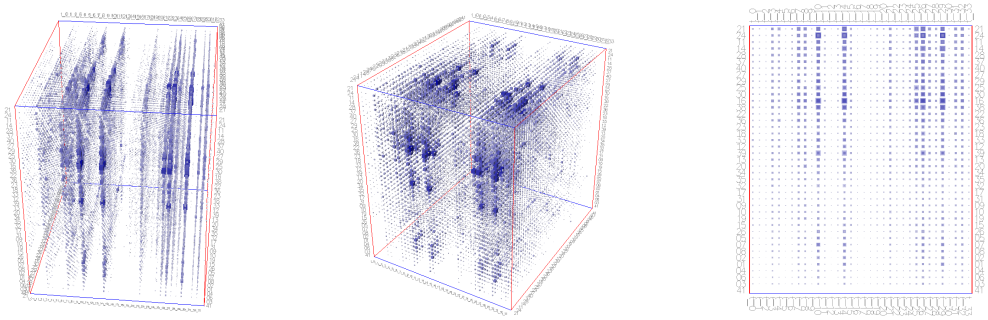
Figure 6.21.: *ALMA* AMP *dataset. (a) Overview of* AMP *in which four time steps seem to stand out. (b) The time-projection view on* AMP*, showing an abnormal behavior of one specific antenna (1), various baseline specific outliers such as (2). A different behavior of the first eight antennas compared to the other ones is clearly visible: (3) vs. (4).*

difficult to spot specific baselines that would feature significant variability. The expert realized this, and decided to switch to the time-projection view Figure 6.20(b). Some rows and columns stood out, indicating possible antenna-based problems.

From there, he asked that we switch to the view that shows node slices as small multiples, as he wanted to get an overview of what particular antennas feature higher variability, something that can easily be seen in this view by visually scanning the whole set of small multiples (Figure 6.20(c)). This view revealed another pattern: for some antennas, roughly half of the baselines had much larger delays than the other half. The astronomer switched back to stacked node slices to quickly brush through the slices in more detail. There was no clear explanation for this observation, which requires further investigation by looking at additional data not available here.

AMP We then switched to AMP data, shown in Figure 6.21. Figures 6.21(a) and 6.21(b) show the same 3D view but the astronomer was very interested in the time-projection view, as the two main types of problems were readily apparent. First, antenna-based problems affect all baselines that connect to it, which visually translates into the corresponding row/column featuring a lot of variability (see ①). Second, problems affecting a single baseline, possibly due to, e.g., atmospherical changes, which visually translate into isolated cells featuring a lot of variability (see ②). Another pattern that clearly stood out in this view is the difference between the eight left-most columns ③ and the following ones ④, attributed to the different types of antennas that compose ALMA[2]. Finally, the

[2]We discussed the possibility that the antenna in ① could belong to group ③. This was ruled out because of the way the data was generated, with antennas sorted alphabetically, and thus by type.



(a) 3D view (b) 3D view (c) Node projection view

Figure 6.22.: *ALMA* SNR *data set showing similar patters to the* RMS *and* AMP *data sets.*

astronomer looked at SNR data, identifying similar visual patterns, and then finished the session.

6.6.3. Brain Function Activity

The second expert, a researcher in neurology, studies signal patterns between regions of the brain under different stimuli, collected using EEG-correlated fMRI (Goldman, Stern, Engel, & Cohen, 2000). He is interested in identifying time steps that feature a high activity level overall, connectivity patterns that reoccur over time, and which regions are active at specific moments. All of these can then be compared across different stimuli.

The neurologist currently uses spatial brain maps to explore his data, but the physical position of brain regions is not important when analyzing this type of dataset: maps do not properly convey the network's topology. Currently, no visualization faithfully represents his data since node-link diagrams fail to provide a legible view of such dense networks. Therefore, just like the astronomy expert, neurologists typically look at time-series for each region of the brain, without any visual feedback about the topology.

The methodology employed for the neurologist was slightly different than for the first expert. The neurologist was located in Canada, and thus sessions were conducted remotely using a videoconferencing tool. The neurologist was provided with the Cubix software; the system ran simultaneously on both ends, with screen sharing enabled. We conducted two sessions, with one day in-between the two. After looking at the data and explaining to us what it was about, the neurologist asked for a few changes to the visual representation to reflect domain specific information. We planned for the second session to implement the requested changes and give him time to explore his data with the adapted version of Cubix. The second session was conducted under the same conditions, using the new visual representation.

Data

We obtained data about three different regions of the brain from multiple subjects, observed under 3 different stimuli. For each of these regions, signals for 50 different frequencies in 8-to-16 subregions were recorded. Several measures are used to characterize connectivity, such as Granger causality, and we can only show one dynamic network at a time from this rich dataset in a matrix cube. The networks visualized in Cubix consist of 8 or 16 nodes over 150 time steps (5 minutes of recorded brain activity at a 2-second sampling frequency). Measures exist between any two regions, meaning that networks are fully-connected, the corresponding matrix cubes containing from $8 \times 8 \times 150 = 9600$ to $16 \times 16 \times 150 = 38400$ cells. Figure 6.23(a) shows 15 out of 150 time steps for a network of 8 regions.

During the first session, the neurologist explained that edge weights have normalized values in $[-1.0, 1.0]$, and that edges of interest are those near the extrema. We added an additional color scale and mapping, emphasizing those values, as illustrated in Figure 6.23: 1.0 gets mapped to red, 0 to light gray, and -1.0 to blue. A new mapping of edge weight to cell size was also added, assigning smaller sizes to edges with lower absolute values. The range slider used to filter out edges based on their weight now features an option to invert the selection, so as to remove values in-between the two knobs rather than those outside the corresponding range.

Usage Scenario

Data visualized during the first session corresponded to a resting phase (no visual stimulus, subjects' eyes closed). At the neurologist's request, the second session focused on data observed during a phase featuring visual stimuli. Using the new color and size mappings revealed the symmetric distribution of red and blue cells in the 3D overview (Figure 6.23(a)). Several time steps also stood out as significantly denser than others, distributed over time. Switching to time-projection view made it possible to get a clear look at the denser time-slices, confirming the symmetry between red (sending) and blue (receiving) edges. Beyond symmetry, one pattern appeared consistently across all slices that feature a high level of activity, made even more apparent when reordering rows and columns based on their similarity: two or three regions received from exactly four other regions (blue cells in the same row) while sending signals to the remaining four regions (red cells in the same row, ①). Sending and receiving regions were the same. However, one of the two active regions also sent to the other one, not receiving any signal from it.

Switching to the node-projection view (Figure 6.23(c)), salient patterns could be observed when looking at the temporal distance between highly-active time steps. The neurologist indicated again that edges with low absolute values were irrelevant; those got removed accordingly, keeping cells whose absolute value was larger than $|0.5|$ only. We adapted translucency as illustrated in Figure 6.23(b). Rotating some time slices in-place

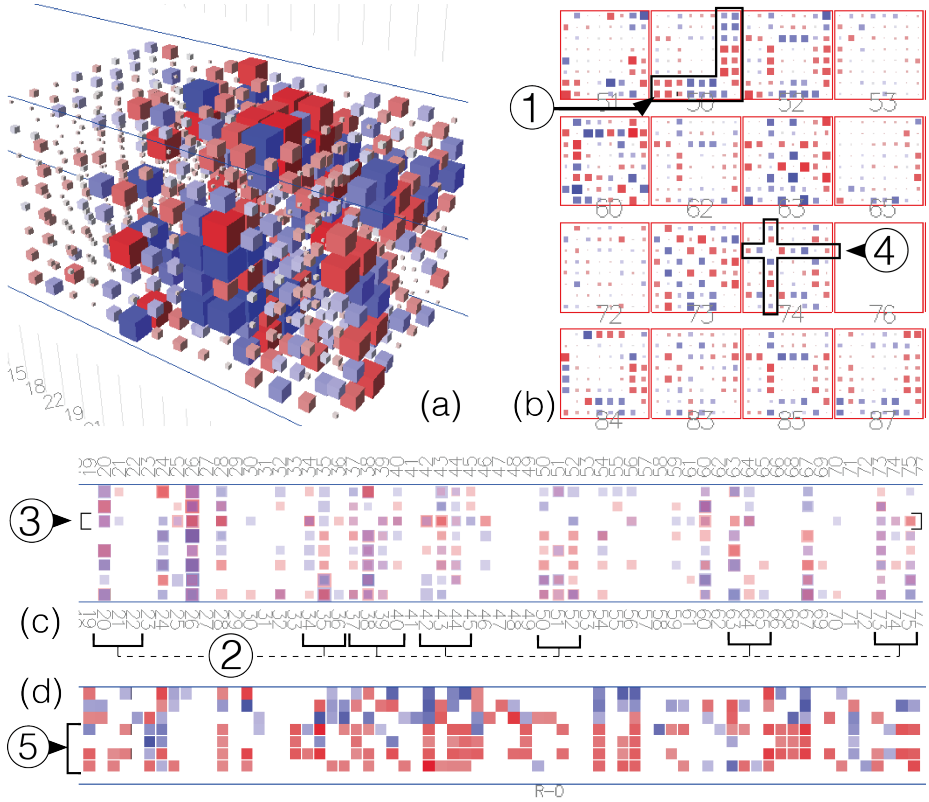


Figure 6.23.: *Brain connectivity data. 8 regions over a sample of 15 time steps. (a) Overview reveals that some time steps are more active than others. (b) Detailed view of 16 time slices. (c) node-projection view, showing activity patterns and revealing one region more active than the others. (d) Individual node slice sending more than it receives.*

enabled the neurologist to recognize the topological patterns previously spotted in the time-projection view. Temporal patterns seemed randomly distributed but occurred in intervals of 3 or 4 time steps ②. As illustrated in Figure 6.23-c, one region was more active than the others ③. Switching to time small multiples (Figure 6.23(b)) confirmed that region R-4 was indeed active in many time slices, and was often among the most active ones ④. Switching to the node small multiples, the neurologist found two regions that were more active than the others, one of them ⑤ sending more often than it received, and almost always sending to the same regions (Figure 6.23(d)).

6.6.4. Key Findings

The cube metaphor was quickly understood. It helped, along with animated transitions, to switch between views. Right from the start, both experts understood what the cube was about, as well as the views that were derived from it. For instance, as the view was animating from the 3D cube to the time-projection view, the astronomer made the following comment: *"Ah! ok, now you are stacking time."* Once he

took over, he always managed to switch between views without assistance, sometimes hitting the wrong key but quickly recognizing that this was not the view he wanted. The neurologist used both the Cubelet and keyboard shortcuts. There was only a bit of confusion for the astronomer the first time we switched to the view that shows node slices as small multiples, but that view was then well understood and used, though the astronomer asked for assistance to go back to this particular view from the time-projection view. It is worth noting that he knew exactly what view he wanted; his problem was that he did not remember how to get there.

Further evidence of the experts' understanding of the matrix cube comes from a comment the astronomer made as he was looking at DELAY data in the time-projection view. As he was looking for baselines featuring significant variability, i.e. squares within squares in a given cell, he realized on his own that for a given baseline, larger cells earlier-in-time would hide smaller cells that were behind them in this particular view, thus masking some types of variability patterns. This is indeed true; we then explained how to overcome this limitation in Cubix by, for example, adjusting cell translucency to see through them, or by switching to another view.

The astronomer also took for granted that cells filtered out in one view would still be hidden when transforming the cube to another view, without us mentioning this at all. We interpret this as further evidence that the cube was well-understood as a model/pivot view.

All views were considered useful for different tasks. According to the astronomer, the time small multiples view was very useful, but slices were very small. This fact could be partially lowered by switching to an equal size for all cells, thus keeping color intensity as only weight indicator. Better interaction capabilities (e.g zoom on slice) could easily improve this view. In the brain scenario, the time slice view revealed repetitive patterns at a glance.

Slice rotation in node-projection very useful to get quick overview over each node's activity, given that cells in the same time slice overlap. In fact, since the entire set contains too many dimensions to be visualized in a single cube, experts were looking for prominent time steps across the cube. They could then draw conclusions from the regions activity back to the visual stimuli at that time.

The 3D view was slightly but constantly moved to better perceive depth of the model. Being able to switch between different views was essential to his analysis due to the density of the data and the different patterns visible in any of these views.

Filtering, brushing and linking were useful. To look at individual slices, the astronomer used the slice list on the right side in node- and time-projection views. Sometimes he looked at one particular antenna he had identified in another view, sometimes he browsed through slices to find antennas with particular characteristics hinted at by the default view, which shows all slices stacked.

Filtering was found useful by both experts, given the density of the data and the opportunity to focus on different ranges of edge weights (low weight, high weight, extremes). The astronomer understood on his own that filtering was happening on the cube as an object, not on a particular view of it. He made one request: in addition to being able to select what slices to show, he wanted to select which slices to hide (which was at the time only possible by selecting all slices not to hide).

The astronomer further mentioned that brushing and linking in the small multiple views was very helpful to compare particular values across slices. The brain scenario contained too many time slices (150), which made it hard to compare distant time slices in detail.

Cell color was constantly mapped to edge value. The neurologist asked specifically for a new edge-weight-to-color mapping that would encode edge direction (blue, red) and used it exclusively. The astronomer was told about both original color mappings: time and value. He immediately stated that redundantly encoding time with color was of no interest to him. Instead, he wanted cell color to reflect the associated edge value.

Row and column reordering are useful, but can be confusing. Rows and columns of matrices can be either sorted alphabetically or reordered based on their similarity, so as to make some visual patterns such as clusters more apparent. By default, users are presented with the latter option. The astronomer had not realized the type of ordering, which was confusing to him at first, as he is used to looking at matrices sorted alphabetically, by antenna type and name. As a result, he was not expecting to see some of the visual patterns that appeared, and was instead expecting individual, isolated rows and columns to stand out. He realized this when we switched to *name ordering* while discussing possible differences due to antenna type. This alphabetically-sorted view felt much more familiar to him, as the visual patterns of clusters were no longer there to *"prevent him from seeing individual rows and columns."*

The neurologist specifically requested that rows be reordered based on similarity when looking for patterns in the time-projection view. However, since there was much variation between time slices, the general reordering was not very optimal. A better way would be to either select individual time slices of interest or particular time ranges, and then order those time slices.

Matrix Cubes succeeded in providing a legible visualization where other approaches had failed so far. The astronomer currently visualizes data using line charts and scatterplots. While this worked fine during the early construction phase of the observatory with only a few antennas available, this solution clearly does not scale to the over-a-thousand baselines now formed by the almost full complement of 66 antennas. What the astronomer appreciated with Cubix was that he could clearly see patterns at once, such as antenna-based problems, without having to select and plot specific antenna pairs as he has to do now, and which is very tedious: *"With Cubix, you visualize it*

instantaneously. It is much faster." He also enjoyed the fact that he could smoothly transition from one view to another. In the same vein, the neurologist commented: *"With your tool you just load the data and you see it [...] It is a very nice piece of work."*

At the time of writing, both experts expressed interest in exploring their data further with Cubix. The astronomer has prepared additional datasets that he then explored with our system. The neurologist stated that he wanted to further use Cubix. The data he looked at was the result of a recent experiment, and he wanted to spend more time exploring it as to become more familiar with it.

6.7. Conclusions

This chapter introduced the Matrix Cube, a representation for dense dynamic networks with weighted edges. We described how an interactive system, Cubix, is used to visualize such networks with the Matrix Cube and interactively deduces 2D views, including basic interaction support and animated transitions between view states.

The feedback we got from the two experts is encouraging: they both were able to understand the cube metaphor, interpret the views, chose views according to the current task, as well as to navigate between views. Cubix enabled them to make interesting observations about their data, in a way they were not able to see it before, as they stated during the study. Both experts expressed interest in using it further to explore additional data in the future.

However encouraging, our evaluation is only a first step and we need to further evaluate the potential and usefulness of the Matrix Cube as well as novel interface too exploit and explore it. A limitation of the study is that it only involved two domain experts. While they were from two very different domains (astronomy and neurology), Cubix should be evaluated with more experts and datasets, in order to validate the generalizability of our results. Controlled experiments are required, involving low-level network exploration and analysis tasks, to help identify particular strengths and weaknesses of Cubix and Matrix Cubes.

The design decisions made in Cubix are aimed at making the cube model and associated interactions simple to understand and consistent across views on the data, providing users with an effective interface metaphor. It can only serve as a basis for the many open questions, that should be addressed in future research on visual representation, interaction and analytical methods.

Cubix currently implements several visual mappings for cells, including color and size. However, projection views may require more powerful encodings. While currently cells overlap, glyphs such as used by Yi et al. (2010), as well as Brandes and Nick (2011), can show information about the projected dimension.

A major drawback of the cube model is that it does not provide a direct way to encode changing attributes on the nodes. One solution is to visualize node attributes inside cells when in the node-projection view, creating a heatmap as in NetVisia (Gove et al., 2011). A second solution, also requiring a projected view, consists in showing changes in node attributes as, for example, time lines next to the nodes' labels.

The Matrix Cube makes a link between a detailed analysis, including focused and precise tasks, and the general visual exploration. A simple extension in this direction is to provide multiple coordinated windows (Roberts, 2007) on the same cube, while each window can show different views and states thereof. While Cubix is currently designed for desktop computer setups, which rely on 2D screens, mouse and keyboard, future research could investigate appropriate exploration techniques that make use of beyond-desktop techniques; (multi) touch screens, large wall displays, stereoscopic displays, or interaction through physical props. The multitude of ways to look at the cube may inspire the design of appropriate interaction techniques.

While the current model focuses on undirected networks, it is technically easy to adapt it to visualize directed networks, as already explained in Section 6.1.3. However, a particular problem is the visualization of large networks. While the cube's size and visual complexity quickly increase with the number of nodes and times steps, the derived 2D views remain less affected by network size due to aggregation and clear layout of cells. Many options can be envisioned to further reduce the cube's size, informed by its geometry:

- **Pan+Zoom:** Show only a portion of the large cube, a technique described by Kraak (2003).
- **Collapse:** Filter out low-weighted edges and remove slices with no visible cells.
- **Aggregate:** Aggregate node slices into higher level slices using clustering, aggregating neighbors such as used in ZAME (Elmqvist, Do, et al., 2008), or aggregate adjacent time slices at different levels of temporal granularity (cf. Section 2.2).

7. A Framework for Unfolding Space-Time Cubes

Contents

VISUALIZING temporal data is a problem in many domains where data is rich and involves more than a single dimension. Networks are only one example where two spatial dimensions (node-link diagrams and matrices) are used, while visualizing evolution over time requires abstraction and changing the mapping to space as discussed in Section 2.6, for example, using techniques such as that described in the chapters on GraphDiaries (Chapter 4) and the comparison of weighted networks (Chapter 5). The particular visualization problems (combination of space and time, density, amount of change, types of change, etc.) are similar to, for example, visualizing geo-temporal information such as bush fires, earthquakes or people's movement over time, or analyzing and exploring large collections of videos.

Many different techniques have been designed to visualize such time varying data, including some of the oldest examples of information visualization, such as Minard's "Napoleon's march to Moscow" (Figure 7.1). Some techniques have become common across domains such as *animation* or *small multiples*. Other techniques use unique names that are easy-to-remember, such as *Parallel Edge Splatting*, *Massive Sequence Views* or *GraphFlow* (see Section 2.6). However, the term *small multiples* can refer to two different representations of the same data, depending on the meaning of the spatial dimensions within each picture (Figure 7.2). In contrast, the examples in Figure 7.3 have different names but are conceptually similar as they show time as the horizontal spatial dimension.

A missing common terminology for all these visualizations makes it hard to describe, as well as to think about and discuss them. A discussion of the advantages and disadvantages

Figure 7.1.: *Example of visualizing time and space: Napoleon's march to Moscow (Tufte, 1986)*

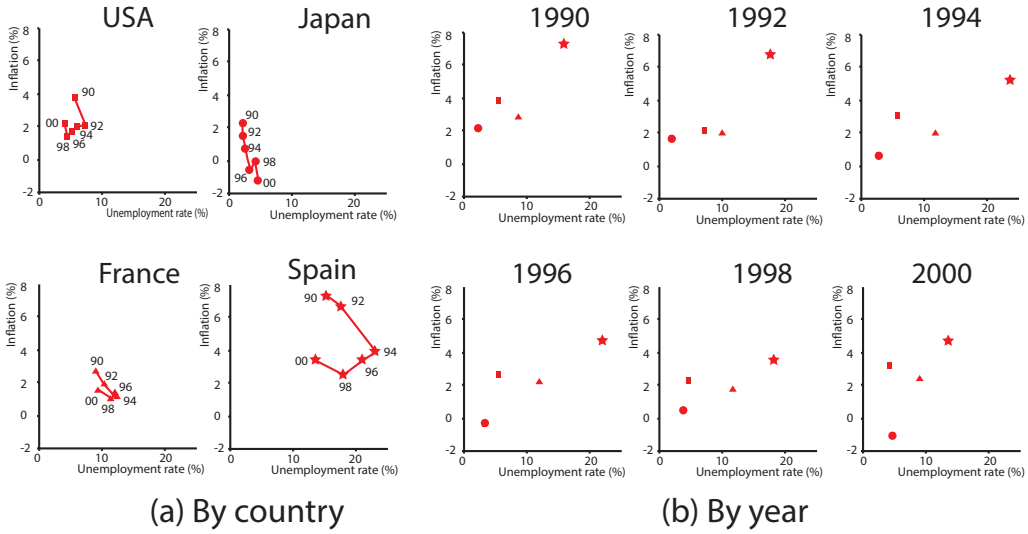


Figure 7.2.: *Small multiples used for two different views on the same data set; left: per country, right: per year.*

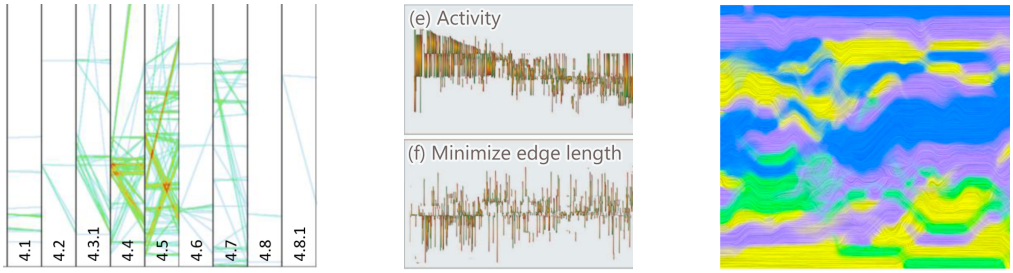


Figure 7.3.: *Examples of techniques with different names, but similar concepts to show time and topology. (a) Parallel Edge Splatting (Burch et al., 2011), (b) Massive Sequence Views (van den Elzen et al., 2013), and (c) Pixel based flow maps in GraphFlow (Cui et al., 2014)*

of techniques is important, since the current literature does not provide sufficient evidence and context to guide the application and design of new techniques (see Section 2.8).

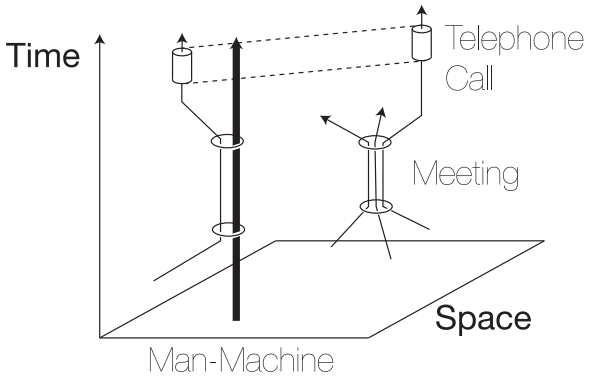


Figure 7.4.: *Illustration of space-time cube after*Hägerstrand *(1970). On the bottom the two spatial dimensions, while time proceeds to the top. Curves in the cube describe the trajectory of peoples, involved a group event "meeting" (right trajectories), or interacting with a machine (left trajectory). Telephone call is an event between distant peoples.*

The previous chapter on Matrix Cubes (Chapter 6) made an attempt to define a more structured design space to visualize and unfold dynamic network data, although with a very specific type of visualization and on a specific type of data. I introduced the notions of *operations* and *views* where operations are performed on the 3D space-time cube and lead to specific 2D views, each view supporting a complementary set of tasks (Chapter 6). Although the particular space-time cube appears to be domain-specific in the first place, its approach to visualization is generalizable to space-time cubes in other domains.

Motivated by the way Cubix uses the Matrix Cube to visualize dynamic networks, this chapter addresses the following questions:

- ▷ *How can operations on the space-time cube be described in a consistent design space?*
- ▷ *How can we create a common understanding of the space-time cube that allows for the evaluation of existing, and generation of novel, visualizations across domains?*

Space-time cubes originate from geography. They have first been described by Hägerstrand (1970) in order to provide a model to describe peoples' movement in space and time[1]. Individual peoples' *space-time paths* can reveal their interaction with other peoples as well as their range of action (*space-time-prism*). Figure 7.4 illustrates several peoples' space-time paths, some bundled in an event called "meeting". Geographical space is represented as a 2D plane, and time evolves towards the third spatial dimension to the top.

In this chapter, we present a framework in which the space-time cube serves as a *conceptual* representation for datasets that include a temporal plus a spatial component (*spatio-temporal*). "Spatial" does not need to refer to geographical data, but applies to all datasets that are usually visualized in two dimensions; dynamic networks, videos,

[1]Hägerstrand describes the space-time cube as to *"help us to develop a kind of socio economic web model"* (Hägerstrand, 1970, p. 10).

multivariate data in scatterplots, bar charts etc. By *conceptual* we mean that the space-time cube does not have to appear explicitly in the final visualization, nor does the cube or individual operations have to be implemented in a visualization system, as it is done in Cubix. Our model of the cube serves as a thinking tool with the purpose to:

- describe and classify operations,
- help in evaluating operations and visualizations, and
- serve as a design space to create novel visualizations.

To group techniques for dynamic networks in Section 2.6, I used the terms *aggregation*, *animated sequences*, *timeline*, *temporal multiples*, *embeddings*, and *ego networks*. All of these visualizations are *conceptually* related to space-time cubes. For example, *animated sequences* can be described as quickly showing time slices to users; *temporal multiples* can be described as laying out time slices side-by-side on the screen. Views in Cubix were already created in such a way (Chapter 6). Here, we use a new terminology, informed by *operations*, and review additional visualizations of spatio-temporal data from related visualization domains (Section 7.1). In general, while some views can be described by a single *elementary operation*, others require multiple operations, described as a *compound operation*. We group and describe elementary operations in a taxonomy and describe how they are assembled into compound operations in Section 7.2.

In order to discuss operations' appropriateness and allow for transfer across domains, we then analyse the different types of space-time cubes, as resulting from different characteristics of the data (Section 7.4). Independent from the actual type of data, space-time cubes can be described, for example, as *dense* or *sparse*, containing *few* or *many* changes. These characteristics are manifested as 3D structures inside the space-time cube and described in Section 7.3.

For any data set, we assume the space-time cube is given and we can discuss the appropriateness of our operations in creating 2D views from the 3D cube structure, based on (i) the the cube's inner structure, and (ii) on how they apply to the space-time cube.

7.1. Static Visualizations as Space-Time Cube Operations

In Section 2.6 of this dissertation I categorized dynamic network visualizations in seven groups: *temporal multiples, animated sequences, aggregations, nestings, timelines, ego-networks* and *space-time cubes*. This section shows how visualizations in each of these groups can be described by space-time cube operations, thereby extending each group with examples of visualizations for general temporal data. I also link the operations with those developed throughout this dissertation (Chapters 4, 5 and 6). We focus on a small but representative selection of examples from the literature, and describe operations informally.

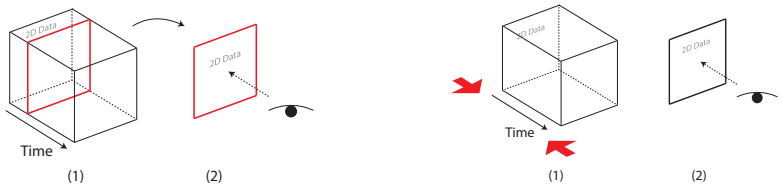


Figure 7.5.: *(a) Time cutting and (b) time flattening.*

Our conceptual space-time cube has three major axes: a *time axis*, and two orthogonal axes that we call *data axes*. The 2D plane formed by the two data axes is referred to as the *data plane*.

7.1.1. Time Cutting

The *time cutting* operation consists in extracting a particular temporal snapshot from the cube to be presented to the viewer. Figure 7.5(a) illustrates this operation: the left part (1) shows the initial space-time cube and the temporal snapshot that is being extracted, while the right part (2) shows the resulting image that is presented to the viewer.

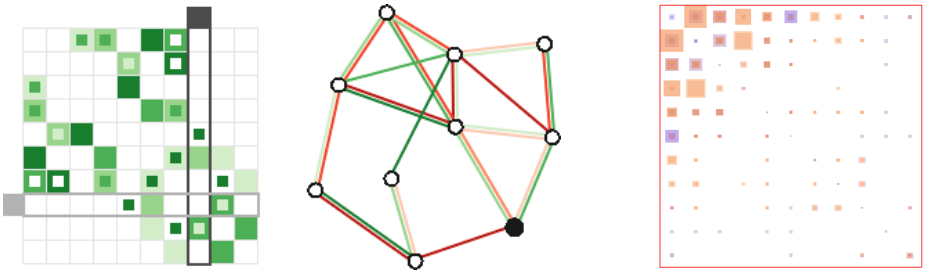
A time cut in network visualization corresponds to a single picture showing the state of the network at a particular time point, called a *time slice*, *snap shot* or *time cut*. Cubix implements time cutting as the user selects a single time slice and then switches to the time-projection view, or if already in the time-projection view, as the user hovers over a slice label. In GraphDiaries, a single selected time step is shown enlarged in the network view.

However, temporal visualization rarely consists in a single time slice. As we will see in Section 7.2, time cutting, typically, is either performed multiple times and used in combination with other operations, or is used in combination with animation and interaction. For example, in Cubix and GraphDiaries, users *interactively* navigate between time slices, while GraphDiaries shows *animated* transitions to convey changes. Animations are used in the majority of other graph visualization systems, already summarized in Section 2.6.2: Marey (Friedrich & Eades, 2001), SoNIA (Moody et al., 2005), Gephi (Bastian et al., 2009), Visone (Brandes & Wagner, 2003), and TempoVis (J.-W. Ahn et al., 2011).

7.1.2. Time Flattening

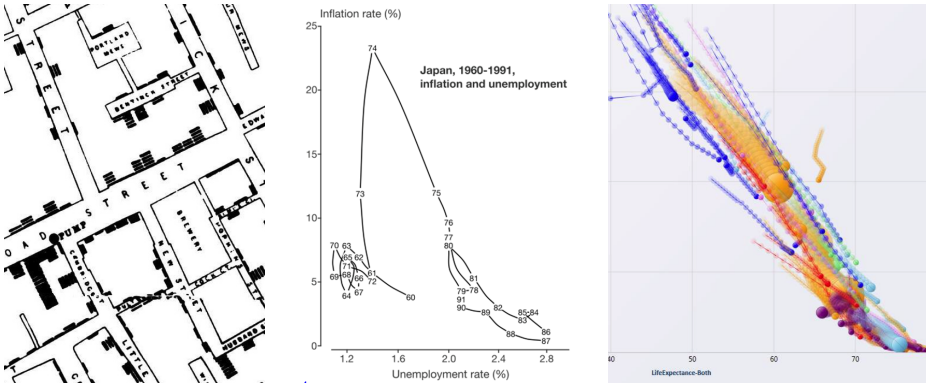
Time flattening projects the space-time cube along its time axis, merging all time slices into a single 2D image (Figure 7.5(b)).

Chapter 5 presented designs to compare two graphs using matrices (Figure 7.6(a)). In that particular case the space-time cube contained only two “time slices” and flattening



(a) Time Flattening for direct graph comparison (b) Time Flattening in Cubix

Figure 7.6.: *Examples of time flattening (a) the direct comparison of graphs, and (b) in Cubix.*



(a) (b) (c)

Figure 7.7.: *Examples of time flattening: (a) Detail of the map of the cholera outbreak in London 1854, by Dr. John Snow. Piled bars mark the number of death per house. (b) Another example of time flattening: scatterplot showing the relationship between inflation and unemployment in Japan from 1960 to 1991 (Tufte, 1990). Measures are connected by a line to indicate their order of time. (c) Scatterplot indicating development data for countries (Robertson et al., 2008).*

also required a *scaling* of one of the two matrices in order to obtain the cell patterns required for preattentive visual perception. Cubix provides the time-projection where time vectors superimpose in order to provide a legible picture of the network topology (Figure 7.6(b)). Different color encodings and translucency values for cells allow different information to be aggregated such as accumulated edge weight or time steps in which a node pair is connected (number of edges).

Other network visualizations also make usage of flattening, summarized in Section 2.6.3, and referred to as *"aggregations"* (Chi et al., 1998; Brandes et al., 2004; Andrews et al., 2009; van Ham et al., 2009; Gove et al., 2011; Hascoët & Dragicevic, 2012).

One of the earliest uses of time flattening to visualize information is Minard's illustration of Napoleon's march towards Moscow (Figure 7.1). The illustration shows on a

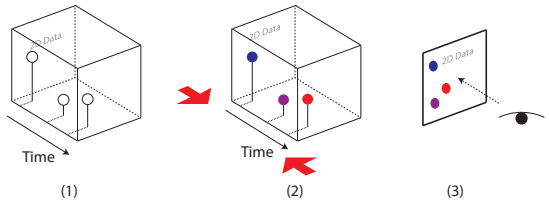


Figure 7.8.: *Colored time flattening.*

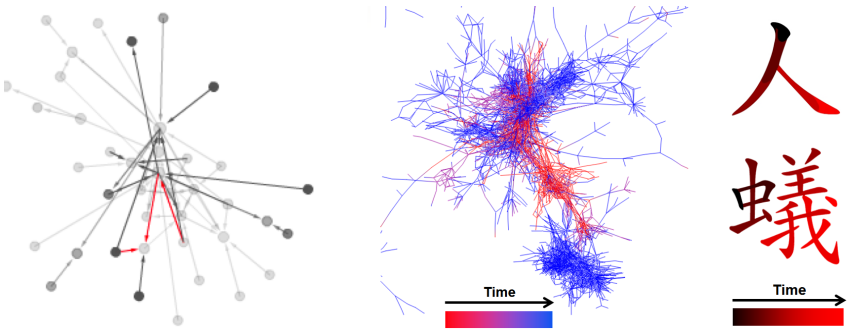


Figure 7.9.: *Examples of visualizations using* colored time flattening. *(a) Older nodes are faded to gray in TempoVis(J.-W. Ahn et al., 2011) (b) graph in Gevol (Collberg et al., 2003), and (c) stroke order in Chinese characters (Wikimedia, 2013). The color legends have been added.*

single image the state of Napoleon's army (position, size, key events) at different points in time during the Russian campaign in 1812 (Tufte, 1990). Another early example is Dr. John Snow's map showing where deaths from cholera occurred in London in 1854 (Figure 7.7(a)). Since the map shows multiple events aggregated over time (deaths), it could be seen as a time-flattened space-time cube.

Time flattening has been employed in a large variety of modern information visualization systems as well as static data graphics, and is not limited to geographical data. Figure 7.7(b), shows a Phillips Curve, i.e. a 2D plot of the relationship between unemployment and inflation in a country across several years (Tufte, 1990). This diagram can be seen as a time-flattened version of a dynamic 2D scatter plot. The scatterplot has a single data point whose coordinates are the two country indicators and evolve over time. Another example using scatterplots is used in a study by (Robertson et al., 2008), shown in Figure 7.7(c). Each colored point refers to a country in one year, while points of the same color indicate a country's *trace* though time.

7.1.3. Colored Time Flattening

The *colored time flattening* operation is similar to the time flattening operation, but time slices are assigned a color before being combined (Figure 7.8).

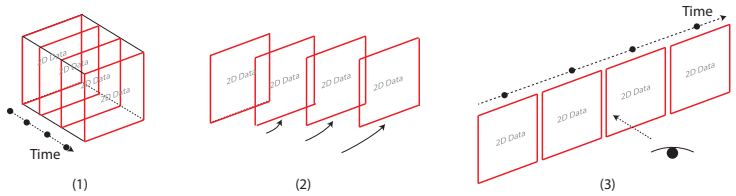


Figure 7.10.: *Time juxtaposing.*

Both flattening techniques in Figure 7.6 use coloring to differentiate between graphs. In Cubix, *coloring* is used independently from flattening; it is available in all views. Three examples of visualizations obtained by colored time flattening are shown in Figure 7.9: (a) TempoVis fades nodes to gray, which have been in the network for a long time (J.-W. Ahn et al., 2011), (b) Gevol where old links (in red) are distinguished from new links (in blue) (Collberg et al., 2003), (b) Chinese characters where first strokes (in black) are distinguished from later strokes (in red) (Wikimedia, 2013). Minard's map (Figure 7.1) also makes use of a simplified form of colored time flattening, since the army's forward march and return are distinguished using two different colors.

A special form of coloring is difference encoding, which has been used to compare two graphs (Andrews et al., 2009) or multiple graphs (Hascoët & Dragicevic, 2012). GraphDiaries (Chapter 4) includes color highlighting in the transitions, while Chapter 5 shows designs to compare networks with matrices.

7.1.4. Time Juxtaposing

Time juxtaposing consists in extracting multiple time slices from the space-time cube and then placing them side-by-side or on a grid (Figure 7.10).

Cubix and GraphDiaries, both support time juxtaposition. Other examples for networks as summarized in Section 2.6 (*Temporal Multiples*) are Matrix Flow (Perer & Sun, 2012), Flow Maps (Boyandin et al., 2012), GraphAEL (Erten et al., 2003), and Ploceus (Liu et al., 2011).

Beyond graph visualization, time juxtaposing is often used to show temporal data such as time-evolving maps and trajectories in space. Figure 7.11(a) shows forest harvest data over 11 years. Figure 7.11(b) shows a more complex example of time juxtaposing, where not the entire space, but a detail, is shown for each time step. Additionally, the example uses a flattening for each time step shown in order to visualize trajectories within one location. Time juxtaposing has also been widely used for video summarization (Truong & Venkatesh, 2007).

For spatio-temporal data, time juxtaposing is often referred to as *small multiples* (Collberg et al., 2003), although small multiples are not necessarily built from time slices, as for example demonstrated in Cubix, where multiple pictures are used to show node slices.

(a)

(b)

Figure 7.11.: *Time juxtaposing showing (a) approved forest harvest applications (Gretchen, 2011), and (b) traveling routes.*

(a) (b)

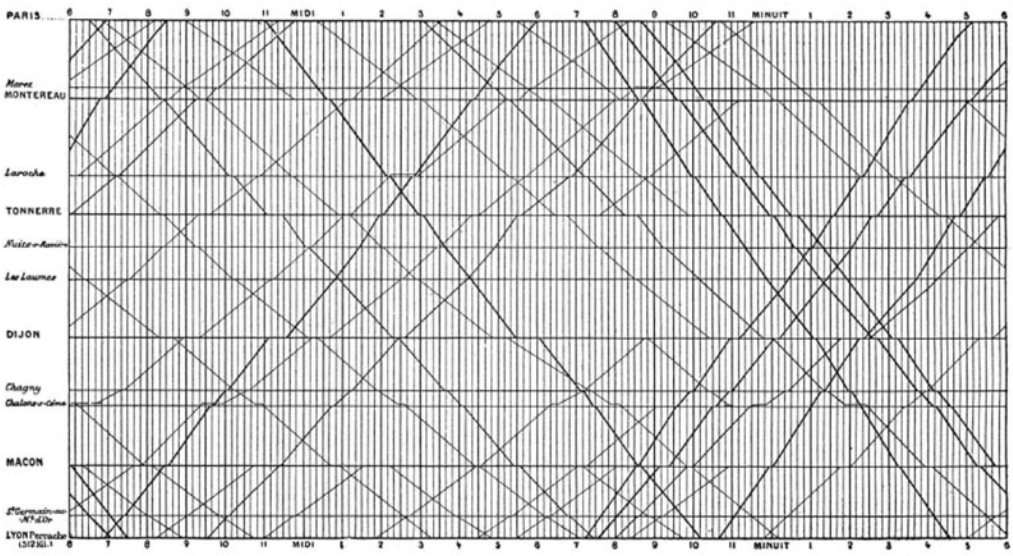
Figure 7.12.: Space cutting *and space flattening.*

7.1.5. Space Cutting

Space cutting consists in extracting a planar cut in a direction perpendicular to the data plane (Figure 7.12(a)).

Cubix allows space cutting as the user selects a single node slice in the 3D or node-projection view. A single node-slice shows the dynamic ego network of the corresponding node. While the Matrix Cube provides a geometry that allows for easy space cutting, a space cut in node-link diagram is less intuitive. The only space-time cube system for node-link diagrams that allows for multiple operations is the one described by Brandes et al. (2004) (see Section 2.6).

Space cutting has also been employed for visualizing temporal data. In the 19th century, Marey created a visualization of train connections between major french cities that employs space cutting (Figure 7.13(a)). Marey created a visualization using space cutting to visualize train connections between major french cities (Figure 7.13(a)). Space is cut along the rails connecting cities and diagonal lines indicate positions of trains at



(a)

(b)

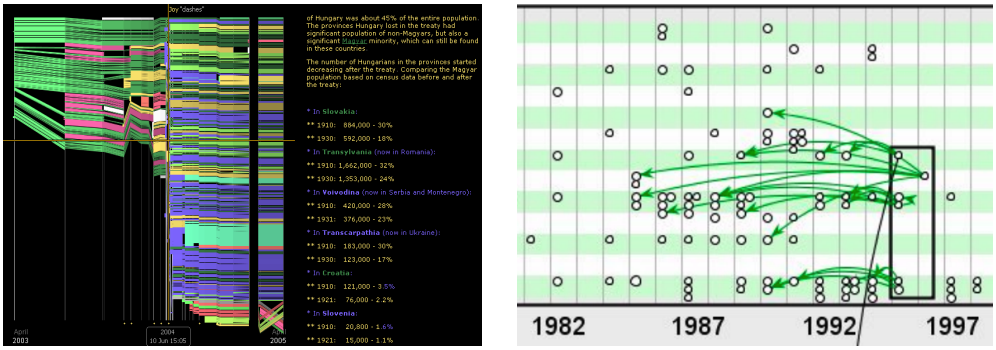
Figure 7.13.: *Example of space cutting: (a) horizontal lines indicate train stops, vertical lines indicate times, and diagonal lines indicate trains (Marey, 1878), (b) Space cutting used to show road traffic (Tang, Greenberg, & Fels, 2008).*

any time (Tufte, 1990; Marey, 1878). Again, the cut through space is not a simple plane, since it follows the course or the rails.

More recently, space cutting was shown to be useful for analyzing video logs (Tang et al., 2008): Figure 7.13(b) shows a space cut (called tear in the original work) extracted from a video scene, and revealing traffic activity (car count, speed and direction) on a road. The time slice at t_1 is shown to the left, together with the position of the segment extracted. The system is also able to show multiple longitudinal slices on top of each other (i.e. *space juxtaposing*).

7.1.6. Space Flattening

Space flattening is similar to space cutting, but involves projecting the cube along a particular direction on the data plane instead of extracting a cut (see Figure 7.12(b)).



(a) (b)

Figure 7.14.: *Examples for space flattening: (a) edit history of a Wikipedia article in HistoryFlow (Viégas, Wattenberg, & Dave, 2004), (b) space flattening showing temporal connection patterns in a network in semantic substrates (Shneiderman & Aris, 2006).*

Cubix uses space flattening analog to time flattening, in the node-projection view. Many network visualizations use space flattening, summarized in Section 2.6(*Timeline*): e.g., Semantic sustrates Shneiderman and Aris (2006) or parallel edge splatting (Burch et al., 2011). In none of the cases, the flattening is "perfect", i.e. additional effort was required to remove node overlap, which would have resulted from simply projecting a 2D node-link diagram. Space flattening for node-link diagrams turns into finding an optimal ordering along the spatial dimension orthogonal to the time axis.

An example of use of space flattening in infovis is the History Flow technique for visualizing document histories (Viégas et al., 2004), illustrated in Figure 7.14(a): the right panel shows the last revision of a Wikipedia article, each color corresponding to a specific contributor. The left visualization shows the history of the article, built by collapsing each article revision into a one-pixel column, and then displaying all columns side-by-side. These operations are equivalent to flattening the article's space-time cube along the x-data axis.

7.1.7. Sampling

Sampling is an operation that consists in extracting space cuts (samples) from a space-time cube at several locations on the data plane, then rotating those samples in-place so they face the viewer (Figure 7.15).

Cubix provides two ways of sampling; (a) tilting the cube (Figure 7.16(a)) or rotating slices (Figure 7.16(b)).

Two visualization techniques based on sampling were mentioned in this chapter's introduction (Figure 7.17). The left example shows the evolution of crime statistics in every US state (N. Andrienko & Andrienko, 2004), while the right example shows the evolution of high school population in several districts across three years (Shanbhag,

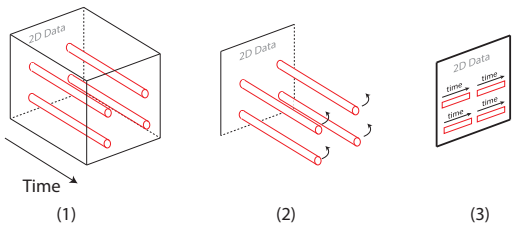


Figure 7.15.: *The* sampling *operation.*

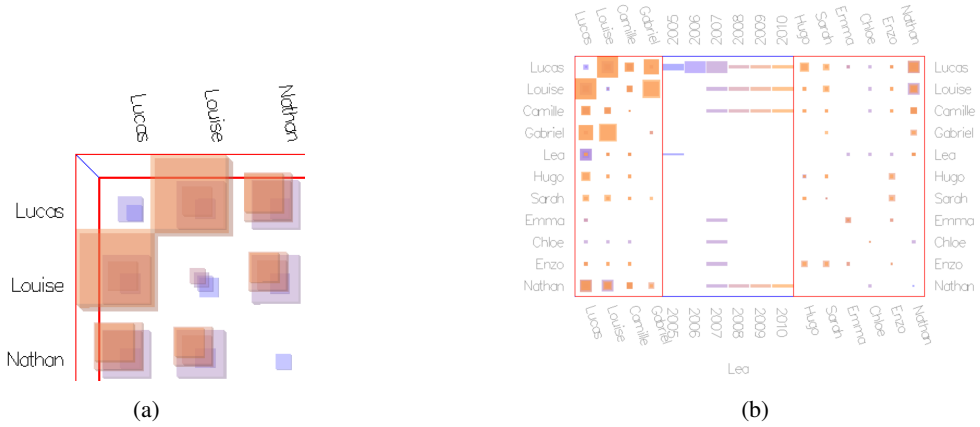


Figure 7.16.: *Sampling in Cubix, by (a) tilting the cube, and (b) rotating individual slices*

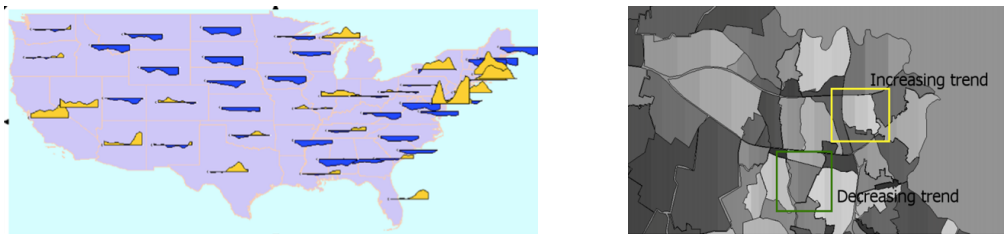


Figure 7.17.: *Examples of sampling: (a) the evolution of crime statistics in every US state; (b) the evolution of high school population in several districts across 3 years.*

Rheingans, & des Jardins, 2005). Although more complex operations are involved (e.g., using silhouette graphs to encode values), both examples are conceptually based on a sampling operation.

7.2. Taxonomy of Space-Time Cube Operations

Some operations are rather simple (e.g., *time cutting*), while others are more complex (e.g., *sampling*) and could be described as a combination of several lower-level operations. Yet, some particular techniques such as Semantic Substrates or Maray's train schedule,

require particular operations in addition to *space flattening*, in order to obtain the final 2D visualization.

In this section, we provide a more systematic description of the design space of space-time cube operations. This design space can be seen as an extension of the ones described for Cubix in Figure 6.4.

7.2.1. Basic Terminology

A *space-time operation* takes a space-time object and produces another space-time object. A *space-time object* is a geometrical object within a space-time coordinate system (i.e. two spatial dimensions and one temporal dimension). Possible space-time objects include, but are not limited to:

- *space-time volumes* (of which a complete space-time cube is an example),
- *space-time surfaces* (planar and non-planar),
- *space-time curves*,
- *points*, as well as
- sets of disconnected volumes, surfaces, curves and points.

The ultimate goal of space-time operations is to transform a space-time cube into a space-time object whose shape is compatible with the shape of the media employed to convey the information. By media we mean the visualization's *physical presentation*, which is the physical object or apparatus that makes a visualization observable to the end user (Jansen & Dragicevic, 2013). In the vast majority of cases (i.e. computer displays and paper) the media has a planar shape.

For a given media, a space-time operation is *complete* if it takes space-time volumes as input and produces space-time objects whose shape match the media's shape. Otherwise the operation is *incomplete*: it cannot be used to produce a valid visualization from a space-time cube. Several elementary space-time operations can be chained, in which case they form *compound operations*. A compound operation is complete if the first operation takes space-time volumes as input, and the last operation produces space-time objects whose shape is compatible with the media.

7.2.2. A Taxonomy of Elementary Space-Time Operations

A taxonomy of elementary space-time operations is shown in Figure 7.18, illustrated by small illustrations. The taxonomy breaks down space-time operations into five main classes:

1. **Extraction** consists in selecting a subset of a space-time object (e.g., extracting a line or cut from a volume),

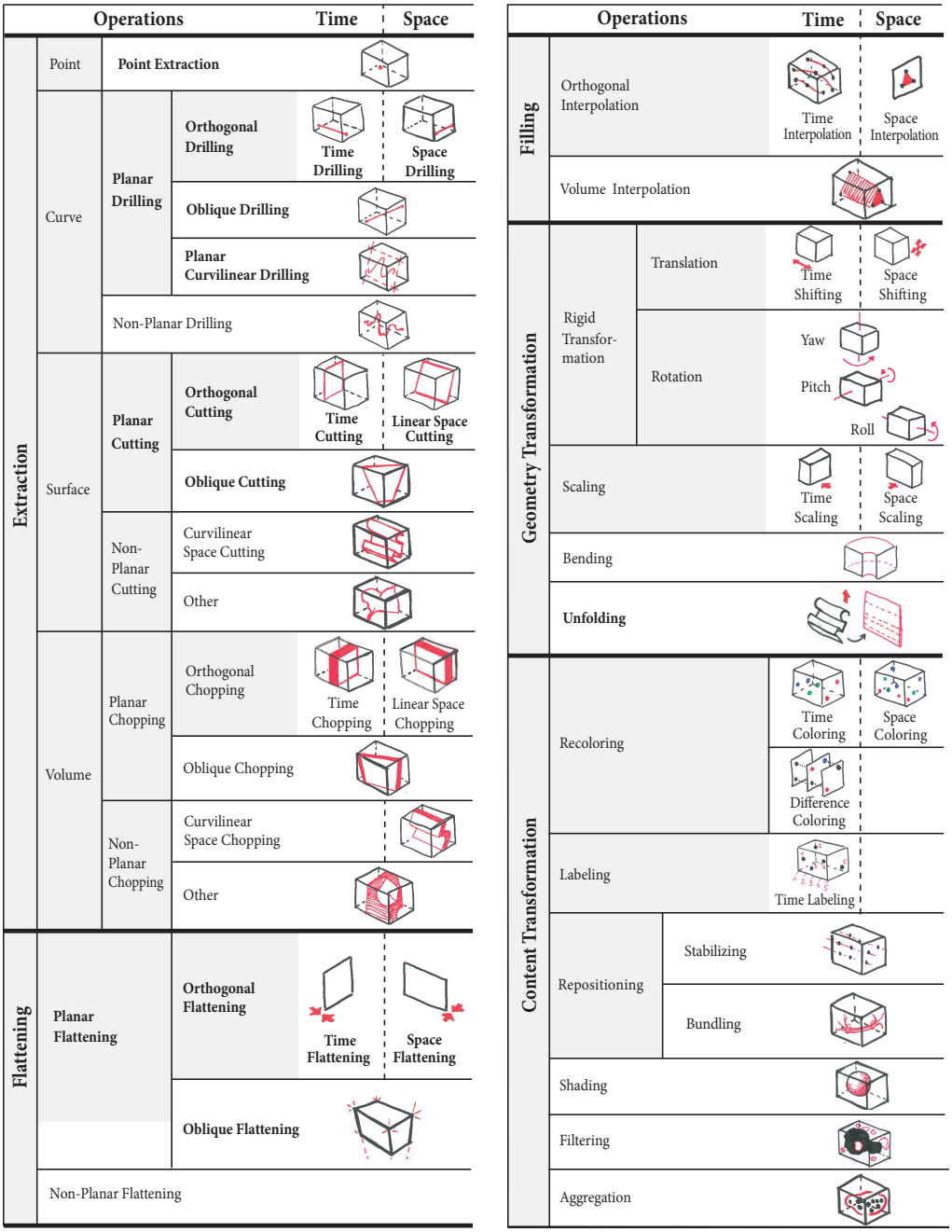


Figure 7.18.: *Taxonomy of elementary space-time operations with schematic illustrations. Gray shading indicate non-leaves, bold indicates complete operations. The* Time *column regroups operations that are applied according to the time axis, while the* Space *column regroups operations that are applied according to the data plane. Elementary operations can be performed in sequence, called* compound operations.

2. **Flattening** consists in aggregating a space-time object into a lower-dimensional space-time object (e.g., projecting a volume onto a surface),

3. **Filling** consists in turning a set of disconnected space-time objects into a fully connected space-time object,
4. **Geometry transformation** consists in transforming a space-time object spatially without change of content,
5. **Content transformation** consists in changing the content of a space-time object without affecting its geometry.

The table in Figure 7.18 shows how general operations break down into more specific operations. On each of the two columns, general operations are on the left while more specific operations are on the right. Operations that are the most specialized (i.e. leaves on the taxonomy tree) are shown on a white background. Operations written in bold are those which produce planar surfaces, i.e. that can be used as final operations on screen-based and paper-based media.

We quickly review the most specialized operations (white background), going from top to bottom on the left column, then on the right column. We also describe the *parameters* necessary to specify each space-time operation.

Extraction

- **Point extraction** consists in selecting a specific point inside a space-time volume. This operation is defined by a 2D position on the data plane and a time value.
- **Time drilling** consists in extracting a line parallel to the time axis. It is uniquely specified by a 2D position on the data plane.
- **Space drilling** extracts a line perpendicular to the time axis. It is specified by a 2D line and a time value.
- **Oblique drilling** consists in extracting an arbitrarily oriented straight line from within a space-time volume.
- **Planar curvilinear drilling** consists in extracting a planar 3D curve from a space-time volume. This operation, as well as all operations above, is complete for 2D media.
- **Non-planar curvilinear drilling** consists in extracting an arbitrary 3D curve from a space-time volume. It is incomplete, and hence needs to be combined with other operations like *flattening* or *unfolding*.
- **Time cutting** consists in extracting a planar cut from a space-time volume in a direction orthogonal to the time axis (cf. Section 7.1.1). It takes as parameter a time value that defines the cut position on the time axis. It is a complete operation for 2D media.
- **Linear space cutting** consists in extracting a planar cut from a space-time volume in a direction orthogonal to the data plane (see Section 7.1.5). It is also complete, and takes as parameter a line or a segment parallel to the data plane that, once extruded over time, defines the cutting surface.

- **Oblique cutting** consists in extracting a planar cut from a space-time volume that is neither orthogonal to the time axis, nor orthogonal to the data plane (Fels et al., 2000). It takes as parameter a 3D cutting plane.
- **Curvilinear space cutting** is similar to linear space cutting except the cutting surface is produced by extruding a curve parallel to the data plane that is neither a line nor a segment. This operation produces non-planar space-time surfaces that further need to be flattened or unfolded.
- **Time chopping** is similar to time cutting but slices have a thickness instead of being infinitely thin. Since it produces volumes, it is not complete for 2D media, and thus needs to be complemented with additional operations. It takes as parameter a time segment that defines the two cutting slabs (a slab is the infinite region between two planes).
- **Linear space chopping**, **oblique chopping** and **curvilinear space chopping** are similar to the previous cutting operations, with the difference that they produce volumes with a certain thickness instead of infinitely thin surfaces.

Flattening

- **Time flattening** aggregates a space-time volume into a plane orthogonal to the time axis (see Section 7.1.2). This operation takes as parameters a time value, a projection function and an aggregation function. The *projection function* maps 3D points to points on the plane. Examples include orthographic projection and perspective projection. The *aggregation function* describes how point values are combined. If values are defined in an RGBA color space, the function maps vectors of RGBA colors to a single RGBA color. Examples of such functions include alpha-blending (e.g., averaging all colors) and overplotting (i.e. only keeping the last color) (Collberg et al., 2003; Kapler & Wright, 2004; Robertson et al., 2008).
- **Space flattening**, **oblique flattening** and **non-planar flattening** are similar operations, but the surface on which the volume is projected is different (see previous cutting operations for more details) (Viégas et al., 2004).

Filling

- **Time interpolation** consists in filling "holes" in space-time objects (volumes, surfaces or curves) by interpolating between values *along the time axis*. It takes as parameter a monovariate interpolation function. For example, a piecewise linear time interpolation operation will transform a set of time slices into a full space-time cube by linearly interpolating the values (e.g., RGBA colors, positions) between pairs of successive time slices. For example, in order to provide fluent animations for dynamic networks, node positions are interplated (e.g., GraphDiaries).

- **Space interpolation** consists in filling "holes" in space-time objects by interpolating between values *on each data plane*. It takes as parameter a bivariate interpolation function. For example, a bilinear space interpolation operation will transform a set of lines parallel to the time axis into a full space-time cube.
- **Volume interpolation** consists in filling "holes" in space-time objects by interpolating *across both space and time*. It takes as parameter a trivariate interpolation function. One example is interpolating video frames using motion estimation techniques (Choi, Lee, & Ko, 2000).

Geometry Transformation

- **Space shifting**, **time shifting**, **yaw**, **roll** and **pitch** consist in moving or rotating space-time objects. They can be used, e.g., for placing multiple cuts side-by-side. They each take a single scalar value as parameter.
- **Time scaling** and **space scaling** rescale space-time objects along their principal axes. They take as parameters one and two scalar values respectively, that define the scaling factor.
- **Bending** deforms space-time objects. For example, a space-time volume can be bent such that the time axis follows an arc instead of a line (Daniel & Chen, 2003). This operation takes as parameter a deformation function that maps 3D locations to 3D locations.
- **Unfolding** transforms a non-planar space-time surface into a planar space-time surface. An analogy is a map projection function that transforms a sphere or portion of sphere into a plane. An example of space-time unfolding is Maray's train schedule (Figure 7.13(a)), which can be seen as an unfolded curvilinear space cut performed on a time-evolving 2D map.

Content Transformation

- **Time coloring** consists in altering the colors of each time slice according to time (Collberg et al., 2003; Erten et al., 2004). Examples include coloring each time slice uniformly according to a linear color scale (Figure 7.9), changing the hue of each time slice, or dividing the time axis in different regions and applying a discrete color scale (Figure 7.1).
- **Space coloring** alters the color of points in a space-time volume depending on their 2D position on the data plane.
- **Difference coloring** consists in altering the colors of each time slice according to the difference between time slices. One example is highlighting appearing nodes and disappearing nodes in a dynamic graph such as in GraphDiaries (Chapter 4) or Rufiange and McGuffin (2013); J.-W. Ahn et al. (2011).

- **Time labeling** consists in adding time labels to each time slice or to objects inside a space-time volume (Figure 7.7(b)).
- **Stabilizing** consists in repositioning objects on each data plane so that their trajectories are as parallel as possible to the time axis. Examples include computing stable layouts for dynamic networks, summarized in Section 2.7 (Erten et al., 2003; Diehl et al., 2001; Diehl & Görg, 2002) and stabilizing videos (Battiato, Gallo, Puglisi, & Scellato, 2007).
- **Bundling** consists in repositioning objects on each data plane in order to bring their trajectories closer to each other. One example is bundling air plane routes (Hurter, Ersoy, Fabrikant, Klein, & Telea, 2013).
- **Shading** consists in altering the color of a space-time volume's content by simulating light propagation mechanisms (e.g., diffusion, specular reflection, drop shadows).
- **Filtering** consists in removing parts of a space-time volume's content. One example is removing all points of a certain color or value as in Cubix (Chapter 6) or in Tardis (Carpendale et al., 1999) and V^3 (Daniel & Chen, 2003).
- **Aggregation** replaces multiple space-time objects by a single, larger space-time object. Different methods exist. For example, 3D kernel density estimation transforms a set of space-time points or space-time curves into 3D volumes or 2D (iso) surfaces (Demšar & Virrantaus, 2010).

7.2.3. Adaptive and Semantic Operations

Adaptive operations take into account the shape or content of the space-time objects they operate on. For example, an *adaptive time cutting* operation can cut cubes according to regions with large changes instead of cutting them into regularly-spaced slices. This technique is used, for example, in adaptive video fast-forward (Petrovic, Jojic, & Huang, 2005). Similarly, an unfolding operation that works on any surface (as opposed to, e.g., only spheres), would be an *adaptive unfolding* operation.

Semantic operations take into account the data semantics of the space-time objects they operate on. One example would be a *semantic volume interpolation* operation, which connects discrete sets of moving objects (e.g. in dynamic scatterplots (Rosling, 2006)) with lines or tubes (see Figure 7.7(b) for another example). This operation is semantic because it needs to know the *identity* of the objects to be able to match them on successive time slices. *Time labeling* operations such as the one used in Figure 7.7(b) are also semantic, because they need to know the location of datapoints of interest to place the labels appropriately. *Filtering* operations can also be semantic, as well as *recoloring* operations (Viégas et al., 2004; Robertson et al., 2008).

7.2.4. Compound Operations

We previously defined *compound operations* as several operations applied in sequence. According to our taxonomy from Figure 7.18, some of the operations we introduced in Section 7.1 are elementary, namely *time cutting*, *time flattening*, *space flattening*. Others are compound and can be broken down as indicated in Table 7.1. In our notation, the symbol $+$ refers to a composition, the symbol $*$ refers to operations applied multiple times and the symbols [ ] refer to optional operations.

Compound Operation	**Elementary Operations**
Discrete time flattening	time cutting* + time flattening
Colored time flattening	time coloring + time flattening
Time juxtaposing	(time cutting + space scaling + space shifting)* + time flattening
Marey's schedule	curvilinear space cutting + yaw + unfolding
Slit tears	(linear/curvilinear space cutting + yaw + [unfolding] + space shifting)*
Sampling	(time drilling + time scaling + yaw)*
3D rendering	[shading] + oblique flattening

Table 7.1.: *Compound operations decomposed.*

A compound operation inherits the parameters of its sub-operations. For example, a *discrete time flattening* operation is specified by a sequence of time values, as well as a projection function and an aggregation function. But in practice, most compound operations enforce constraints between their parameters. For example, all *space scalings* from a *time juxtaposing* operation are typically the same.

7.2.5. Dynamic Operations

So far we only considered operations (both elementary and compound) that transform a space-time cube into a static visual representation. On computer displays, operations can also be applied in a dynamic manner. Dynamic operations can involve either animation or interaction.

Animation

We refer to *animation* as the process of applying different operations on a space-time cube over time, or similarly, varying the parameters of an operation over time.

The most common form of animation consists in changing the position of a cutting operation over time, i.e. *animated time cutting*. This results in the space-time cube content being "played back". For example, if the space-time cube represents a visual scene like video surveillance data, synchronizing the motion of the slice with a clock

will result in a real-time playback of the original scene. When significant data is skipped during playback, the animation is closer to a *discrete time juxtaposing* operation, except slices are shown in sequence instead of being laid out side-by-side.

An animated time cutting operation can be preceded by a *filling* operation in order to produce smooth animated transitions. Examples for dynamic networks have been given in Section 2.6.2 and in (Chapter 4), but animated time cutting has also been used for scatterplots (Rosling, 2006; Robertson et al., 2008). Most of these examples can be described as *semantic volume interpolation + animated time cutting* operations. Animated time cutting can also be combined with other space-time operations such as *time flattening*. For example, Gapminder can combine scatterplot animations with static trails for points of interest (a *filtering + time flattening* operation) (Rosling, 2006).

While many animation techniques can be described as animated time cutting on *static* space-time cubes, more elaborate techniques require operations to be applied in real-time. For example, they system by Hurter et al. (2013) uses *animated time chopping* to animate a network over time while preserving temporal context information. At every animation frame, a *time flattening* is applied that produces colored trails and a dynamic *bundling* operation is applied that guarantees a continuous animation without jumping bundles (Hurter, Ersoy, & Telea, 2012).

Although *animated time cutting* and its many variants are the most common forms of animation, other animated operations exist. For example, *animated 3D rendering* can explain a transition between two space-time operations to a user such as used in Cubix when switching between views.

Interaction

Interaction is similar to animation except the changes in the space-time cube operations are under the user's control.

Consider *animated time cutting*: if the position of the cutting plane is controlled by the user, for example by dragging a slider, instead of being automatically moved, then the operation becomes *interactive time cutting*. A common implementation of interactive time cutting is the slider on a video player. As with animations, any operation can be made interactive.

GraphDiaries implements particular interaction techniques for *interactive time cutting*: (a) transitions are possible between non-adjacent time steps, i.e. time cuts are not traversed in the order of time (inter-time step navigation), (ii) transition type and duration are controlled by the user (intra-time step navigation).

In Cubix most of the operations are interactive. (i) *Time cutting*, as well as *space cutting* is performed and parameterized interactively when leafing through nodes (cf. Section 6.3.6). (ii) Interactive *time chopping* is supported by the time slider allowing a

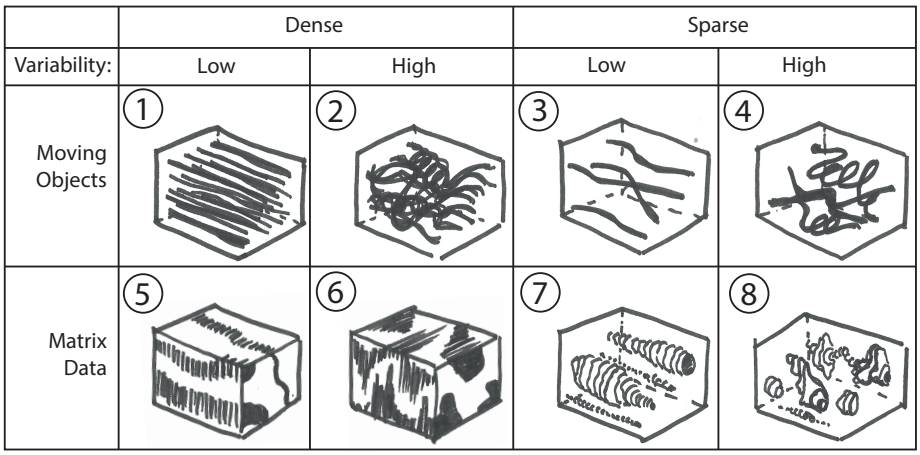


Figure 7.19.: *Different inner structures in a space-time cube.*

time window to be moved "through the cube". Further interactive operations in Cubix include (iii) the rotation of slices, a *compound operation* that consists of a *space* or *time cutting* + *yaw*, while (iv) tilting the Matrix Cube is a single *yaw* on the entire cube by a very small degree. Finally, Cubix allows for various (v) interactive filter mechanisms, summarized in Section 6.4.

7.3. The Inner Structure of Space-Time Cubes

So far we considered space-time cubes as monolithic 3-dimensional entities. However, two space-time cubes can look very different, depending on (a) the data that is visualized: migration of animals, earthquakes, changes in vegetation and ecosystems, dynamic networks, surveillance videos, or scatterplots, and (b) the data's characteristics. The way a space-time cube "looks" will be referred to as its *inner structure* and can be described independently from the actual data. As we discuss in Section 7.4, inner structure is an important factor when choosing and discussing space-time operations, because it influences each operation's efficiency.

7.3.1. Decomposing Space-Time Cubes

We assume that any space-time cube can be subdivided into lower-level space-time objects. A common example is a set of 3D curves or tubes generated by *moving objects* such as cars, persons, nodes in a network, or points in a scatterplot, illustrated in the first row in Figure 7.19 (In the following we use circled numbers to refer to the cases in Figure 7.19). These curves can yield different patterns depending on the data (①-④). A second typical structure is obtained from dense *matrix data* such as videos, temperature maps or Matrix Cubes (⑤-⑧). For these datasets, the object of interest (e.g., a pixel or a matrix cell)

is fixed in space but its attributes change through the time dimension. Again, different patterns of variation are possible such as between ⑤ and ⑥.

A space-time cube can often be subdivided in different ways. For example, pixels with an alpha channel can yield higher-level space-time objects with intervening spaces: ⑦ could be obtained by filtering a pixel-based landscape visualization according to the type of vegetation, while ⑧ could be obtained by isolating bush fires (Carpendale et al., 1999). Alternatively, optical flow can be extracted from a video, which would yield a structure more similar to cases ①-④. Conversely, any animated visualization could be turned into a video and yield a structure similar to cases ⑤ and ⑥. These are all different ways of looking at a space-time cube. Ideally, however, the focus should not be on the specific decomposition used, but rather on the overall properties of the inner structure.

7.3.2. Dimensions of Inner Structures

Figure 7.19 only shows eight possible examples of inner structures, defined by three properties: density, variability in position, and whether moving objects or matrix data is visualized. However, many more structures are possible. For example, moving objects can change in color or shape over time (Dwyer & Eades, 2002), or disappear and reappear. However, four dimensions are however enough to capture the most important properties of inner structures, namely: *density*, *variability in positions*, *variability in visual attributes*, and *object lifespan*. We review them here.

Density

The density of a space-time cube refers to how much non-empty space it contains. The inner structure of a space-time cube can be *dense* such as in cases ①②⑤⑥ or *sparse* such as in ③④⑦⑧.

Datasets like videos, and potentially Matrix Cubes (⑤⑥) produce inner structures that are maximally dense, and which will be referred to as *complete*. Dense and complete cubes are often transformed into sparser cubes (⑦⑧), by filtering elements (e.g., Cubix, Tardis (Carpendale et al., 1999)). Furthermore, a space-time cube can be *spatially dense* but *temporally sparse* (e.g., incomplete video data, or networks with a few dense time steps, cf. Figure 6.22), or vice versa (e.g., a GPS log over several days, a sparse network with many permanent connections). Density is a key property of inner structures, and it is the only dimension that is independent from the decomposition used.

Variability in positions

This dimension refers to the amount of motion that 2D objects undergo in the space-time cube, which in turn affects the shape of the 3D space-time objects that make up its inner structure.

For example, ③ (which could be four moving people) involves rather straight curves, while case ④ (which could be moving cars) produces more spaghetti-like curves. In networks, ④ uses an unstabilized layout, while ③ has been stabilized (cf. Section 2.7). Matrix data (⑤-⑧) yield zero variability in positions, since pixels are fixed and space-time objects are parallel with the time axis. However, using an optical flow decomposition, case ⑥ would exhibit more variability in position than case ⑤.

Variability in visual attributes

This dimension refers to the amount of changes on the objects' visual attributes, such as color and size.

For example the pixels in ⑤ and ⑥ (which could be pixels in a video or cells in a Matrix Cube) change in color, with a much higher variability in ⑥. In contrast, moving objects in ①-④ have fixed shape and color (black lines) and therefore exhibit no variability in visual attributes. There would be variability in visual attributes if, for example, objects were moving taxis and their color or radius would encode the number of customers they carry. Likewise, size and cells in time vectors in a Matrix Cube can exhibit changing edge weight over time as in ⑦ (see Figure 6.17(d)). ⑧ involves "blobs" moving over time, which can be bushfires or clusters in a network (cf. Figure 6.8(a)).

Object lifespan

Object lifespan refers to how long space-time objects persist, which has an influence on the portion of the time axis covered by the 3D space-time objects. The lifespan is maximum when all objects span the entire time axis. Regular matrix data also yields a maximum lifespan.

Objects with a long lifespan can be permanent connections in a network (e.g, trading flows) ⑤-⑦ or taxis moving in a city ①-④. In contrast, objects can reflect events that have a start and an end such as in earth quakes. Events without any duration produce the shortest possible lifespans, and yield inner structures consisting of points (Figure 7.20). Since objects do not persist, there is no variability in position or visual attributes.

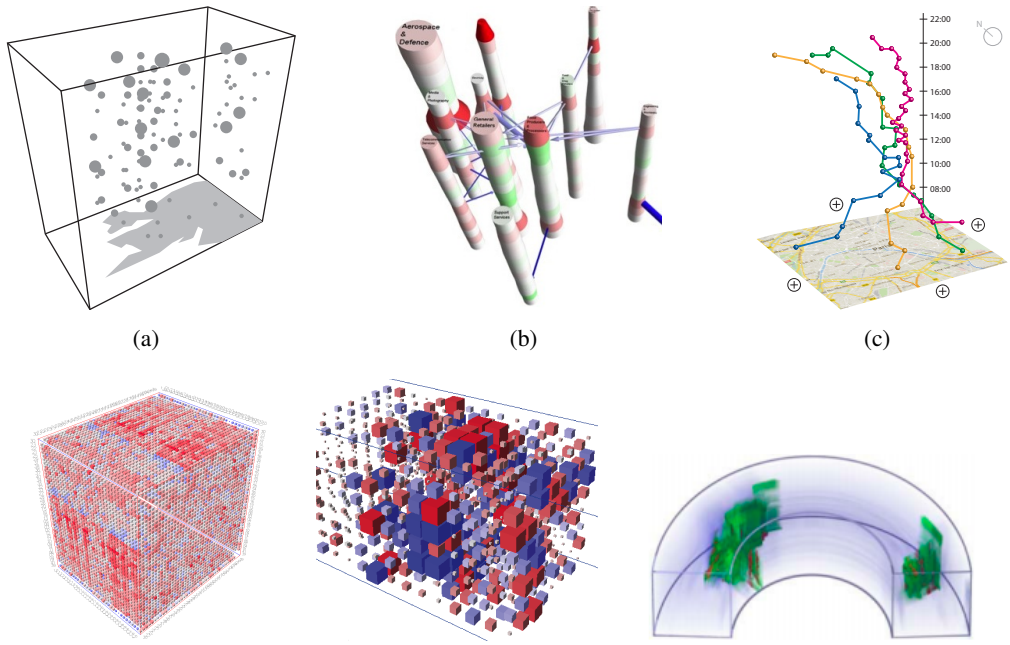


Figure 7.20.: *Examples of inner structures within space time cubes. (a) Earthquakes, illustration after N. Andrienko and Andrienko (2006), (b) Fund manager holdings (Dwyer & Eades, 2002), (c) Trajectories in GeoTime, illustration after Kapler and Wright (2004), (d) Signals between antennas (Cubix, Chapter 6), (e) Brain connectivity (Cubix, Chapter 6), (f) Surveillance video (Daniel & Chen, 2003).*

7.3.3. Examples of Space-Time Cubes

This section describes representative examples of space-time cubes, showing different data sets, created with different visualization systems. Yet, the cubes' inner structure can be described by the properties of the space-time objects.

Earthquakes The cube in Figure 7.20(a) is *sparse* and objects have *zero lifespan*. They further vary in size, but in no other visual attributes. The cube resembles ⑧ in Figure 7.19.

3D node-link diagram The dynamic network in Figure 7.20(b) shows nodes as columns and links as bridges between columns. The cube is relatively *sparse*, objects have different lifespans; nodes have *maximal lifespan* while links have *zero lifespan* and appear for a single instant only. Nodes can be described as moving objects. However, node positions have been *stabilized* and there is *no variability in position* but high *variability in visual attributes* (color, radius). The example resembles ①.

People's trajectories Figure 7.20(c) shows peoples' trajectories (*moving objects*), discriminated by color (*space coloring*). Objects also have maximal lifespan, do not

vary in visual attributes, and show high variability in position. While generally the cube appears *sparse*, its center is quite *dense*. The cube resembles ④, or ②.

Antenna communication in the Matrix Cube The data set in the Matrix Cube in Figure 7.20(d) is *complete* and resembles ⑤. The cube contains matrix data, i.e. there is *no variability in position*. However, cells in time vectors along the time dimension vary in size and color (edge weight), and since the cube is complete, time vectors have *maximal lifespan*.

Brain connectivity Figure 7.20(e) shows also a complete cube of location values with maximal lifespan and no variability in position, but with high variability in visual attributes. It resembles ⑥.

Surveillance video Figure 7.20(f) is an example of a video cube, but only changing pixels have been kept, the others have been removed (the cube shows two people passing a door). The video cube is made of pixels, hence shows *location values*. There is *relatively high* variability in visual attributes, resembling ⑧.

In the next section we will see why these characteristics matter, and how the inner structure of a space-time cube can be altered with space-time operations.

7.4. Which Operations to Chose?

So far, this chapter focused on the descriptive function of our framework: the different types of operations, how they can be combined and made dynamic, and the inner structure of space-time cubes. We now have all the elements to discuss the practical strengths and weaknesses of space-time operations. This section extends the previous discussions in this dissertation (mainly Sections 2.8 and 4.1) by using the framework introduced in this chapter and considering studies beyond the scope of dynamic networks. The discussion in this section aims to support the choice of operations when designing visualizations or researching novel visualization techniques to leverage the strengths and overcome the drawbacks of the space-time operations.

7.4.1. Empirical Evaluations

Studies related to dynamic network visualizations have been discussed in Chapter 2 (Section 2.8). These studies showed that *time-juxtaposing* is to be preferred for small networks with few time steps. Robertson et al. (2008) showed that animations performed least but were encouraging and fun to watch for participants.

In Chapter 4 we showed that *time interpolation* and *animated time cutting* significantly improve users' performance, compared to simple *time-cutting* without animation. *Differ-*

ence coloring and improved temporal navigation further improved users' performance as well as resulted in much more positive feedback. The tested data sets were much *larger* and *denser* than in previous studies. Object *lifetime* was short for some nodes, long for others.

In Chapter 5 we compared node-link diagrams with *time coloring* (each graph had an individual color) and matrices with *space scaling* and *space transformation*, both in combination with *time flattening*. Node-link diagrams represent the network using *moving objects*, while matrices represent the network using *location values*. Both representations were *stabilized*, i.e. node and edge positions were exactly the same in both networks. We found that matrices perform much better than node-link diagrams.

Beyond dynamic networks, studies have been conducted on data about *moving objects* as well as with location values with minimal lifespan (Kjellin, Pettersson, Seipel, & Lind, 2010). The studies majorly compared *time flattening* against *3D rendering* and sometimes *animated time cutting*. On a data set of four people moving through space, (Kristensson et al., 2009) compared *3D rendering* with *shading* against *time flattening*, while using *space coloring* (each person had an individual color). In summary, the results show that for simple tasks, *time flattening* performed better, while *3D rendering* was better for more complex tasks.

A study by Willems, van de Wetering, and van Wijk (2011) compared (non-interactive) animated time cutting with *3D rendering* and an enhanced version of *time flattening* described earlier by same authors (Willems, van de Wetering, & van Wijk, 2009). Results across three different tasks showed that overall, each technique was best for one task. The study lists many exceptions, depending on data *density*, and *variability in position*. *Animated time cutting* was reported to be the least robust to a *high variability in position*. For the number of objects, Willems et al. report that *3D rendering* was least scalable, while *animated time cutting* performed best.

Kveladze and Kraak (2012) conduced a participatory design session with experts in geography, analysing movement patterns in *3D renderings*. The study aimed in investigating proper designs for people's trajectories to encode attributes such as gender and income. Experts reported that *shading* and mapping attributes to trajectory thickness cluttered the visualization. Instead, they suggested *space coloring* in order to better distinguish people. The data contained about 203 trajectories of commuters in a single city and compared to *time flattening*, experts found *time flattening* (showing a map of the city) too overloaded. However, experts further reported that *3D rendering "helps in earlier stages of the research to get some hints about the data"*[2].

In the case of dynamic networks, only a small subset of all possible operations have been covered by the mentioned studies; *time juxtaposing*, *animated time cutting*, as well as *time flattening* and sometimes *3D renderings*. Tasks and implementation are different

[2] Kveladze & Kraak, 2012, p. 7

across studies, which makes it difficult to compare them. While study results are already diverging for the case of dynamic networks, many of the existing findings are hard to generalize beyond the domain, type of data, and operation parameters used in the stimuli.

Using our framework, with its ability to reduce complex techniques into elementary operations and describe the inner structure of space-time cubes, we now discuss operations on a general level, driven by general knowledge on visual perception and on common sense. Where appropriate, we refer to the listed studies throughout our discussion.

7.4.2. 3D Rendering

3D rendering ([shading] + oblique flattening) is the operation most commonly associated with the space-time cube concept. The major difficulties of 3D visualizations (Shneiderman, 2003) include *(i)* occlusion, *(ii)* depth ambiguity, *(iii)* perspective distortion, *(iv)* color distortion and *(v)* navigation difficulties. There are further discussed below. However, *3D rendering* can be useful for providing a general overview of a space-time cube's inner structure (density, variability and object lifespan). It is also the most effective approach when the goal is to have observers explicitly think in terms of a space-time cube by conveying the metaphor, and for an initial overview of the data (cf. Kveladze & Kraak, 2012).

Occlusion issues are due to the use of a *flattening* operation together with an *overplotting* aggregation function (i.e., near objects are drawn on top of far objects). The amount of occlusion depends on the *density* of the inner structure. Therefore, *3D rendering* should mostly be adapted to *sparse* structures (cf. Willems et al., 2011; Kveladze & Kraak, 2012). Additional space-time operations should be employed for visualizing *dense* (and sometimes *complete*) structures. An effective technique involves sparsifying the cube's structure using *interactive filtering*, such as in Cubix.

A related approach involves the use of *translucency filtering* or more generally, translucent space-time objects. However, translucency produces visual clutter without fully addressing the problem of occlusion, and we found few instances of this technique yielding legible results. Also, occlusion is an important depth cue (Wanger, Ferwerda, & Greenberg, 1992; Kjellin et al., 2010) that is mostly lost when translucency is employed. For similar reasons, we recommend against making the whole space-time cube translucent – or equivalently, using an *alpha blending* aggregation function in combination with 3D rendering. Elaborate interactive operations exist that can decompose a space-time cube using *extraction* and *rigid transformation* operations, and allow users to look inside (Cowperthwaite, Carpendale, & Fracchia, 1996).

Depth ambiguity is further caused by the use of a *flattening* operation in 3D space. In the real world, most depth ambiguities are resolved through stereoscopic vision and structure from motion (Todd, 2004; Wanger et al., 1992), also there is evidence that stereoscopic displays do not improve performance for point clouds (Kjellin et al., 2010).

Structure from motion is as powerful as stereoscopic vision (Todd, 2004), and this is why *3D rendering* is the most effective when the space-time cube is allowed to rotate through *animated* or *interactive 3D rendering*. Incidentally, being able to "look behind" space-time objects also mitigates occlusion problems. When interaction cannot be used (e.g., on paper media), static depth cues are important. One approach for structures with *no* or *very low variability in position* is to have space-time objects "touch" the ground. This is already the case for structures with maximum *object lifespans* such as in Figure 7.20(b). For short *object lifespans* like in Figure 7.20(a), it is advised to either add lines connecting objects to the ground (Kapler & Wright, 2004) or add drop shadows (Wanger et al., 1992; Turdukulov, Kraak, & Blok, 2007) by applying a *shading* operation with perpendicular lighting.

Perspective distortion is due to the use of a *perspective* projection function: far objects appear smaller than near objects, making it difficult to accurately compare sizes and lengths. This issue can be eliminated with the use of an *ortographic* projection function instead. However, doing so will eliminate an important depth cue and may produce depth reversals analogous to the necker cube illusion (Wanger et al., 1992). Again, this type of ambiguity can be alleviated by the use of other depth cues such as occlusion, contact and *shading*.

Color distortion is a side-effect of the *shading* operation, since it alters the color of the space-time objects. This is not a problem when color is not encoding information or when simple color scales are used as in Figure 7.20(b), but could be an issue when numerical data is encoded through brightness. Yet, Kveladze and Kraak (2012) suggest to use not too many visual encodings on space-time cube objects.

Navigation difficulties with 3D visualizations are due to the *oblique flattening* operation having too many degrees of freedom. One way to facilitate navigation is to add constraints, i.e., only let users manipulate *3D renderings* in ways that are essential to their task (Smith, Stuerzlinger, Salzman, Watson, & Buchanan, 2001; Shneiderman, 2003), or restrict interaction so as to allow only for rotation around the y-axis. Restricting users freedom of interactions was one of the design goals for Cubix, leading us to use fixed views with easy navigation between them.

Generally speaking, space-time cubes are a convenient conceptual tool for thinking about temporal visualizations, but explicitly showing them using *3D rendering* poses a number of problems and should be done only when necessary (e.g, for showing an overview or for explanatory purposes). Computer screens and paper are 2D media, and are optimized for 2D visualizations (Shneiderman, 2003). As we have already seen, many 2D solutions exist to show temporal data.

7.4.3. Time Flattening

Time flattening (Figure 7.5(b)) can be seen as a 2D technique, or as a specific form of *3D rendering* projecting orthogonally on the data plane, as implemented in Cubix. *Time flattening* essentially turns a space-time cube into a spatial visualization: each point on the resulting image shows the history of this point aggregated over time. As a result, spatial relationships and trends are preserved and are clearly visible, such as the path of Napoleon's army in Figure 7.1 or the distribution of data values in Figure 7.7(b). Any aggregation function can be used, including alpha-blending for showing averages, or more elaborate techniques for showing cumulative statistics (Figure 7.7(a)).

The major drawback of *time flattening* is that it reduces the time axis to a point (Figure 7.5(b)), so most temporal information is lost, including the ordering of events (cf. Willems et al., 2011), temporal intervals, and absolute timestamps. However, part of this information can sometimes be cognitively reconstructed from object trajectories. For example, when reading Napoleon's march (Figure 7.1), one knows that an army does not jump between random locations but progresses smoothly, so not all orderings of events are plausible. Knowing that an army generally shrinks in size is then enough to infer its direction of motion: eastward then westward.

For flattened trajectories to be legible and informative, the space-time cube needs to be *spatially sparse* (cf. Kveladze & Kraak, 2012), with long *object lifespans*, and exhibit a reasonable amount of *variability in positions*. If an object's color is not used to encode domain-specific attributes, *time coloring* (Figure 7.9) can be used. This operation has the additional advantage of explicitly encoding time. However, color is a poor visual attribute since the human visual system cannot discriminate between many colors (Conti, Ahamad, & Stasko, 2005), and no color scale exists that naturally maps to time. If time intervals (or object speeds) are more important than absolute times, a more effective approach is to change the color or shape of objects at regular time intervals.

Another prominent problem of time flattening becomes evident in the Matrix Cube. Since there is *no variability in position*, i.e. objects (time vectors) do not appear as traces, as for example, in scatterplots (Figure 7.7(c)). This makes it impossible to communicate when, or when not, a space-time cube object is present at a particular position, since all objects overlap equally. In Cubix, cells are still visible, if they vary in size (edge weight). Otherwise, the tilting operation allows a quick preview, if cells are not too big.

Discrete time flattening (*time cutting* + time flattening*) is a variant that can also clearly expose time intervals and object speeds. Since this technique can result in loss of information, it works best for structures that are *sparse in time* or *exhibit low variability*. The way time cuts are placed on the cube is important; equidistant cuts generally expose relative times, durations, and rates of change. When not all cuts are equally informative, non-equidistant cuts may be preferred, but at the cost of loosing most of this temporal

information. If cuts are chosen so that objects do not overlap, these objects become clearly visible.

Additional operations can be applied to a space-time cube in order to reintroduce missing time information. For example, the Phillips curve in Figure 7.7(b) uses *semantic volume interpolation* to connect points and show their ordering. Both Figure 7.7(b) and Figure 7.1 (for the army's return trip) employ *time labelling* to explicitly convey event timestamps. However, since time labelling increases the number of visual elements on the screen, it should be used with care and may be inappropriate if many visual objects are shown already (Figure 7.7(c)). Moreover, time labelling is less useful to relate object position at the same time, since this requires reading labels which does not happen preattentively. In contrast, color is a preattentive visual variable.

In summary, *time flattening* is an operation that produces easy-to-interpret visualizations: space simply maps to space. It is useful for showing data patterns aggregated over time. But since it collapses the time axis into a point, it is most powerful when it can leverage humans' ability to reconstruct time from trajectories. This often requires the use of additional space-time operations and a very careful visual design.

7.4.4. Space Flattening

Space flattening (Figure 7.12(b)) is another form of *orthogonal flattening* that does not suffer from the drawbacks of regular *3D rendering*. The orthogonal projection function creates a meaningful spatial encoding, since one axis maps to time while the other axis maps to space.

Space flattening is effective when both time and space are important, and there is a meaningful mapping from the 2D data plane to a 1D line. For example, in order to be meaningful, time flattening in node-link diagrams requires a technique for node overlap removal or any other technique that orders nodes along a line (cf. Section 2.6.5). However, it produces visualizations that are generally less easy to interpret when the data plane does not have a natural coordinates system (e.g., in a node-link diagram or on a map), but may work well in cases that visualize location values, such as Cubix.

7.4.5. Time Cutting

So far we covered the different types of *flattening* techniques. These techniques essentially aggregate the data in a space-time cube and are helpful for overview tasks, but fail at providing details. Among the *extraction* techniques, *time cutting* is by far the most common, although mostly used in combination with *time interpolation* and/or *interaction*. A single time cut provides a clean and fully detailed snapshot of a particular time stamp. This is useful if the cube is *spatially and temporally dense*, since the entire screen is available for visualization and only a single time is shown.

However, since the goal of a temporal visualization is to show information across time, more than one time cut must be shown. We already mentioned *discrete time flattening* as a possible approach, although it still suffers from the drawbacks of aggregation. Two aggregation-free alternatives are *time juxtaposing* and *animated time cutting*. We review them separately.

7.4.6. Time Juxtaposing

Time juxtaposing ({*time cutting* + *space scaling* + *space shifting*}* + *time flattening*) allows to show different time cuts simultaneously. Usually, *space shifting* follows the occidental convention of laying out time cuts either linearly from left to right, or on 2D from left to right then top to bottom (McCloud, 1994).

The main advantage of *time juxtaposing* over *flattening* is the legibility of individual time cuts due to the absence of occlusion, and the ability to compare distant time steps (cf. Slocum, Robert S. Sluter, Kessler, & Yoder, 2004; Farrugia & Quigley, 2011). But the need for selecting time cuts generally means a loss in temporal resolution. *Time juxtaposing* is therefore mostly recommended for space-time cubes that are both *spatially dense* but *temporally sparse*. However, the *space scaling* operation also has the effect of shrinking time cuts (cf. (Boyandin et al., 2012; Farrugia & Quigley, 2011)): the more cuts are shown, the less space – and therefore resolution – every cut has. The *time juxtaposing* operation therefore requires finding the best trade-off between spatial and temporal resolution.

A potentially serious drawback of *time juxtaposing* is that time cuts use inconsistent spatial coordinate systems, due to the *space shifting* operation. This can make it difficult or impossible to relate moving objects across time cuts (cf. Archambault et al., 2011a). Hence, tracking elements or understanding trends can be hard. However, if objects on a time cut can be distinguished by their visual appearance and have a consistent appearance across time (i.e., low *variability in visual attributes* and possibly *in position*), relating time cuts becomes possible (cf. (Boyandin et al., 2012)). With few objects and preattentive visual encodings (Ware, 2004), relating cuts can be effortless. If not, it may still require visual search, split attention, and cognitive effort. One way of alleviating this problem is through interaction, by adding brushing and linking support (Becker & Cleveland, 1987).

Like with *discrete time flattening*, the proper choice of time cuts is crucial with *time juxtaposing*, with the difference that granularity also affects spatial resolution. Again, equidistant cuts are better at preserving temporal information. But when not all cuts are equally informative, non-equidistant cuts (i.e., *semantic time cutting*) may be preferred. For example, when showing changes in territory, historical atlases and newspapers often employ this technique to emphasize different periods and historical events (Figure 7.21). Individual dates typically need to be added (i.e., *time labelling*) to reintroduce the missing time information.

Figure 7.21.: *Borders of countries on the territory of later Yugoslavia, in different years (*Kids Britannica*, 2014).*

7.4.7. Animated and Interactive Time Cutting

Animated time cutting inherits the benefits of *time cutting* compared to *flattening* in that no aggregation is done, time cuts can therefore be seen clearly, without occlusion. In contrast with *time juxtaposing*, *animated time cutting* does not suffer from the reduced spatial resolution due to *space scaling*: each time cut spans the whole visualization and show more objects.

Moreover, it benefits from a much higher temporal resolution. Although time cuts may still need to be selected (if, e.g., the animation's duration must be short), many of them can be typically shown (e.g., 60 in only one second on a 60Hz display). *Animated timer cutting* is therefore mostly recommended for *spatially and temporally dense* structures but *low or medium variability in position*. If variability is too high, *time interpolation* or *volume interpolation* need to be used to produce smooth animated transitions, such as studies in Chapter 4; changes can also be highlighted (e.g., through a *difference coloring* operation) in order to facilitate their detection and interpretation.

Animated time cutting can be extremely effective for revealing changes between successive cuts, as our visual system is equipped for preattentively detecting visual changes (Bartram, 1997). Ideally, objects' trajectories on the screen are smooth across time cuts and do no change abruptly (cf. (Willems et al., 2011)).

While comparative studies are missing, the study in Chapter 4 shows that animation helps to understand more complex changes occurring across sequences of several time cuts, under a range of *variabilities in position* and in *visual attributes*. Our perceptual ability to detect and interpret changes, however, requires the absence of visual discontinuities (e.g., flicker) between time cuts of interest (Rensink, 2002), and ideally, that successive time cuts are similar enough to be visually integrated and perceived as a motion (Feldman & Tremoulet, 2006).

One drawback of *animated time cutting* is that an animation takes time, especially if the structure is *temporally dense* (e.g., a surveillance video). Additionally, users have to watch the entire sequence of time cuts, to get an overview over the data, a fact that users

may complain about (cf. Archambault et al., 2011a; Robertson et al., 2008; Farrugia & Quigley, 2011).

As with previous techniques, *semantic time cutting* can be used to speed up the playback when not all time cuts are equally informative (Petrovic et al., 2005). A related drawback is that since time cuts are not shown simultaneously, they cannot be freely accessed (Tversky et al., 2002). This makes it difficult to carry out many types of exploratory tasks, such as comparing two non-successive cuts, or examining a cut in context with other cuts. To overcome this issue, we enabled navigation and transition between non-adjacent time steps in GraphDiaries. However, comparison is still limited to two time cuts.

Finally, both *animated* and *interactive time slicing* can only be used on dynamic displays, whereas all previous methods are compatible with static media such as newspapers.

The controversy about all these operations is far from being settled. Most likely, the respective advantages of both techniques depend not only on the task, but also on the type of dataset and on the specific operations used. For example, employing smooth transitions (an *interpolation* operation) or giving subjects control over animations (*interactive time cutting*) can make a difference. Also, different datasets may produce widely different *inner structures*, and dramatically influence the efficiency of both techniques. By clarifying these differences and providing a terminology to describe them, we hope that our framework will help design more informative user studies.

7.4.8. Stabilization

The *stabilization* operation can be used to complement several of the previously seen approaches. While *interpolation* allows to produce smooth transitions between predefined time cuts, *stabilization* allows to reduce *variability in position* when positions do not encode data. Although it may reduce the legibility of trajectories when using *time flattening*, this technique may facilitate object tracking tasks in *animated time cutting* and object mapping tasks in *time juxtaposing*. Stabilization also has an effect in *space flattening* and *space cutting*, where it keeps object positions stable on the y-axis. Stabilization is mostly used in dynamic graph drawing under the term “mental map preservation”, summarized in Section 2.7.

7.4.9. Other operations

We discussed the benefits and drawbacks of space-time operations commonly used for visualizing temporal data. As suggested by our framework (the elementary operation taxonomy in Figure 7.18 and the different ways to combine them), a possibly infinite variety of operations exist and it is impossible to analyze them all. It is likely that

among all possible operations, the least constrained ones such as *oblique* and *non-planar* operations, are the most difficult to read and to interpret.

In fact, often used approaches are particular in that they mostly involve orthogonal operations, especially along the time axis (i.e., *time flattening* and *time cutting*). One striking exception is *3D rendering*, which combines several complex operations to emulate how we perceive solid objects in the real world, and therefore produces familiar (but sometimes illegible) images.

However, it is also likely that the immense design space of space-time cube operations contains techniques that are hard to understand in the beginning and must be learned, but can yield unique benefits for specific temporal data analysis tasks that usefully complement standard approaches. In providing a framework for even more experimental techniques, we want to stress the importance of research in novel temporal visualization techniques. By discussing the multiple trade-offs involved in known techniques and explaining how these trade-offs originate from the intrinsic properties of elementary operations, we hope that our framework will be able to guide designers and accelerate future discoveries. Another promising line of work consists in supporting multiple space-time cube operations such as in Cubix, and as discussed in the next section.

7.5. Space-Time Cube Systems

We have shown that the choice of the best space-time operation depends on many factors and always involves tradeoffs: there is not a single "best" approach. In this section we review visualization systems that address this tradeoff by supporting multiple space-time operations, comparable to Cubix. Such systems almost invariably use *3D rendering* as an explicit representation of the space-time cube, both for showing an overview and for explaining how different operations relate. We call these systems *space-time cube systems*. Because these systems work by letting users switch between different operations and tune their parameters, *interaction* is a central feature.

7.5.1. Space-Time Cubes for Exploration

Hägerstrand's intention with the space-time cube was to provide a thinking and analysis tool, not envisioning actual graphical usage due to missing graphical capabilities (Kraak, 2003). About 30 years later, another geographer, Kraak (2003) described the potential of the space-time cube in a *viewing environment* with *"interaction, dynamics and alternative views"*[3]. These multiple coordinated views (Figure 7.22) comprise four main views. The *working view* (Figure 7.22 right) shows a region of the geographical space and a certain time span (creating a sub-cube). The *2d-view* (top left) shows a map with the projected

[3] Kraak, 2003, p.1991

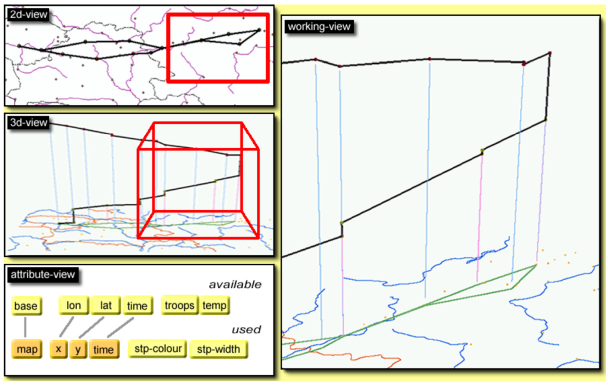


Figure 7.22.: *Interface for space-time cube exploration as described by Kraak (2003)*

trajectories, and the 3d-view (center left) shows all the data, much like an overview portal. Red boundaries indicate the location of the sub cube[4], shown on the right.

7.5.2. CommonGIS

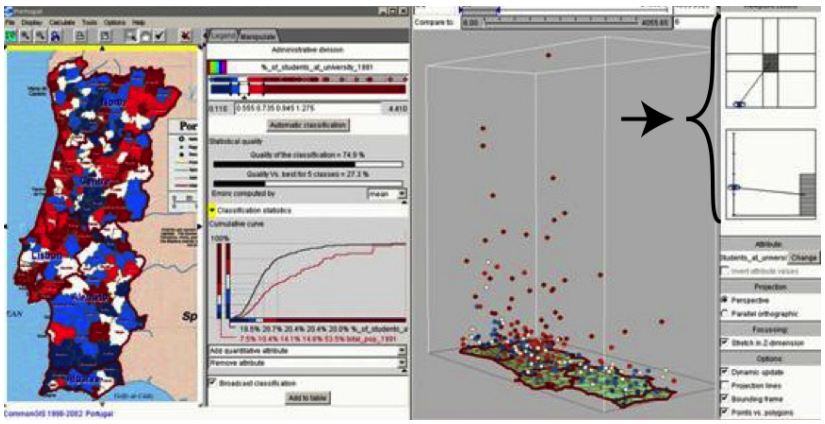
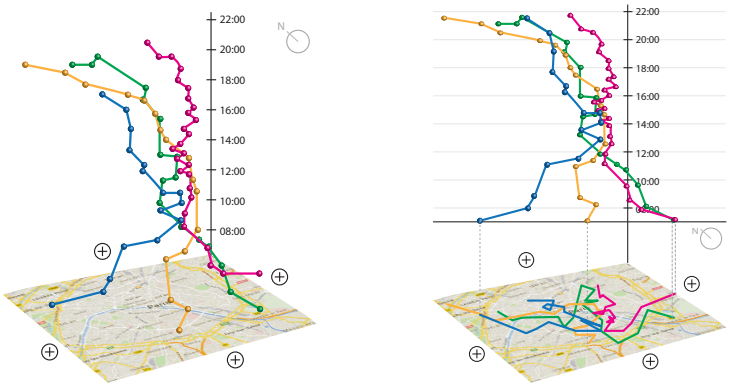


Figure 7.23.: *CommonGIS (G. Andrienko & Andrienko, 1999)*

CommonGIS (G. Andrienko & Andrienko, 1999) is a feature-rich analytical system for geotemporal data. It supports several space-time operations, including *time flattening* and *3D rendering* (Figure 7.23). The *3D rendering* view is combined with a *semantic filtering* operation to make the space-time cube *sparser*: geographical context is only shown on a single time slice as reference plane, once at the bottom of the 3D rendered cube, and as a time-flattened map. To control for the camera's position with respect to the space-time cube, two widgets allow to parameterize the *projection function* (indicated by an arrow in Figure 7.23). One widget controls the camera position around the cube, the other one controls its height.

[4]I define the term *sub cube* here, since Kraak did not provide a proper term for that concept.

7.5.3. GeoTime



(a) 3D rendering (b) Space flattening

Figure 7.24.: *GeoTime, illustrations after (Kapler & Wright, 2004)*

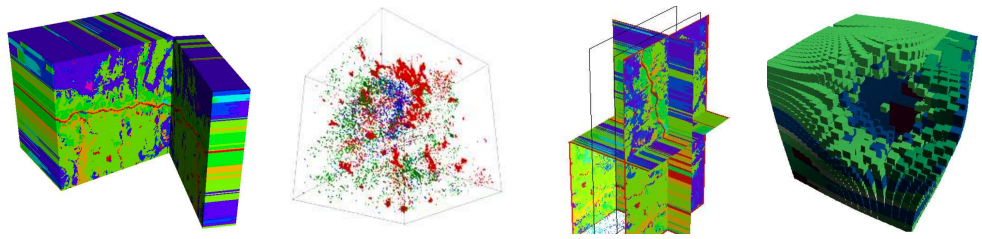
GeoTime is a carefully-designed commercial system for analyzing geotemporal data (Kapler & Wright, 2004). Events are shown as spheres on a *3D rendering* view that can be freely rotated (Figure 7.24(a)). This view also uses a reference plane, and a *semantic volume interpolation* operation is applied to indicate event ordering. Geotime implements Kraak's concept of detail (Kraak, 2003) as users zoom and pan in space and time. Dragging the timeline allows for interactive *time chopping* by defining size and position of *time chopping*. A zoom on the timeline results in *time scaling*. Similar, pan & zoom in space allows for *space chopping* + *space scaling*.

GeoTime further supports *time flattening* and *space flattening*. Figure 7.24(b) shows a *space flattening* view where time runs from top to bottom, and a reference plane is provided that can be rotated. This reference plane shows a space flattening. Events in both flattening views are connected through thin gray lines.

7.5.4. Tardis

Tardis (Carpendale et al., 1999; Carpendale, Fall, et al., 1996) is a system for visualizing environmental data using *3D rendering* in combination with advanced space-time operations. While CommonGIS and Geotime are made for the visualization of *movement data* and sparse cubes, Tardis aims are visualizing *matrix data* that is *complete* (Figure 7.25(a)). Voxels are color-coded depending on the type of vegetation, its age, soil characteristics or the presence of bush fires.

To sparsify the cube, Tardis implements *interactive semantic filtering*: users can, e.g., select a particular type of vegetation or bushfires (Figure 7.25(b)). In addition, Tardis supports *interactive orthogonal cutting*, but in contrast with our previous examples, cutting is always used in combination with *3D rendering*. Users can define and manipulate



(a) (b) (c) (d)

Figure 7.25.: *Examples of operations supported in Tardis (Carpendale et al., 1999); (a) time chopping + yaw rotation. (b) Time and Space Cutting in 3D Rendering, (c) Semantic filtering, (d) 3D fisheye distortion (Carpendale, Fall, Cowperthwaite, Fall, & Fracchia, 1996).*

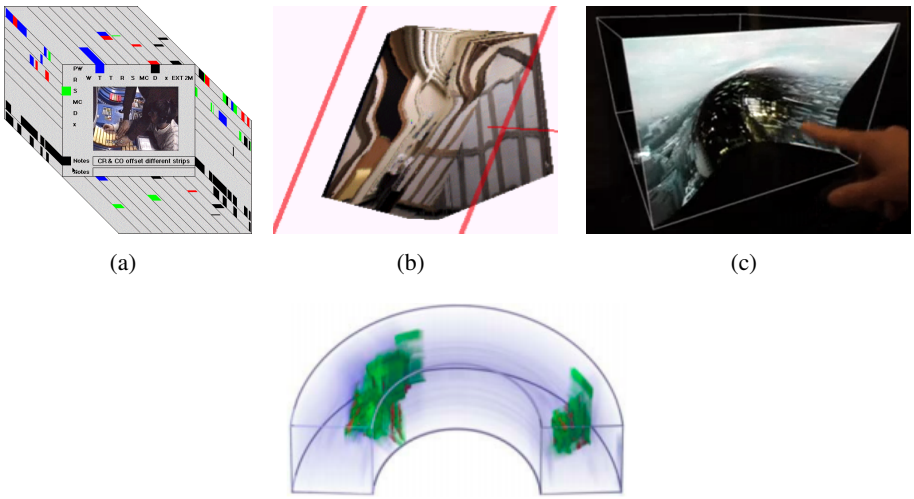
multiple orthogonal cutting planes, illustrated in (Figure 7.25(c)). Further operations include opening the cube like a book (*interactive (volume extraction + rotation)**) (Figure 7.25(a)) or apply a 3D fisheye effect (*interactive (volume extraction + translation)**). This fisheye effect, called "Visual Access Distortion" (Cowperthwaite et al., 1996), pushes away voxels from the cursor (Figure 7.25(d)) (Carpendale, Fall, et al., 1996) so as to make inner voxels visible.

In the inner structure of the data, Tardis is comparable to Cubix, and in fact employs similar operations. Interactions are all performed manually and no predefined views exist. Also, Tardis does not support operations such as *time juxtaposing* and *space juxtaposing* nor *flattening*.

7.5.5. Video Cube Systems

Space-time cubes for video analysis result from stacking all the video frames in their temporal order. As in Tardis, the resulting cubes are dense, and sophisticated algorithms have been developed to recognize and track moving objects, for example to analyse video surveillance material (Daniel & Chen, 2003; Botchen et al., 2008) and annotate videos (Mackay & Beaudouin-Lafon, 1998).

Diva displays an isometric *3D rendering*, while *interactive time cutting* allows for exploration of the individual video frames. Frames are annoted by colored tabs (Figure 7.26(a)) which remain visible at the cube's sides and indicate where frames have been annotated the same way. Video Cubism (Fels et al., 2000) uses a more unconventional way of exploration by *interactive oblique cutting*, shown in Figure 7.26(b). Similary, Khronos projector (Cassinelli & Ishikawa, 2005) manipulates the cube in a *non-planar cutting*, using touch and mid-air gestures (Figure 7.26(c)). By pointing into the video frame, the frame is "bended" into time, changing the shown time for parts of the video. V^3 (Volume Visualization for Videos) by Daniel and Chen (2003), supports different operations, including *time juxtaposing* and a *3D rendering* view that can be combined



(a) (b) (c)

(d)

Figure 7.26.: *Examples for space-time cubes used in video analysis. (a) Diva (Mackay & Beaudouin-Lafon, 1998), (a) Video Cubism (Fels et al., 2000), (c) Khronos Projector (Cassinelli & Ishikawa, 2005) (d) V^3 (Daniel & Chen, 2003).*

with a *bending* operation (Figure 7.26(d)). V^3 also supports *filtering* operations that allow to remove pixels of a certain color, or pixels that do not change across a given time period.

7.5.6. VISUAL-TimePAcTS

VISUAL-TimePAcTS is a system for analyzing activity diaries (Vrotsou et al., 2010). It uses non-geographical space-time cubes. The cube's two data axes can be mapped to data dimensions such as individuals, locations, or activities. We focus on the case where one axis maps to individuals while the other axis maps to activities. Activities are also encoded using color.

VISUAL-TimePAcTS supports *linear space flattening* on both data axes. Figure 7.27(a) shows 6 individuals (horizontal axis) and their activities (colors) across time (vertical axis). Figure 7.27(c) shows the evolution of activities aggregated across all people over time. VISUAL-TimePAcTS supports a seamless transition between the two operations through *interactive 3D rendering* (Figure 7.27(c)). Since 3D rendering employs orthographic projection and no shading, it is essentially an *oblique flattening* operation.

VISUAL-TimePAcTS supports a more elaborate space-time operation that prevents visual marks from overlapping due to *flattening*. In Figure 7.27(c), for example, individuals are horizontally offset when several of them do the same activity at the same time. This technique is similar to the tilt in Cubix and is called shearing by the authors, and is illustrated in Figures 7.27(d), 7.27(e). This technique is essentially a *(linear space cutting*

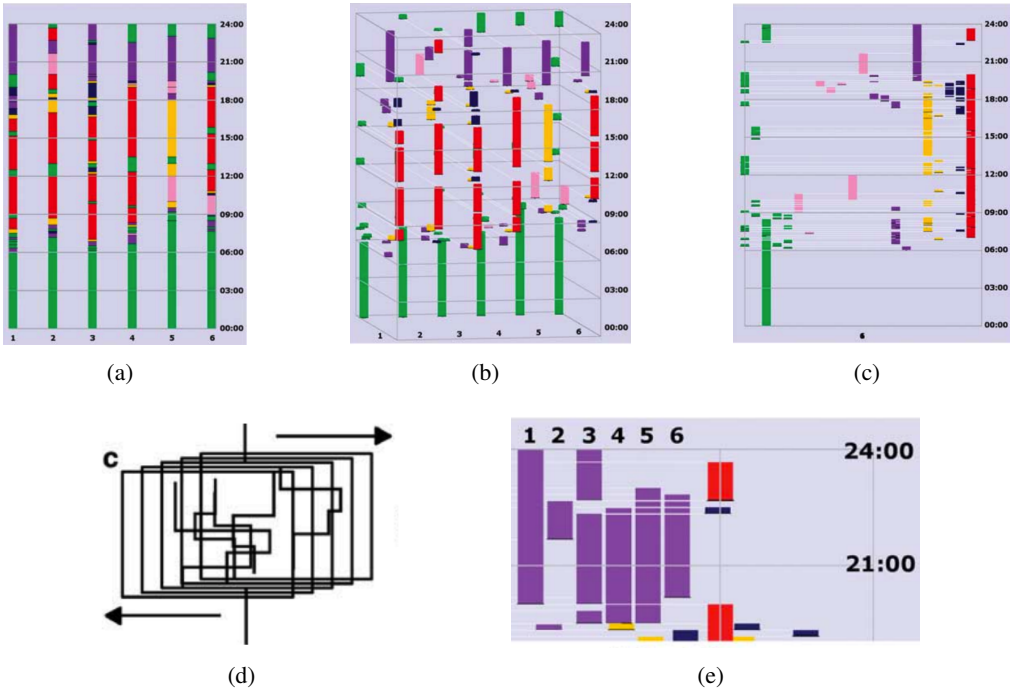


Figure 7.27.: *Views on the space-time cube in VISUAL TimePAcTS (Vrotsou, Forsell, & Cooper, 2010): (a) space Flattening on activities, (b) oblique flattening, (c) space flattening on individuals, (d) sharing illustrated, and (e) the result of shearing.*

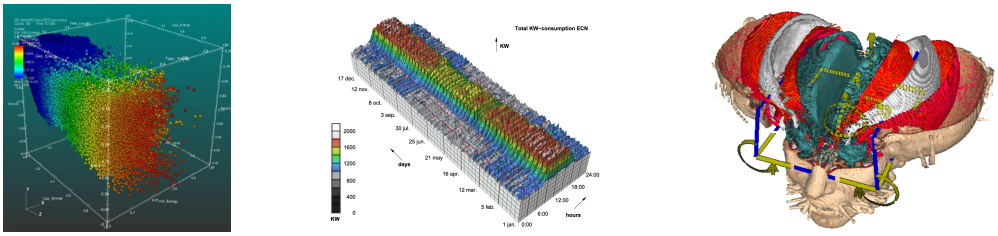
+ *space offset)** + *space flattening* operation, and is a hybrid between *space juxtaposing* and *space flattening*.

7.6. Discussion

This section briefly discusses our model with respect to the Information Pipeline by Card et al. (1999) in its ability to create visualizations. We also discusses its applicability to non-temporal data and possible limitations.

7.6.1. Information Visualization Pipeline

The most common software architecture for information visualization toolkits is the infovis pipeline (Card et al., 1999; Chi, 2000; Jansen & Dragicevic, 2013), already introduced in Chapter 1 and shown in Figure 1.2. As our framework builds on the notion of composition of operations, it can also be seen as a data-flow process, transforming a 3D cube into 2D visualizations. However, the way our operations map to the traditional infovis pipeline is not immediately obvious. Some space-time operations can be performed at different stages of the infovis pipeline. For example, *time flattening* can be done at the



(a) (b) (c)

Figure 7.28.: *Examples of 3D visualizations partially or not captured by our taxonomy. (a) 3D scatter plot (3D scatterplot, 2007), (b) Time-time cube (Van Wijk & Van Selow, 1999), and (c) operations on complex 3D data (McGuffin et al., 2003)*

data transformation stage, by aggregating raw data over time. Alternatively, it could be emulated by explicitly rendering a 3D space-time cube on the screen and using a proper camera placement and projection transformation. In that case, it would be implemented at the *rendering* level. However, both methods only give access to a very narrow range of operations.

7.6.2. Non-temporal Data

This chapter focused on temporal visualizations that involve two dimensions plus time. However, many other structured 3D visualizations exist, such as 3D scatterplots (Figure 7.28(a)) and bar charts. Van Wijk and Van Selow (1999) created a time-time cube, a type of space time cube but with two temporal axes and one axis to indicate values (Figure 7.28(b)). A single timeline is "folded" into that time-time space. Although our operations apply to 3D scatterplots and time-time space cubes with little modifications, there may be particular operations on those data sets, that are not captured by our taxonomy yet. For three-dimensional scientific data, McGuffin, Tancau, and Balakrishnan (2003) propose an extensive set of data driven decomposition methods (Figure 7.28(c)).

7.6.3. Limitations

Our framework is only a thinking tool. Like any model, it is necessarily incomplete. First, our taxonomy of elementary operations is not meant to cover all operations possible on space time cubes. Second, it does not provide much guidance for interaction design, i.e. how can operations be supported by interaction? Thereby, our framework is not meant to restrict creativity but rather to help visualization designers find new solutions, extend or generalize existing ones, and think out of the box. Only if we have a model to think about visualization, we can look for ways to overcome it.

Again, our framework assumes that the space-time cube already exists. It does not provide guidance for producing the space-time cube itself. Examples in this chapter—videos,

dynamic networks, geo-spatial data, scatterplots—are rather common to be represented as space-time cubes. For other temporal data, especially 3D spatial data (Baker & Bushell, 1995), creating a space time cube may not be so trivial.

Another issue we did not discuss is whether operations should duplicate data or should only aggregate and decompose space-time cubes. A case for not duplicating data has been made before (Hurter, Tissoires, & Conversy, 2009). At the same time, multiple coordinated views (Roberts, 2007) can help analyze complex datasets, shown in GraphDiaries. More research is needed to find a sensible trade-off.

7.7. Conclusion

This chapter presented a detailed analysis of techniques to visualize spatio-temporal data, by describing them as sequences of parametric operations applied to a conceptual space-time cube. Our operations are independent from the underlying data and can be applied across a range of application domains, be they geo-temporal visualization, dynamic networks, scatter plots, bar charts, or video visualizations.

By introducing domain-agnostic concepts, a clear terminology, and by extensively discussing trade-offs between known techniques, we aim at facilitating the comparison of different approaches to unfold and visualize temporal data. Existing visualizations from one data domain can be analyzed in terms of elementary operations and then be adapted to other domains and problems. Our model also aims at facilitating the design of user studies and discussing results in a more informative manner. Currently, the body of evidence is largely inconsistent due to the lack of a clear terminology and of low-level concepts that are necessary to tease out important factors. More controlled studies are needed to understand the trade-offs between different space-time operations, and how they perform depending on the task, the data, and the users.

Eventually, by giving a better vision of the richness of the design space, we hope our model will also motivate the exploration of new approaches. However, our framework also stresses the importance of developing fully-integrated interactive systems and toolkits that can support a range of techniques in a consistent manner. Which minimal set of views is necessary? How can multiple views be combined in a consistent way? It is important is to design effective navigation and interaction techniques to switch views as well as adapt and parameterize individual operations on the space time cube.

Finally, we have not involved tasks in our discussion of techniques. While a task taxonomy has been sketched in Chapter 3 of this thesis, mapping individual tasks to views has been done for views in Cubix in Section 6.3. Discussing tasks in the context of operations for general time-space cubes is important but is left as future work.

Acknowledgements *The work described in this chapter started as a presentation about ways to visualize temporal data, given by Pierre Dragicevic (INRIA) at the Dagstuhl seminar on temporal data visualization in Spring 2012. The presentation included some of the visualizations discussed in Section 7.1. An initial discussion in that topic involved Daniel Archambault (University of Swansea, UK), Sheelagh Carpendale (University of Calgary, Canada), and Christophe Hurter (ENAC, France). Within this joint-work, led by Pierre and myself, I summarized and completed many of the* unfolding *concepts discussed and implemented with Matrix Cubes as well as during the other projects in this dissertation. I contributed equally to all contributions described in this chapter, namely the operation design space, the definition of compound operations, the description of the 3D structures within the space-time cube, as well as the survey of techniques, and major parts of the discussion about techniques. Parts of this chapter have not been included in the submission yet. Discussion mainly took place between Pierre Dragicevic and I.*

8. Conclusion

THIS dissertation set out from the question: *"How to support the interactive visual exploration of dynamic networks"*. I explored approaches to—in a metaphorical sense—*unfold* dynamic networks, i.e. to look inside, to extract and re-lay out its parts, to re-organize, to leaf through, to superimpose, to project from 3D them to 2D space, making some intrinsic patterns visible. Unfolding is an iterative and interactive process, not only in terms of managing the interface but also for the user's mind; it involves posing questions, formulating hypothesis, confirming and rejecting hypothesis, making unexpected discoveries, and relate discoveries, gaining knowledge about the processes the data represents. Unfolding means to change perspectives, reveal the hidden and provide means to make the complex simple and visualizations can help mapping complex tasks to simple visual forms.

The individual chapters and contributions in this dissertation can be described as implementing four principles of unfolding dynamic networks:

- complementary views and techniques,
- interaction support to switch views,
- animated transitions for cognitive and perceptive support when switching views, as well as
- a consistent interface, possibly based on interface metaphors.

In this final chapter, I summarize the project in this dissertation by reviewing the individual contributions. This chapter concludes with an outline of future and open research questions.

8.1. Summary and Contributions

State-of-the-Art Review Chapter 2 first gave an overview of the notions and variety of dynamic networks and then presented a state-of-the-art in visualizing dynamic networks, using the following typology: *animated sequences, temporal aggregation, nestings, dynamic ego networks, temporal multiples, and space-time cubes.* We can think of these groups as general complementary views allowing to discuss individual visualizations' respective advantages and drawbacks, as well as how each visualization addresses particular drawbacks in its group (e.g. node ordering in timeline views, layout stabilization and change encoding in animations, encode time in aggregated views, etc).

Some studies on the effectiveness of techniques across groups exist, but the results are often hard to compare, due to that individual techniques, data, and task conditions differ. The overview in Chapter 2 demonstrated the variety of problems and approaches, and motivated the further work in the remainder of this dissertation.

Chapter 3: Towards a Task Taxonomy for Dynamic Networks Our task dimension defines the three main dimensions along which dynamic networks are explored: the elements of the network (WHERE), temporal units and their respective relationships (WHEN) as well as how network elements behave over time (WHAT). Simple (1-dimensional) tasks correspond to queries that consist of given values from two dimensions and search for the corresponding value in the third (open) dimension. The retrieved graph elements, time points or behaviors are usually further analyzed by tasks described, for example, by Amar et al. (2005). Higher-level tasks involve multiple dimensions and require to perform compound tasks, for example to *compare time steps* or to *describe the evolution of a cluster*.

Our taxonomy is a first step towards understanding *what* we should support with interactive visual exploration of dynamic networks. More expressive taxonomies can be imagined, capturing more tasks, reveal different groups of tasks, and result in different implications for visualization and interaction design. Rather than creating one large and complex taxonomy, which may be difficult to use and memorize by people, taxonomies could be specialized and be used in parallel.

Chapter 4: Animated Transitions and Temporal Navigation With GraphDiaries we provide ways to explore dynamic networks along the three dimensions of our taxonomy. Since many techniques support different tasks, there is no simple mapping from tasks to techniques. GraphDiaries unfold networks with animated transitions by allowing flexible temporal navigation; showing one time step allows to fully visualize network topology and its attributes, supporting tasks about WHERE (Figure 8.1(a)). Transitions between time steps (Figure 8.1(b)) highlight changes and provide visual clues to track elements (WHAT). Transitions are adjustable in what they show and between which time steps. Important for navigation is the time line with the small multiples and the navigation slider (Figure 8.1(c), WHEN).

For evaluating GraphDiaries we chose representative tasks and which involve comparing, tracking and observing compound graph elements (groups of nodes); size, number of permanent members, and amount of change. Each task represented and required to explore a different dimension in our taxonomy. Such experiments and tasks are harder to design, but they can yield more information about (i) which strategies users employ to explore dynamic networks, (ii) which techniques they use, and (iii) how useful a particular technique is under more realistic settings (ecological validity). We found that users overall preferred the extended navigation and visual feedback, and that our transitions increased

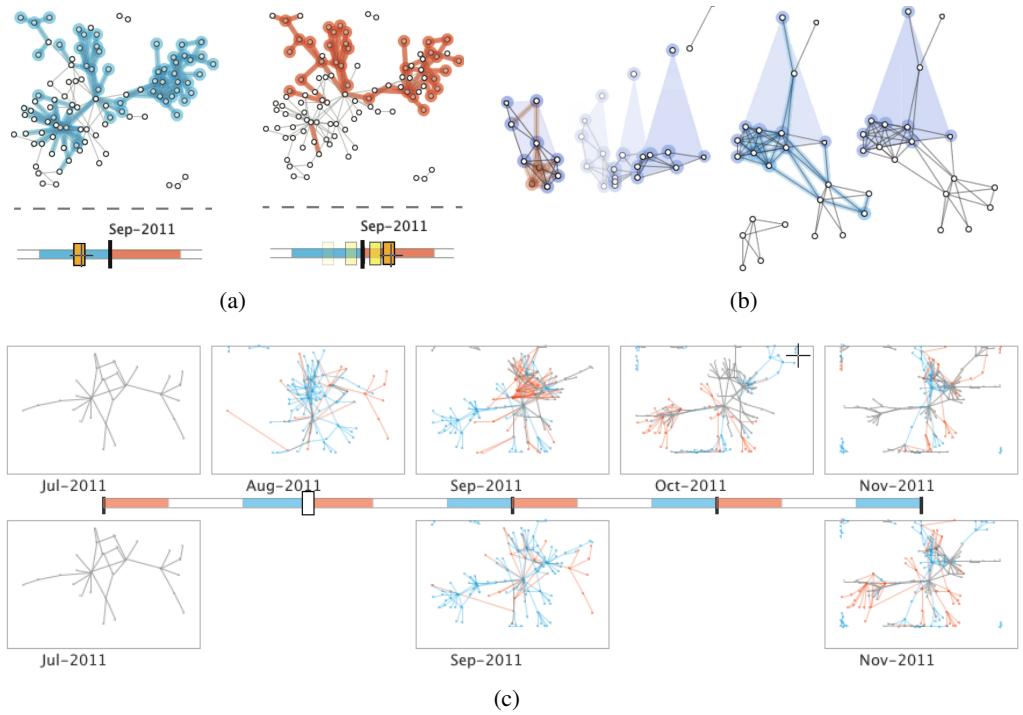


Figure 8.1.: *Visualizations and techniques in GraphDiaries (Chapter 4): (a) Change highlighting and slider navigation, (b) staged animated transitions and node query, and (c) temporal multiples.*

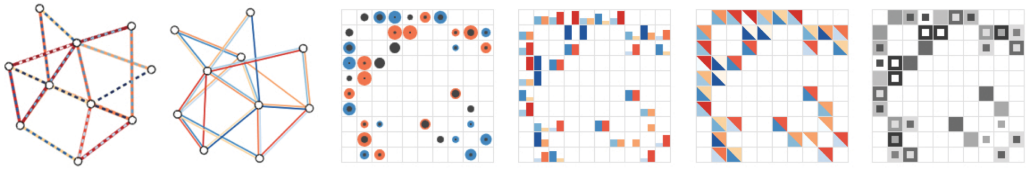
users' performance. From these results, we conjecture that in being able to support more complex tasks, GraphDiaries presents a useful approach to dynamic network exploration.

GraphDiaries can be extended in many ways including visual encoding of time relevant network measures and temporal aggregation of multiple time steps in the network view. Open questions include the visualization of a large number of small multiples on a limited screen space or to otherwise summarize the network over time in order to provide visual landmarks and previews to facilitate temporal navigation.

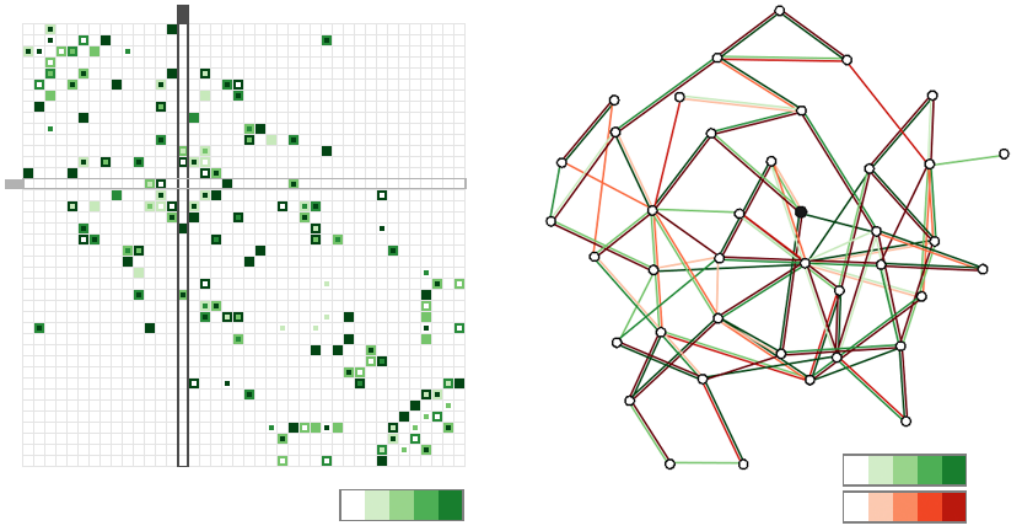
While in the process of publishing GraphDiaries, Rufiange and McGuffin (2013) presented DiffAni a system that combined similar techniques than GraphDiaries; integrating small multiples, interactive animations and difference views. We see this as evidence of the potential of providing ways to integrate multiple techniques in a consistent interface.

Upon multiple external requests, the research prototype of GraphDiaries is currently under preparation to be released in the upcoming months.

Chapter 5 Comparison of Dense Weighted Networks Density and changing weights of the connections are two aspects that make networks complex. In our use case about brain connectivity, analyzing the literature and interviewing neurologists revealed a list of tasks, many of which can be described as *graph comparison* tasks. Although to



(a)



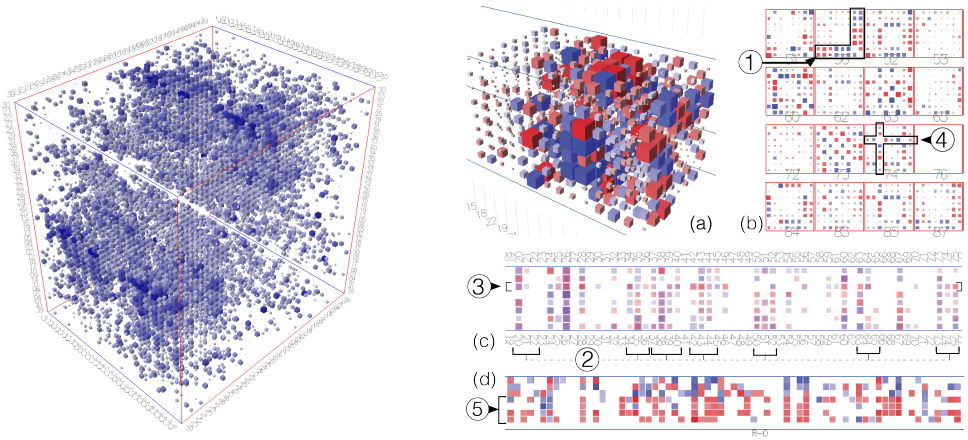
(b)

Figure 8.2.: *Designs for comparing two weighted graphs (Chapter 5): (a) Selected designs based on node-link and adjacency matrix representations, (b) condition from the experiment comparing the two best designs for both representations.*

a small extent, graph comparison tasks involve the three dimensions of our taxonomy. Techniques based on node-link diagrams fall short in both aspects of brain networks, as shown in our study (Figure 8.2(a)). For that study, we developed and discussed several designs for both node-link diagrams and adjacency matrices (Figure 8.2(b)). The success of matrices is likely due to their better capabilities to encode information in their cells, in comparison with lines in a node-link diagram.

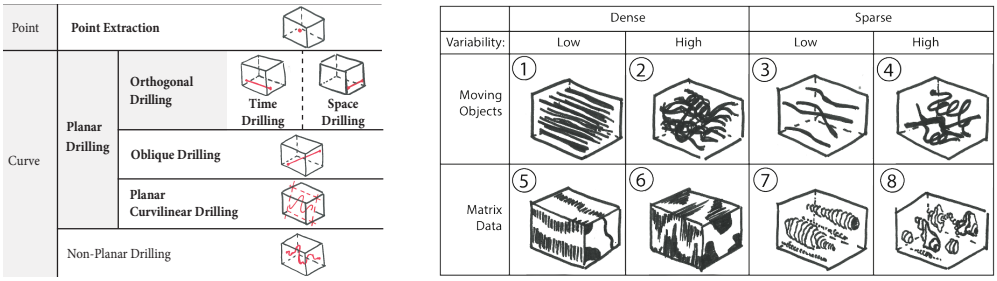
A particular problem in brain analysis is to represent the anatomical locations of the regions. It requires further designs and studies to provide legible representations of dynamic networks that require "spatialization", including geographical networks such as migration flows and transportation networks. An integration of our matrix design into GraphDiaries is straight forward and planned as a future extension.

Chapter 6: Unfolding Networks with the Matrix Cube A central model in this thesis was the space-time cube, serving as a base for interactive exploration. The Matrix Cube provides us with a consistent way of representing and exploring dynamic networks, especially dense ones and those with changing edge weight (Figure 8.3(a)). The cube's value results from its ability to apply operations to yield proper 2D views on the data,



(a) (b)

Figure 8.3.: *The Matrix Cube (Chapter 6). (a) 3D view showing an almost complete antenna signals network, (b) 3D and derived views showing a brain connectivity network.*



(a) (b)

Figure 8.4.: *Illustrations from the space-time cube framework (Chapter 7). (a) Selection of space-time cube operations, (b) selection of types of inner structures in space-time cubes*

while the 3D cube can serve as visual pivot and to communicate the metaphor to the user. Each way of looking at and *unfolding* the cube produces different visual patterns. It can be seen as direct translation of mapping tasks from data space (data in the cube and its visual encoding is the same in all views) into different visual tasks; here the user can (or must) decide which representation solves his task best (Figure 8.3(b)). Views are still complementary, since we cannot guarantee one optimal view per task.

The feedback we got from the experts we provided with Cubix was very positive and we continue on working with these experts and applying Matrix Cubes to their problems: for example, we want to investigate how we can represent multi-variate networks and allow for comparison of multiple dynamic networks with Matrix Cubes.

Chapter 7: A Framework for Unfolding Space-Time Cubes Cubix is a space-time cube system, since it implements the cube explicitly and allows users to apply

operations interactively. In order to explore the design space of operations and views on space-time cubes (e.g. consisting of node-link diagrams), we generalized that concept to other domains that use space-time cubes, such as video visualization and geo-temporal analysis. We first reviewed different 2D visualizations from those domains and proposed a structured and unified design space for visualizations, based on space-time cube operations (Figure 8.4(a)). We then described 3D structure within space-time cubes, which describe the data in domain-independent terms (density, change etc)(Figure 8.4(b)). Describing data is important to know which visualization is likely to yield good results, in other words, which visualization technique could fail. We showed that these data characteristics and the set of operations we described, are a good means to discuss visualizations of spatio-temporal data.

However, we were not yet able to properly discuss tasks in relationship to our framework. The dimensions of our task taxonomy are inherent in the cube; WHEN on the temporal dimension, WHERE in the spatial one and WHAT as the patterns inside the cube. Yet, since techniques support tasks, we can ask for which operation supports which queries and tasks. For example, queries define two dimensions, while requiring to search in the third one. Translated to the model of the cube, this results in *customizing* the cube:

- highlight time points according to the elements given in the WHEN dimension (e.g., slice extraction, time chopping, time coloring, time labeling),
- highlight graph elements given in the WHERE dimension (e.g., space cutting, unfolding, space coloring, stabilizing).
- highlight behavior over time and changes (e.g., filtering, time flattening (without stabilizing), difference coloring)

However, tasks are domain-dependent and even for dynamic networks, the discussion including tasks, techniques and data characteristics is far from being settled.

We hope our framework will help others to design novel visualizations, to improve existing ones, to compare and evaluate techniques, and most importantly to help them to *think* and to *communicate* about space-time cubes and how to *unfold* complex data. We finally hope to expand the range of operations as we describe more visualizations, new techniques get designed, and new data gets described with space-time cubes (e.g. propagation in networks, see Section 8.2).

8.2. Future Research

Descriptions of individual future research on each of the projects in this dissertation have been described in the last section and in the corresponding chapters. In this section I refer to which open problems remain and how results from this dissertation can contribute.

Visualize Propagation and Exchange The techniques described in this dissertation apply to the dynamics of changes in network topology and attributes. A widely under-explored area is the visualization of propagation and events, such as viruses, ideas, vehicles, messages, etc. Furthermore, topology and events interact in a way similar to rivers shaping their bed, while the bed defines where the river flows (adapted from Seoung, 2012). For example, brain connections strengthen where there are many signals. That means that both topology and events form the network and provide two ways of looking at a complex system. Space-time cubes intuitively represent spread in space and time, but *which particular (novel) operations are necessary? How do the visualizations described in Chapter 2 are useful? How should future visualizations be designed? Which tasks do users perform?*

Comparing Dynamic Networks Another under-explored domain in information visualization is the comparison of two and more dynamic networks (such as brain activity across individuals during an experiment). An obvious solution would be to treat networks as two disconnected components in the *same* network and use the existing techniques for dynamic networks visualization. However, comparing the evolution in different networks, possibly across different time spans, poses interesting challenges to visualizations and interaction design. Matrix Cubes as a concise way to visualize networks, perhaps can account for rapid comparison of dynamic networks. Operations could be applied synchronously to all cubes, while showing them side-by-side.

Compound Networks Dynamic Networks are aggregated along time, as users change temporal granularity. Temporal aggregation in a space-time cube is equivalent to the compound operation: *(time chopping + time flattening)**. Structural aggregation, i.e. grouping nodes into meta nodes and edges into meta edges, can be described as: *(space chopping + space flattening)**. Cubix can easily be extended to allow for aggregating node slices, yet there are many problems related to visualizing hierarchical networks that change over time.

Multivariate and Directed Dynamic Networks Multivariate dynamic networks with different node types, many time-varying attributes per node, and potentially different types of connectivity are difficult to visualize. *How can the evolution of node and edge attributes be shown at the same time? How to show changes on more than one type of connection between two nodes?*

Toolkit Support Toolkits exist for many types of networks, but little has been developed for dynamic networks. Perhaps because they are such a complex issue and have not yet been studied sufficiently, compared to large and multivariate networks. A toolkit for dynamic networks should provide appropriate data structures for large networks, as well as fast access. It should provide an API to pose queries and retrieve data, create and stabilize, the layouts, and it should be easily understood by programmers. According the

three types of storing networks, (see Section 2.3) I can imagine three complementary representations (or interfaces) to manage dynamic networks; a *topology-centric* that stores information per node and edge, a *time-centric*, storing information according to time (e.g. temporal snapshots), and an *event-centric*, storing individual change events. The appropriateness of each representation will depend on the data and visualizations to build with. The cube model and its operations, as well as the visualizations in this dissertation, could inform the set of queries that such a toolkit should support.

Mental Models Finally, can we think of other models to unfold dynamic networks, other than space-time cubes (a model very much stressed in this dissertation)? Does the notion of *operations* on consistent structures such as cubes, help us understand other problems in visualizing networks and design appropriate interfaces? Which other metaphors help us think about unfolding dynamic networks and to address the problems that cubes cannot solve in a straight manner?

◇

Appendices

A. Random Graph Generation for Evaluating Network Visualizations

WHILE evaluating our prototype for GraphDiaries we had to find appropriate data sets that would suite the needs of a controlled user study. Controlled experiments not only require controlled tasks, but also controlled data. Controlling data means to control characteristics of the data in order to make data samples comparable across evaluation conditions. Within the same experimental condition, data sets should be rather similar, yet across condition certain carefully designed variations are required, including variation size, or density. The number of required data samples can be very huge.

Real world networks are desirable for external validity, but finding networks that meet a set of desired characteristics is hard or impossible. Even if they do exist, data might not be available in sufficient quantity or variety, or may contain sensitive information requiring more than anonymization to be disclosed.

An often used alternative to real-world graphs is the generation of graphs using random graph generators. These generators allow to create graphs that satisfy certain characteristics but do not control for others. This limited flexibility may not be sufficient for formal evaluations of algorithms and layouts, for example if one wants to evaluate graphs with an upper bound on the node degrees or a given number of connected components.

Testing for higher-level tasks, on the other hand, requires yet a different approach to control data. Many higher-level tasks require higher-level graph patterns in the graph such as clusters of certain "shape" that resemble common patterns from the real world. Yet, domain and graph experts can best decide on the appropriateness of a synthetic data set when *seeing* it and comparing it to their knowledge about domain data.

In general, the problem does not only consist in finding the "right" graph generator, but finding the "right parameters" that lead to a certain desired graph. We describe the process of generating graphs as an interactive process that requires the computer to generate graphs, while experts control the generation interactively, based on immediate feedback of the machine's results.

Dynamic networks are a particular case for several reasons. (1) Changes in a graph are *not* easy to see (visualize) and hence it is not easy to judge whether the data set meets the desired requirements, (2) for almost the same reason tne graphs are hard to compare, for

example in terms of difficulty, and (3) dynamic networks are even harder to describe and quantify than static graphs, with respect to automatic generators and evaluation. Yet, we are interested in a general way to improve the generation of graphs, and also to reduce the problem; in this chapter, we focus on static networks. We explore and describe:

- ▷ *How to generate static graphs with particular measures and visual patterns?*
- ▷ *How the process of graph generation can be controlled interactively by the user, involving visualization for feedback?*

This chapter introduces *GraphCuisine*, an interactive system that lets users steer an Evolutionary Algorithm (EA) to create random graphs matching user-specified properties. GraphCuisine has been designed with three main scenarios in mind:

(i) **Generate graphs with specific graph measures** When designing a new layout, routing, or edge bundling algorithm, being able to test its behavior with varying topological properties, described by graph measures (e.g., density, diameter or clustering-coefficient) is essential and almost impossible with current generators.

(ii) **Generate graphs that "look like" a target graph** not knowing in advance the measures that contribute to the graph's structure. When testing the appropriateness of an algorithm for a specific application domain (e.g., network traffic, brain networks) with only a few samples of real graphs available, generating graphs similar to the samples is important. In that case, selecting the measures that best characterize the sample graphs requires visualization, human judgment, and interaction.

(iii) **Anonymize a specific graph** by generating a graph with similar or identical measures. When a company has confidential graphs to analyze (e.g., e-mail exchanges between employees or telephone calls between possible suspects), collaborating with researchers or specialists can be impossible. Generating graphs as similar as possible to the interesting confidential graphs makes collaborations much simpler.

To allow for these different modes of interactive random graph generation, GraphCuisine consists of an interactive interface and a graph generation model that combines different generators and optimizes their parameters with an EA. Graphs are created by traversing a pipeline of generators, each acting on the output of the previous one by adding or removing nodes and edges in very specific ways, according to the generators respective input parameters. The EA then attempts to find sets of parameters that produce graphs that best match the user-defined properties.

After a discussion of existing graph generators and an explanation of the basic concepts of evolutionary algorithms in Section A.1 Section A.2 introduces GraphCuisine and how it is used, while Section A.3 explains the particular evolutionary algorithm in GraphCuisine. We show how GraphCuisine is used to first *reproduce* and then *adapt* a real world social network in Section A.4. Section A.5 briefly explains how we use the graphs

generated with GraphCuisine to generate dynamic networks, and how we plan to extend GraphCuisine to generate actual dynamic networks.

A.1. Background

This section briefly reviews the foundations of graph generation and evolutionary algorithms. It then describes user interfaces for interactively generating synthetic data.

A.1.1. Random Graph Generation

The most simple random graph generator is the one from Erdös and Rényi (1959), where any pair of nodes is connected with a certain probability p. This generator allows to control only the graph's density, which for larger graphs is p. Generators have been developed using different techniques to construct graphs with varying characteristics. The most common classes are *preferential attachment*, which model "behaviour" of graph elements, and *structural generators*, which directly create certain structures.

Preferential attachment Preferential attachment generators iteratively add nodes to the network and connect them to existing nodes under a certain probability. This probability can depend on different conditions, such as node degree ((Barabási & Albert, 1999)) or distance in a lattice (Waxman, 1988; Kleinberg, 2000; Watts & Strogatz, 1998). While Barabási and Albert (1999) generate graphs with exponential degree distribution, Kleinberg (2000) as well as Watts and Strogatz (1998) generate small-world graphs. These generators do not introduce new connections into the graphs, but reconnect nodes (*re-wiring*).Generally preferential attachment is used to generate graphs with power-law node degree. Other techniques have been proposed to overcome drawbacks of preferential attachment and rewiring techniques, mainly to create networks with more realistic structures.

Structural generators Structural generators first create higher-level structures of the graph, such as clusters and recursively model details (Doar, 1996; Calvert, Doar, & Zegura, 1997). A problem common to most generators is that they focus on producing networks with one or two particular characteristics, disregarding all others. Consequently, techniques have been proposed that treat graph generation as a global *optimization problem* (Frank & Strauss, 1986; Carlson & Doyle, 1999) or using local *forces* between nodes and edges by evolving the network (Fabrikant, Koutsoupias, & Papadimitriou, 2002; Berger, Borgs, Chayes, D'Souza, & Kleinberg, 2004). R-Mat tries to model as many characteristics as possible in a flexible way by partitioning an adjacency matrix into cells, similar to BRITE (Medina, Matta, & Byers, 2000) and creates edges with different probabilities. The model features several graph characteristics and comes with an input parameter fitting function. However, R-Mat generates one graph at a time and

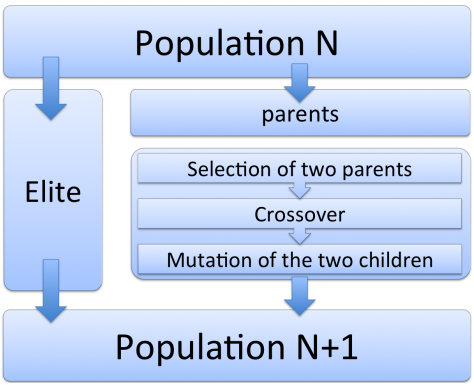


Figure A.1.: *Steps performed during one iteration of an evolutionary algorithm.*

only matches few graph properties. Again, one graph at a time slows down the exploration process.

Choosing the appropriate generator and modelling the desired graph characteristics is crucial in several contexts; the lack of expressiveness of existing generators impacts the generation quality or requires a long sequence of trials and errors to obtain graphs with a desired structure. For dynamic networks, there are even just very few generators, summarized in Section A.5.

A.1.2. Evolutionary Algorithms

Evolutionary algorithms (EA) are stochastic optimization heuristics that copy, in a very abstract manner, the principles of natural evolution that let a population of individuals be adapted to its environment (Goldberg, 1989). An EA considers populations of potential solutions exactly like a natural population of individuals that live, fight, and reproduce, and the natural environment pressure is replaced by an abstract optimization pressure. In this way, individuals which are the best ones, with respect to the problem to be solved, reproduce.

Figure A.1 summarizes the main steps of the iterative process of an EA. Each one is controlled by parameters and specific algorithms. The individual steps are as follows:

1. A set of individuals is selected from a population, called *parents*. The size of the set can vary, as well as the criteria for selecting parents.
2. Information about individuals (solution) is encoded in *chromosomes*, an array of parameters and their values. Similar to nature, in a *crossover* process, chromosomes of two individuals are combined to form a new chromosome, that of the *child*.
3. Finally, the child chromosome is *mutated*, i.e. parameter values are slightly changed in order to create new solutions.
4. New children are added to the old population, while mechanisms ensure an equal size of the population by eliminating the worst solutions.

Evolutionary optimization techniques are particularly well suited to complex problems, where classical methods fail due to the irregularity of the function to optimize or to the complexity of the search space. In this chapter, we define an *interactive evolutionary algorithm* (IEA) where users influence steps and parameters of the algorithm interactively.

EAs have been used in Neuroevolution to evolve the topology and edge weights of neural networks in their creation and learning phase. Neuroevolution faces two main problems: (a) How to encode neural networks in genes, and (b) how to design effective cross-over operators that do not destroy desired sub-structures in the networks. There are mainly two approaches to these problems: Direct encoding and grammar encoding.

Direct encoding encodes each node and edge directly as genes (Gruau, 1994). Edge genes are active or inactive, depending on whether the edge is present or not. Cross-over operators for direct encoding (Stanley & Miikkulainen, 2002) are straightforward in the sense that they swap the presence or weights of edges. With effective cross over, direct encoding can maintain individual topological structures across generations, but the chromosomes can become very large (#genes $\geq$ #edges) and have to vary in size if the network grows, which is quite unusual in EA.

Grammatical encoding encodes rules that create the neural network. Rules are described in a grammar tree, which is traversed and iteratively transforms a seed-node into multiple nodes and change their connectivity (Kitano, 1990; Suchorzewski, 2011). Evolutionary algorithms evolve the grammar tree, not the network. Grammar-based approaches are similar to the algorithm designed for GraphCuisine but differ in that these algorithms optimize *one* network at a time. Important for experiments is that a "good" network can be reproduced many times while copies are not identical, but as similar as possible. Grammar trees are deterministic.

EAs have further been used to create graph layouts. Biedl, Marks, Ryall, and Whitesides (1998) use an interactive EA and present several layouts to the user from which he can chose. The system then tries to infer aesthetic criteria the user is interested in. However, the approach focuses on finding layouts for one very small graph.

A.1.3. UIs for Data Generation and Analysis.

Few approaches exist that tightly involve the user in the data generation process. Albuquerque, Löwe, and Magnor (2011) present an interactive approach to create high-dimensional data sets using generators and sketching methods for 1D, 2D and 3D scatterplots.

Low level graph sketching and graph editing capabilities are provided by several tools (e.g. *yEd Graph Editor*, 2012). Wong, Foote, Mackey, Perrine, and Chin Jr. (2006) generate graphs through sketching on adjacency matrices and adapt pixel-based drawing techniques to draw edges inside the matrix. Although a node-link representation is provided in parallel with the matrix, only the matrix is used for drawing. Except when

drawing cliques, this makes it hard to predict the final layout of the resulting graph in the form of a node link diagram, and, except for node count and edge count, it is almost impossible to control measures of the resulting graph. In general, sketching approaches do not solve the problem of creating similar copies of a graph. GraphCuisine uses "receipes" to create graphs and reproduce graphs with similar properties. For the general creation of multivariate data, Albuquerque et al. (2011) present interactive approaches using generators and sketching methods for 1D, 2D and 3D scatterplots. Still, sketching interfaces do not create diverse data sets, but instead only provide one particular solution.

A.2. Graph Cuisine

A.2.1. Generation Overview

GraphCuisine is a random graph generator for undirected graphs that evolves a population of individuals encoded in *chromosomes*, each of them representing a "recipe" to generate a graph. A graph is generated from a chromosome by running a pipeline of generators, each reading its appropriate input parameters from the chromosome and modifying the graph by systematically adding or removing nodes and edges. GraphCuisine currently implements 12 graph generators, each using 2–5 input parameters and creating basic network structures such as stars, clusters and several types of "noise" (see Table A.1). The target graph towards which the graphs (as representatives of individuals) are evolved is encoded as a set of *target measures*—such as node count, density and clustering coefficient. The fitness of a generated graph is calculated as the distance between the measures of the generated graph and the target values (Section A.3.3).

Encoding the genome as a set of parameters for generators has two major advantages: (1) the genome size remains constant and independent of the graph size, and (2) the measures used to optimize the graphs are independent from the number and parameters of the simple generators, allowing new measures to be added easily, as well as new generators. Having constant-size genomes facilitates crossover between chromosomes, which in turn increases the diversity of generated solutions.

A.2.2. Interface

The interface of GraphCuisine consists of five major parts as shown in Figure A.2:

(a) The **population view** is a visualization of 12 representative graphs from the current population,
(b) the **detail view** shows an enlarged view of a selected graph,
(c) the **data set view**, which shows an imported graph from which target measures can be extracted,

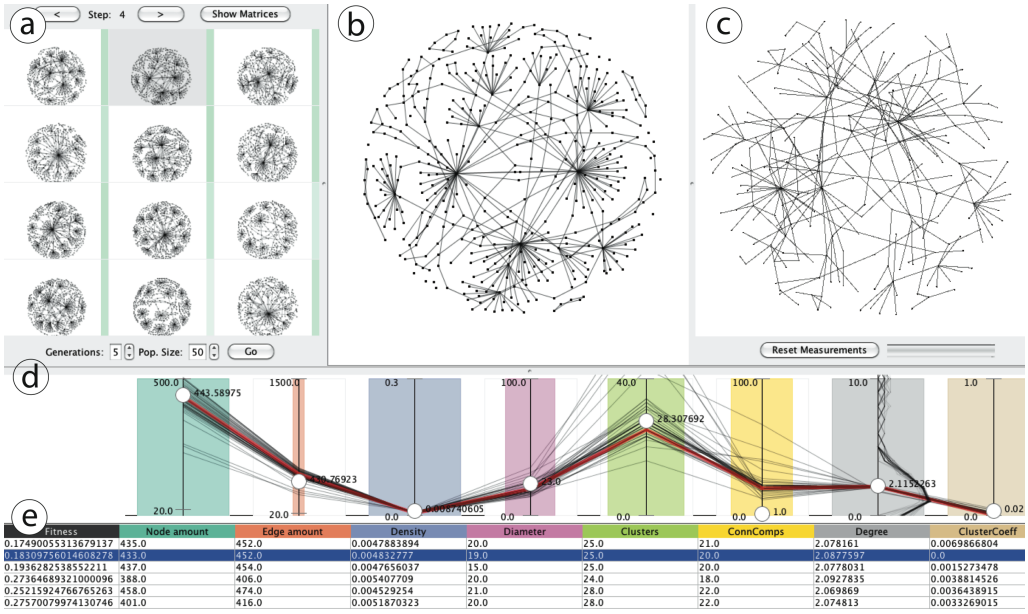


Figure A.2.: *User Interface of GraphCuisine showing (a) the population view displaying representative graphs of the current population (population view), (b) a detail view of selected graph (detail view), (c) an imported graph (data set view), (d) the parallel coordinates plot showing the distribution of measures in the population (measures view) and (e) the table with the graph measures of the representative graphs (measures table). The position of the white markers in the measures view indicate the target measures while the red line heighights the measures of the selected graph.*

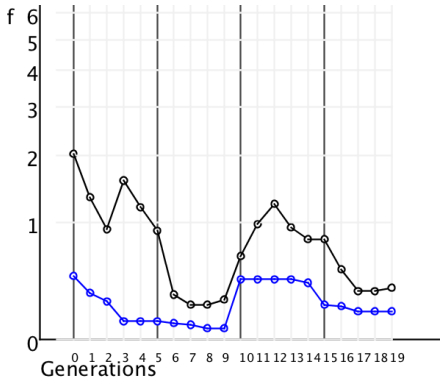


Figure A.3.: *Fitness curve showing the evolution of the population fitness in a logarithmic scale. Better fitness values are closer to 0. The blue line shows the value of the fittest individual in the population and the black shows the mean fitness. The stronger black vertical lines indicate the generations that have been presented to the user and upward peaks in the blue curve show that the user has changed the target measures, leading to a decreased overall fitness of all individuals in the current population.*

(d) the **measures view** employs an interactive parallel coordinates plot to show the distribution of measures in the entire graph population. The width of the color-coded rectangles indicates the weight of the measure in the fitness computation (Section A.3.3). As in standard parallel coordinates, each polyline represents a graph in the current population and connects all the graph's measures. Vertical axes show the value range of each measure as set by the user. Tick marks on the axis lines indicate the desired minimal and maximal values, while the white circle indicates the desired target value for the population. All values can be adjusted interactively by dragging the marks. A vertical line graph draws the distribution of node degrees for each graph, at the right of the gray *degree* measure axis. We opted for line plots because they cause less visual clutter than bar graphs and allow to show how multiple graphs converge or diverge in this measure.

(e) The **measure table** shows all measures with the population, similar to NetLens (Freire, Plaisant, Shneiderman, & Golbeck, 2010).

The user can switch between adjacency matrix and node-link representations for all the graphs on the population and detail views as well as choose between different layout algorithms to highlight particular graph patterns. Views in GraphCuisine are coordinated with brushing and linking: moving the mouse over a polyline highlights the corresponding graph in the population and table views, and vice-versa.

A.2.3. Generation Process

To initialize the graph generation, the user can choose among the following options:

- **Random initialization**: The population is created from a random set of generator parameters for each chromosome and random target measures, providing a base for a more exploratory generation.
- **Set target measures directly**: The values for the desired graph measures can be set interactively by adapting the minimal, maximal and target values as well as the measure weight directly in the measures view.
- **Graph templates**: Templates are predefined sets of generator parameters and measures selected to create graphs with particular characteristics such as *small-sized*, *medium-sized* or *large-sized*, *dense* or *sparse*, following *power-law degree* distribution, or *small-world graphs*. Templates are chosen from a menu where complementary templates can be selected at a time. New templates can be defined by the user for later reuse. Based on a template, an initial population is created.
- **Load an existing graph**: To mimic an existing graph, the graph is imported and shown in the data set view (Figure A.2(c)). Its measures are derived, displayed in the measures view, and serve as target values for the evolution. The measures and their weights can immediately be modified by the user or at any time during the evolution; they can also be reset to the value of the loaded graph.

Generating graphs with GraphCuisine consists of two alternating phases. After the initialization and selection of the target measures, the evolution of the generator parameters is started and machine solutions are created by optimizing the initial population (50 graphs) to fit the target values. After five generations, we select 12 representative graphs. Since showing the entire population is not very practical, our heuristic selects 12 representative graphs; the three fittest and nine at random for purposes of diversity. They are consequently displayed in the population view and the measure table. The measures view is then updated to display the values from the whole population. The saturation of the green bars in the population view indicates the fitness of each graph and adapts immediately if the user changes the target measures in the measures view. This feedback is required for the user to know which graphs are considered similar by the system with the current settings.

Alternatively to changing measures in the respective view, users can select graphs they judge *good* directly from the population view by clicking on them. Multiple graphs can be selected and the target measures are set to reflect the chosen graphs. Measure weights are updated according to the variance of their respective values in the selected graphs—the more a particular measure differs, the lower it is weighted and vice-versa (see Section A.3.4). The evolution can then run for a further step with the new measures being taken into account.

The number of generations as well as other parameters of the EA can be adjusted by the user at any time, allowing for control of the evolution behavior (population size, generations, elitism, etc.). Any generator can be disabled to prevent the graph from showing particular characteristics. Setting generator parameters directly gives additional freedom over the resulting graphs.

Since our fitness value reflects the difference between measures of each graph and the target measures, a lower fitness value means a higher fitness. Behavior and convergence of the fitness of the whole population is monitored in the chart shown in Figure A.3. It shows *minimal* (best), *maximal* (worst) and *average* fitness throughout all generations using a logarithmic scale. In order to avoid previous graphs being lost during the evolution process, GraphCuisine can be reset to any generation to try alternative evolutions. Additionally, the chromosome of any generated graph can be saved as template to initialize new populations later on.

A.3. An Evolutionary Algorithm for Graph Generation

As explained in the previous section, each individual in GraphCuisine's EA corresponds to a graph. It is encoded in a generative way, i.e., its *chromosome*, which fully specifies a sequence of generators, their parameters, and the corresponding values. The creation of a graph from a chromosome and the evaluation of its fitness is done in four steps, illustrated in Figure A.4: (1) From a chromosome, parameters are fed into the graph generators that,

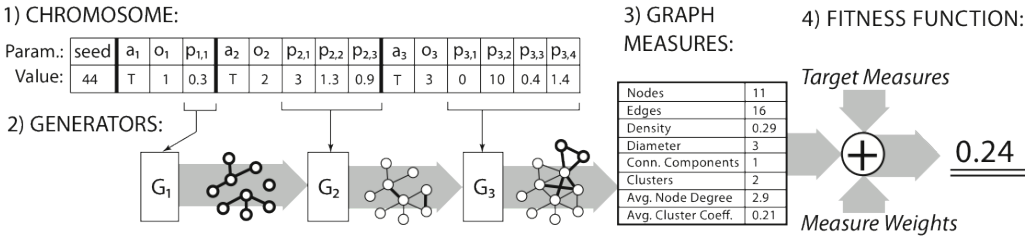


Figure A.4.: *Graph creation process: (1) Parameters encoded in genes of a chromosome, (2) employ graph generators, (3) extract graph measures, and (4) calculate fitness value.*

forming a pipeline, (2) generate a graph. The generated graph is then evaluated according to the (3) measure values and weights, (4) calculating a unique fitness value.

A.3.1. Chromosome Structure: Internal Encoding of an Individual

Each chromosome is divided into three blocks. The first is a single gene with an integer value, which serves as a seed for the random number generator to ensures consistent random values for the generation. The second and third block stand for two different kinds of generators, explained in the next section (Section A.3.2). They are both represented in the same way: the first gene contains a Boolean value indicating whether the corresponding generator is active or not (a_i), and the second gene contains a value specifying the execution order of the generator in the creation pipeline (o_i) (Figure A.4). The remaining genes of each generator contain their input parameters (p_i).

A.3.2. Graph Generators

We implemented two specific types of generators in GraphCuisine: *motif generators* and *noise generators*. They are applied in two steps: motif generators start by creating topological patterns, such as clusters, stars, paths, and cycles. The generators' parameters specify subgraph-specific properties (e.g., the degree of a star's central node, detailed in Table A.1) and how many instances of it should be created. Noise generators are applied in a second step to break these regularities, making structures more varied and less predictable by adding or removing nodes and edges. Nodes and edges can be inserted randomly with different node degree distributions: uniform, power-law, logarithmic or exponential. Nodes and edges to be removed are either chosen randomly or according to some criterion, such as eliminating isolated nodes (degree 0). These two steps are conceptually similar to some structural generators but the motifs and noise we produce are more varied and generalized.

Parameters for noise generators define how many nodes or edges the graph should have (and thus how many should be added or removed) as a means to control the approximate size of the graph. More specific properties exist, such as the probability of connecting

Generator	Description	Parameters
Cluster	Create dense subgraphs.	Number of clusters (integer), ideal cluster size (integer), size variability (double), density (double).
Path	Sequence of nodes in which there is always an edge from one node to the next.	Number of paths (integer), ideal path length (integer), length variability (double). Star and end nodes are chosen randomly from the graph.
Cycle	Closed path. The start/end node is an existing node.	Number of cycles (integer), ideal cycle size (integer), size variability (double).
Star	Tree with one central node and a given number of leaves.	Number of stars (integer), ideal central node degree (integer), central node degree variability (double).
Edge Noise	Adds edges linking randomly chosen nodes.	Ideal number of edges the graph should have (integer).
Eppstein Wang Power Law Noise	Adds or removes edges according to the model by Eppstein and Wang (2002).	Number of iterations (integer).
Exponential Edge Noise	Adds edges while trying to achieve an exponential degree distribution.	Ideal number of edges the graph should have (integer).
Logarithmic Edge Noise	Adds edges while trying to achieve a logarithmic degree distribution.	Ideal number of edges the graph should have (integer).
Node Noise	Adds nodes until the target count for the graph is reached.	Ideal number of nodes the graph should have (integer).
Orphan Cleanup Noise	Removes nodes with degree 0.	Percentage of orphans to be removed (double).
Random Edge Cleanup Noise	Removes randomly chosen edges.	Percentage of edges to be removed (double).
Random Node Cleanup Noise	Removes randomly chosen nodes.	Percentage of nodes to be removed (double).

Table A.1.: *GraphCusine's Motif (above) and noise generators (below).*

two given nodes by an edge. The count of nodes and edges to be inserted and which nodes should be connected depend on what is already in the graph. As such, if applied in a different order, the same noise generators with the same parameters will produce different graphs.

A.3.3. Fitness Function: Quality Assessment of an Individual

The computation of the fitness of each chromosome/graph is based on the set of target graph measures $M = \{m_i | m_i \in \mathbb{R}, i \in [0,n]\}$, with tolerance bounds $m_i \in [\min_i, \max_i]$, and importance weights $w_i \in [0,1]$. For a graph G_k, each measure in M is computed as: $M_k = (m_{0,k}, \ldots, m_{n,k})$ where $m_{i,k}$ is the i^{th} graph measure. The $fitness(G_k) \in \mathbb{R}^+$ of the graph G_k is its distance to the optimal solution, i.e. the following weighted sum:

$$fitness(G_k) = \sum_{i \in [0,n]} w_i * \left| \frac{m_i - m_{i,k}}{\max_i - \min_i} \right|$$

Normalization is necessary to weight and sum up measures equally. Currently, GraphCuisine supports the following measures: *node amount*, *edge amount*, *density*, *graph diameter*, *number of clusters* (using the `EdgeBetweennessClusterer` implemented in JUNG (O'Madadhain et al., 2003)), *number of connected components*, *average clustering coefficient*, and *average node degree*.

Some of the measures such as node amount, edge amount and density, are interdependent so that if the user restricts one measure, others are restricted automatically. We recommend to start with few measures (high weight) while observing the first generations and then add and adjust additional measures. It requires more investigation to provide appropriate interface components to communicate and manage such dependencies.

A.3.4. Interactive Evolution

The population view allows selecting *good* solutions by visual inspection. This feature is based on the assumption that visual node link or matrix representations allow humans to compare graphs and match some of their visible structure effectively. When selecting multiple graphs S with $G_k \in S$, GraphCuisine tries to infer what graph characteristics are important to the user. The target value for each measure m_i is taken to be the average of measures of all selected graphs: $m_i = \overline{m_{i,k}}$. The weights w_i are adjusted to reflect the diversity or similarity of the single graph measures using the standard deviation of the normalized graph measures $m_{i,k}$:

$$w_i = stddev(m'_{i,0}, \ldots, m'_{i,s}) \text{ with } m'_{i,k} = \frac{m_i - m_{i,k}}{\max_i - \min_i}$$

When users change the target measures or their weights, we increase the mutation rate for one generation in order to yield a larger variety of new solutions.

A.3.5. Implementation

GraphCuisine is implemented in Java using the JGAP library[1]. By default, the graph generation process begins with a randomly generated population of 50 individuals, which is evolved for 5 generations in each evolution cycle. These values were chosen to keep a balance between population diversity and acceptable running time for each cycle, an important issue for GraphCuisine due to its interactive nature. In order not to lose good solutions over time, we guarantee that each generation keeps 30% of the individuals from the previous generation (*elitism*). The components of the evolutionary algorithm are the following:

[1] http://jgap.sourceforge.net

- *A tournament selection of size five,* meaning that five individuals are randomly chosen with uniform probability, with the best one being kept as a parent. The process is repeated twice to select two parents for a crossover.
- *A one-point crossover* that creates two offsprings by swapping all the genes of the parents that come after a randomly selected position in the genome.
- *A simple random mutation* that takes 5% of the genes of each individual and replaces them with random values.

Crossover and mutation are applied in cascade. First, new individuals are created by applying a crossover, then all new individuals undergo a mutation that alters 5% of their genes. The configuration of GraphCuisine 's EA is based on the specific problem we approach and on results obtained from experimentation with the different parameters.

Performance of the generation and evaluation process is essentially limited by the computation time for the graph measures (in particular those related to clusters), with creation, mutation, and crossover times being negligible. Measure weights can be set to 0, which accelerates the process by not computing this measure.

Adding new measures or generators to GraphCuisine consists of adding the new implementations and update the user interface. Measures must be added to the fitness function, while new generators require that their parameters be included in the chromosome structure. Integrating new measures into the interface amounts to adding columns to the table and dimensions to the parallel coordinates plot.

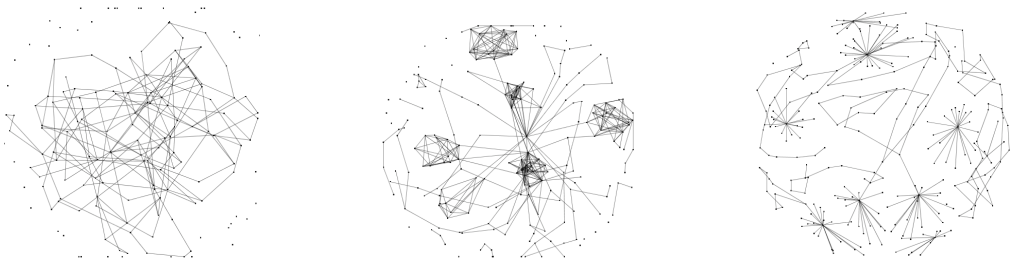
A.4. Case Description

A.4.1. HIV Network

Imagine an HIV (Human Imune Virus) contamination network[2] that must be anonymized while keeping the characteristic structures present so that it can be given to students in epidemiology for analysis. By importing the network in GraphCuisine, its measures are calculated and automatically set as target measures. The graph population is randomly initialized to guarantee diversity. After a first evolution step of five generations, the representative graphs are shown in the population view and the measures table. The parallel coordinates in the measures view shows that the distribution of measure values for all the 50 graphs in the current population has already converged well towards the target values.

The user can then decide to increase the graph size and, proportionally, the number of clusters. This is done by setting the desired *size* and the *number of clusters* in the measures view. Since these two seem to be the most important measures for this evolution, the user slightly increases their weight by varying the width of the colored rectangles in the

[2] http://insitu.lri.fr/~nhenry/socnets/realtreelike/Hiv.xml



(a) (b) (c)

Graph	Node Count	Edge Count	Density	Diameter	Clusters	Avg Degree	Avg Clust. coefficient
(a)	161	191	0.014	9	49	2.37	0.018
(b)	213	457	0.020	15	43	4.30	0.230
(c)	283	273	0.007	11	34	1.93	0.213

Figure A.5.: *Examples of Graphs from a random chromosome initialization.*

parallel coordinates plot. After another evolution cycle, the graphs have grown and show a proportionally higher number of clusters. The degree of optimization and similarity between the generated graphs and the original one can be controlled by adjusting the number of generations in the evolution step.

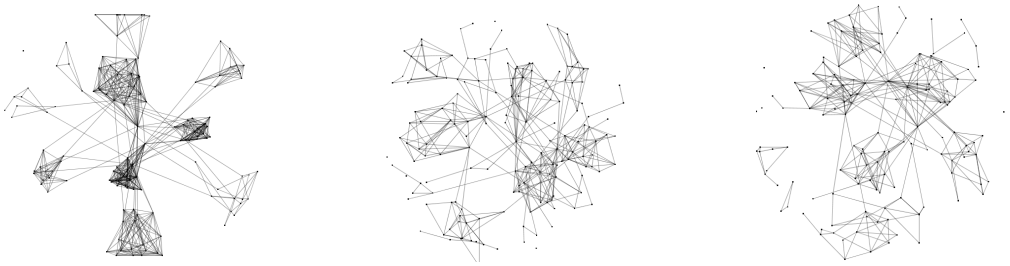
If desired, the user can continue to refine the graphs by, for example, selecting their favorite representative graphs from the population view and running another evolution cycle to obtain graphs similar to the ones they chose. Since selecting graphs causes the target measures to reflect the measures of the selection, they might slightly diverge from those of the imported graph. When a desired result has been reached, generated graphs can then be exported. Figure A.2 shows the imported HIV network (right), the generated graphs (left), as well as a particular selected graph (center) with its measures highlighted in red in the measures view. The fitness view (Figure A.3) shows the convergence of the population fitness declining, unless the target measures have been changed by the user. The pure evolution time for 50 individuals for 20 generations was 6.52 minutes.

A.4.2. Further Examples

Examples of different types of graphs generated with GraphCuisine, in Figures A.5, A.6, A.7, and A.8, are shown along with their measures.

A.5. Generating Dynamic Networks with GraphTabasco

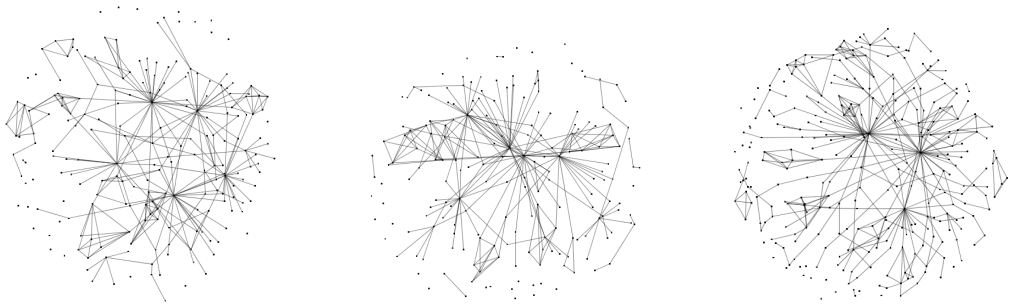
With GraphCuisine we explored a way to generate static graphs in an interactive way and by including users' feedback. Generating dynamic networks is a much more complex process that requires a deep understanding of changes in over time. Simulating dynamic changes in networks is best expressed by preferential attachment generators (Section



(a) (b) (c)

Graph	Node Count	Edge Count	Density	Diameter	Clusters	Avg Degree	Avg Clust. coefficient
(a)	140	733	0.075	7.0	11	10.471	0.621
(b)	161	379	0.029	10.0	12	4.708	0.251
(b)	146	420	0.039	7.0	10	5.753	0.273

Figure A.6.: *Examples of small-world networks created with GraphCuisine.*



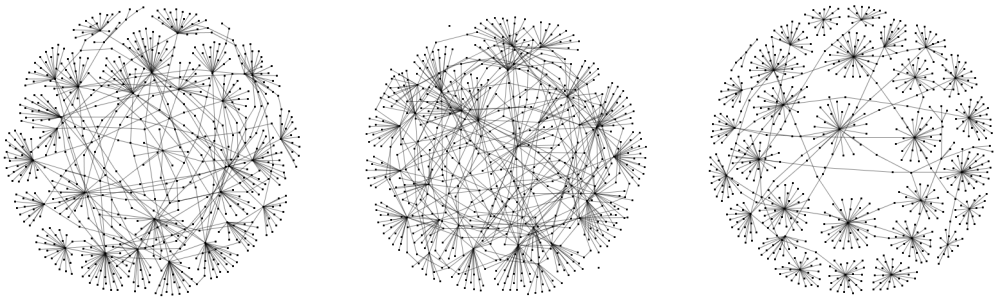
(a) (b) (c)

Graph	Node Count	Edge Count	Density	Diameter	Clusters	Avg Degree	Avg Clust. coefficient
(a)	208	285	0.013	14	31	2.740	0.209
(b)	190	243	0.013	9	35	2.557	0.167
(c)	296	396	0.009	14	35	2.675	0.163

Figure A.7.: *scale-world networks using, among others, the Eppstein-Wang generator.*

A.1.1), since in these generators, graph generation is already modelled as an iterative and evolutionary process. However, these generators generate *one* graph with particular measures, rather than a *sequence*; they do not model abrupt changes, node removal, or control for trends.

Two very recent generators for dynamic network exist that model events and their probability in the graph evolution, both having been published at the same time as GraphCuisine. Jónsson, Vigfússon, and Helgason (2012) use node insertion events whereby nodes have a minimum and an average lifetime. Edges are exchanged as nodes enter and leave the network, or get updated randomly. Attachment probabilities are based on the model by Barabási and Albert (1999). Görke, Kluge, Schumm, Staudt, and Wagner (2012) describe a generator that is also based on Barabási and Albert's model and that models clustered graphs. In each time step, the algorithm decides if a cluster splits or



(a) (b) (c)

Graph	Node Count	Edge Count	Density	Diameter	Clusters	Avg Degree	Avg Clust. coefficient
(a)	673	751	0.003	17.0	–	2.231	–
(b)	600	750	0.004	9.0	–	2.5	–
(c)	745	752	0.002	22.0	–	2.018	–

Figure A.8.: *Three randomly picked graphs from the same population showing convergence across almost all measures after five generations (population size=50). Measures showing "–" have been disabled in order to increase evolution speed.*

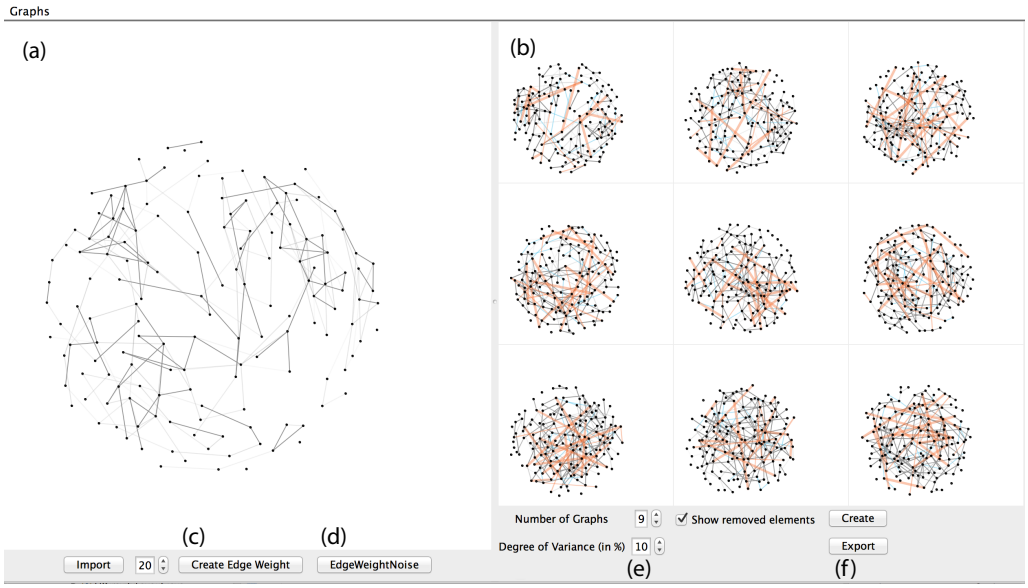


Figure A.9.: *GraphTabasco interface. (a) imported graph, (b) generated graphs, (c) apply edge weight according to distance in layout (close nodes are connected with higher weight), (d) apply random edge weight, (e) adjust generation parameters, (f) export graphs into files.*

two clusters merge, then adding and removing nodes randomly. Between two cluster states, there is a transition period, where clusters get organized in order to avoid abrupt changes in their structure. However, the already discussed issues with generators for

static networks remain; determining input parameters, the quality of generated patters, and lack of users' control over the generation.

Currently, we have chosen a simple way to generate dynamic networks, based on the graphs generated with GraphCuisine. While evaluating our designs for comparing two weighted graphs (Chapter 5), we required multiple similar graphs, but with certain (controlled) differences. I implemented an extension to GraphCuisine, *GraphTabasco*, which takes one graph as input and generates an arbitrary number of *copies* with some adjustable variability. Different from GraphCuisine, where each graph is generated from scratch, graphs in GraphTabasco are *true* copies of the input graph, i.e. node-IDs match across graphs. With GraphTabasco we can (i) introduce (random) edge weights to the imported graph and its copies, (ii) create copies of the imported graph which are modified individually, and (iii) control for changes in the copies. While for the evaluation in Chapter 5, we created only one copy of the input graph, dynamic networks can easily be created by putting multiple individual graphs into a sequence. Since they all originate from the same input graph, the basic structure (e.g. clusters) are maintained.

Figure A.9 shows the interface of GraphTabasco. On the left (a) a graph created with GraphCuisine has been imported, and on the right (b) a set of graphs (read left-right, top-bottom). Blue and orange edges indicate added and removed edges, respectively. Variation is currently controlled by only one parameter (*Degree of Variance*, (e)) indicating the percentage of added and removed edges. However, there is a potentially large number of parameters (number of edges and nodes, to be added, nodes and edges to be removed, variability of values, etc.).

An optimization algorithm, such as in GraphCuisine, may eventually be used to optimize these parameters, while GraphTabasco can support visual feedback and ways of controlling the generation. Matrix Cubes have a high potential in providing quick overview of each generated dynamic networks.

A.6. Conclusion

This chapter described GraphCuisine, a system that uses an evolutionary algorithm to generate static random graphs according to user-specified graph measures. We encode graphs using a generative model where several generators are run in sequence, each taking as input a set of parameters. We store the parameters and execution order in the genome of the EA engine and evolve it to produce graphs matching the set of target measures. GraphCuisine is available online for download at `http://www.aviz.fr/Research/Graphcuisine`. GraphCuisine has been complemented with GraphTabasco to generate a multitude of exact, but slightly different copies, as well as random dynamic graphs.

Generation Process GraphCuisine's parameter encoding has several advantages for our specific goal compared to directly encoding graphs in the genome: it is efficient, both in space and computation time. One genome can potentially generate an unlimited number of similar random graphs. Changing the parameters through mutation generates more diversity than direct encoding. Diversity, in turn, leads to faster convergence of the evolution and can introduce interesting unexpected results that still match the constraints. In our experience, GraphCuisine's EA converges rapidly. However, we want to conduct more formal studies to better understand the behavior of our generators compared to existing ones in terms of convergence and expressive power. Such a formal study would also help to balance speed of convergence of the evolution against diversity in the population.

Usability Generating graphs with GraphCuisine does require some understanding of graphs and the meaning of the particular measures. However, users do not require deep knowledge about evolutionary algorithms and can freely explore the generation space for graphs using the GraphCusisine interface. In order to improve the interface and the underlying algorithm, as well as define additionally required features, an evaluation with actual practitioners remains to be done.

Extensibility The set of generators, measures and templates currently implemented in GraphCuisine is limited to maintain controllability. Extending it is straightforward; new generators can be added to the pipeline and their parameters can be are added to the chromosome; new measures can simply be added and integrated into the interface. We continue seeking for a good minimal set of generators and measures and how to prevent users from obtaining empty search spaces due to dependencies between measures (e.g. *node amount*, *edge amount*, and *density*). Another extension we are working on is to express and match measure distributions such as power-law node degrees. The EA would require little change to support this, but the interface would need more work to effectively let the user specify the distributions.

Dynamic Networks As already mentioned in Section A.5, we want to extend the scope of GraphCusisine towards dynamic networks. Besides proper measures to evaluate dynamic networks by an automatic optimization algorithm, a problem remains the effective visual comparison of the generated networks.

***

Acknowledgements *This work was partially supervised by Evelyne Lutton (INRIA). Interface design and generator design, as well as major parts of the chromosome encoding origins from me. Programming was shared with André Spritzer (Universidade Federal do Rio Grande do Sul, Brazil), while he was responsible for the evolutionary algorithm. Discussions majorly took part between André and me.*

Bibliography

References

3d scatterplot. (2007). http://en.wikipedia.org/wiki/Scatter_plot#mediaviewer/File:Scatter_plot.jpg. (online, accessed:3-Jul-2014)

Abello, J., Kobourov, S. G., & Yusufov, R. (2004). Visualizing large graphs with compound-fisheye views and treemaps. In *proceedings of 12th symposium on graph drawing* (pp. 431–441). Springer.

Abello, J., & van Ham, F. (2004). Matrix zoom: A visual interface to semi-external graphs. In *Proceedings of the IEEE symposium on information visualization* (pp. 183–190). Washington, DC, USA: IEEE Computer Society. doi: 10.1109/INFOVIS.2004.46

Abello, J., van Ham, F., & Krishnan, N. (2006). Ask-graphview: A large scale graph visualization system. *IEEE Transactions on Visualization and Computer Graphics*, *12*(5), 669-676. doi: 10.1109/TVCG.2006.120

Achard, S., Salvador, R., Whitcher, B., Suckling, J., & Bullmore, E. (2006). A Resilient, Low-Frequency, Small-World Human Brain Functional Network with Highly Connected Association Cortical Hubs. *The Journal of Neuroscience*, *26*(1), 63–72. doi: http://dx.doi.org/10.1523/JNEUROSCI.3874-05.2006

Ahmed, A., Dywer, T., Hong, S.-H., Murray, C., Song, L., & Wu, Y. X. (2005). Visualisation and analysis of large and complex scale-free networks. In *Proceedings of the seventh joint eurographics / IEEE VGTC conference on visualization* (pp. 239–246). Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. doi: 10.2312/VisSym/EuroVis05/239-246

Ahn, J., Plaisant, C., & Shneiderman, B. (2012). *A Task Taxonomy for Network Evolution Analysis* (Tech. Rep. No. HCIL-2012-13). Boston, MD: HCIL, University of Maryland.

Ahn, J.-W., Taieb-Maimon, M., Sopan, A., Plaisant, C., & Shneiderman, B. (2011). Temporal visualization of social network dynamics: prototypes for nation of neighbors. In *Proceedings of the 4th international conference on social computing, behavioral-cultural modeling and prediction* (pp. 309–316). Berlin, Heidelberg: Springer.

Aigner, W., Miksch, S., Schumann, H., & Tominski, C. (2011). *Visualization of time-oriented data* (1st ed.). Springer. doi: 10.1007/978-0-85729-079-3

Albuquerque, G., Löwe, T., & Magnor, M. (2011). Synthetic Generation of High-Dimensional Datasets. *IEEE Transactions on Visualization and Computer Graphics*,

17(12), 2317–2324.

Allen, J. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 123-154.

Alper, B., Bach, B., Henry Riche, N., Isenberg, T., & Fekete, J.-D. (2013). Weighted graph comparison techniques xfor brain connectivity analysis. In *Proceedings of ACM CHI conference on human factors in computing systems* (pp. 483–492).

Alper, B., Riche, N., Ramos, G., & Czerwinski, M. (2011). Design Study of LineSets, a Novel Set Visualization Technique. In *IEEE transactions on visualization and computer graphics* (Vol. 17, pp. 2259–2267).

Amar, R., Eagan, J., & Stasko, J. (2005). Low-Level Components of Analytic Activity in Information Visualization. In *Proceedings of InfoVis* (p. 111-117). IEEE.

Andrews, K., Wohlfahrt, M., & Wurzinger, G. (2009). Visual Graph Comparison. In *Proceedings of iv* (pp. 62–67). Los Alamitos: IEEE Computer Society. doi: http://dx.doi.org/10.1109/IV.2009.108

Andrienko, G., & Andrienko, N. (1999). Making a GIS intelligent: CommonGIS project view. *AGILE99*, 19–24.

Andrienko, N., & Andrienko, G. (2004). Interactive visual tools to explore spatio-temporal variation. In *Proceedings of the working conference on advanced visual interfaces* (pp. 417–420). New York, NY, USA: ACM. doi: 10.1145/989863.989940

Andrienko, N., & Andrienko, G. (2006). *Exploratory analysis of spatial and temporal data*. Springer.

Anscombe, F. J. (1973, Feb). Graphs in Statistical Analysis. *The American Statistician*(1), 17-2.

Archambault, D., Munzner, T., & Auber, D. (2007). Topolayout: Multilevel graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics*, *13*(2), 305-317. doi: 10.1109/TVCG.2007.46

Archambault, D., Munzner, T., & Auber, D. (2008, July). Grouseflocks: Steerable exploration of graph hierarchy space. *IEEE Transactions on Visualization and Computer Graphics*, *14*(4), 900–913. doi: 10.1109/TVCG.2008.34

Archambault, D., & Purchase, H. (2012). The mental map and memorability in dynamic graphs. In *Pacific visualization symposium (pacificvis), 2012 IEEE* (p. 89-96). doi: 10.1109/PacificVis.2012.6183578

Archambault, D., & Purchase, H. C. (2013). Mental map preservation helps user orientation in dynamic graphs. In *Proceedings of graph drawing* (Vol. 7704, pp. 475–486).

Archambault, D., Purchase, H. C., & Pinaud, B. (2011a). Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE Transactions on Visualization and Computer Graphics*, *17*(4), 539-552. doi: 10.1109/TVCG .2010.78

Archambault, D., Purchase, H. C., & Pinaud, B. (2011b). Difference map readability for dynamic graphs. In *Proceedings of graph drawing* (Vol. 6502, pp. 50–61).

Archip, N., Clatz, O., Whalen, S., Kacher, D., Fedorov, A., Kot, A., ... others (2007). Non-Rigid Alignment of Pre-Operative MRI, fMRI, and DT-MRI with Intra-Operative MRI for Enhanced Visualization and Navigation in Image-Guided Neurosurgery. *NeuroImage*, *35*(2), 609–624. doi: http://dx.doi.org/10.1016/j.neuroimage.2006.11.060

Auber, D. (2003). Tulip : A huge graph visualisation framework. In *Graph drawing softwares* (p. 105-126). Springer.

Bach, B., Pietriga, E., & Liccardi, I. (2013). Visualizing Populated Ontologies with OntoTrix. *International Journal on Semantic Web and Information Systems*. (to appear)

Baker, M. P., & Bushell, C. (1995). After the storm: Considerations for information visualization. *Computer Graphics and Applications, IEEE*, *15*(3), 12–15.

Balzer, M., & Deussen, O. (2007). Level-of-detail visualization of clustered graph layouts. In *Proceedings of 6th international asia-pacific symposium on visualization (APVIS)* (p. 133-140). doi: 10.1109/APVIS.2007.329288

Barabási, A.-L., & Albert, R. (1999). Emergence of Scaling in Random Networks. *Science*, *286*(5439), 509–512.

Bar-Joseph, Z., Demaine, E. D., Gifford, D. K., Srebro, N., Hamel, A. M., & Jaakkola, T. S. (2003). K-ary clustering with optimal leaf ordering for gene expression data. *Bioinformatics*, *19*(9), 1070-1078. doi: 10.1093/bioinformatics/btg030

Bartram, L. (1997). Perceptual and interpretative properties of motion for information visualization. In *Proceedings of the 1997 workshop on new paradigms in information visualization and manipulation* (pp. 3–7).

Bassett, D., Brown, J., Deshpande, V., Carlson, J., & Grafton, S. (2011). Conserved and Variable Architecture of Human White Matter Connectivity. *NeuroImage*, *54*(2), 1262–1279. doi: http://dx.doi.org/10.1016/j.neuroimage.2010.09.006

Bassett, D., Bullmore, E., Verchinski, B., Mattay, V., Weinberger, D., & Meyer-Lindenberg, A. (2008). Hierarchical Organization of Human Cortical Networks in Health and Schizophrenia. *J. Neurosci.*, *28*(37), 9239–9248. doi: http://dx.doi.org/10.1523/JNEUROSCI.1929-08.2008

Bastian, M., Heymann, S., & Jacomy, M. (2009). Gephi: An open source software for exploring and manipulating networks. In *Proceedings of international AAAI conference on weblogs and social media.*

Battiato, S., Gallo, G., Puglisi, G., & Scellato, S. (2007). Sift features tracking for video stabilization. In *Proceedings of conference on image analysis and processing* (pp. 825–830).

Battista, G. D., Eades, P., Tamassia, R., & Tollis, I. G. (1998). *Graph drawing: Algorithms for the visualization of graphs* (1st ed.). Upper Saddle River, NJ, USA: Prentice Hall PTR.

Becker, R. A., & Cleveland, W. S. (1987). Brushing scatterplots. *Technometrics*, *29*(2), 127–142.

Beckmann, M., Johansen-Berg, H., & Rushworth, M. (2009). Connectivity-Based Parcellation of Human Cingulate Cortex and Its Relation to Functional Specialization. *J. Neurosci.*, *29*(4), 1175–1190. doi: http://dx.doi.org/10.1523/JNEUROSCI.3328-08.2009

Bender-deMoll, S., & McFarland, D. A. (2005). The Art and Science of Dynamic Network Visualization. *Journal of Social Structure*.

Berger, N., Borgs, C., Chayes, J., D'Souza, R., & Kleinberg, R. (2004). Competition-induced Preferential Attachment. In *Proceedings of colloquium on automata, languages and programming* (pp. 208–221).

Bertault, F., & Miller, M. (1999). An algorithm for drawing compound graphs. In J. Kratochvà (Ed.), *Graph drawing* (Vol. 1731, p. 197-204). Springer. doi: 10.1007/3-540-46648-7_20

Bertin, J. (1973). *Sémiologie graphique - les diagrammes, les reseaux, les cartes*. Paris: Éditions Gauthier-Villars.

Bezerianos, A., Chevalier, F., Dragicevic, P., Elmqvist, N., & Fekete, J. D. (2010). Graphdice: a system for exploring multivariate social networks. In *Proceedings of the 12th eurographics / IEEE- VGTC conference on visualization* (pp. 863–872). Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. doi: 10.1111/j.1467-8659.2009.01687.x

Bezerianos, A., Dragicevic, P., Fekete, J.-D., Bae, J., & Watson, B. (2010, November). Geneaquilts: A system for exploring large genealogies. *IEEE Transactions on Visualization and Computer Graphics*, *16*(6), 1073–1081. doi: 10.1109/TVCG.2010.159

Biedl, T. C., Marks, J., Ryall, K., & Whitesides, S. (1998). Graph Multidrawing: Finding Nice Drawings Without Defining Nice. In *Graph drawing 1998* (pp. 347–355). Springer.

Bilgin, C. C., & Yener, B. (2010). *Dynamic network evolution: Models, clustering, anomaly detection.*

Botchen, R. P., Bachthaler, S., Schick, F., Chen, M., Mori, G., Weiskopf, D., & Ertl, T. (2008). Action-based multifield video visualization. *IEEE Transactions on Visualization and Computer Graphics*, *14*(4), 885–899.

Boyandin, I., Bertini, E., & Lalanne, D. (2012). A qualitative study on the exploration of temporal changes in flow maps with animation and small-multiples. *Computer Graphics Forum*, *31*(3pt2), 1005–1014. doi: 10.1111/j.1467-8659.2012.03093.x

Brandes, U., & Corman, S. R. (2003, March). Visual unrolling of network evolution and the analysis of dynamic discourse. *Information Visualization*, *2*(1), 40–50. doi: 10.1057/palgrave.ivs.9500037

Brandes, U., Dwyer, T., & Schreiber, F. (2004). Visual Understanding of Metabolic Pathways Across Organisms Using Layout in Two and a Half Dimensions. *Journal of Integrative Bioinformatics*, *1*(1), 2:1–2:16. doi: http://dx.doi.org/10.2390/biecoll-jib-2004-2

Brandes, U., Freeman, L. C., & Wagner, D. (2013). Handbook of Graph Drawing and Visualization. In R. Tamassia (Ed.), (chap. Social Networks). Chapman and Hall/CRC.

Brandes, U., & Mader, M. (2012). A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In M. Kreveld & B. Speckmann (Eds.), *Graph drawing* (Vol. 7034, p. 99-110). Springer. doi: 10.1007/978-3-642-25878-7_11

Brandes, U., & Nick, B. (2011, December). Asymmetric Relations in Longitudinal Social Networks. *IEEE Transactions on Visualization and Computer Graphics*, *17*(12), 2283–2290. doi: 10.1109/TVCG.2011.169

Brandes, U., & Wagner, D. (1997). A bayesian paradigm for dynamic graph layout. In G. DiBattista (Ed.), *Graph drawing* (Vol. 1353, p. 236-247). Springer. doi: 10.1007/3-540-63938-1_66

Brandes, U., & Wagner, D. (2003). Visone – Analysis and Visualization of Social Networks. In *Graph drawing software* (pp. 321–340). Springer.

Branke, J. (2001). Dynamic graph drawing. In M. Kaufmann & D. Wagner (Eds.), *Drawing graphs* (Vol. 2025, p. 228-246). Springer. doi: 10.1007/3-540-44969-8_9

Bridgeman, S., & Tamassia, R. (1998). Difference metrics for interactive orthogonal graph drawing algorithms. In S. Whitesides (Ed.), *Graph drawing* (Vol. 1547, p. 57-71). Springer. doi: 10.1007/3-540-37623-2_5

Bullmore, E., & Sporns, O. (2009). Complex Brain Networks: Graph Theoretical Analysis of Structural and Functional Systems. *Nature Reviews Neuroscience*, *10*(3), 186–198. doi: http://dx.doi.org/10.1038/nrn2618

Bunke, H., & Shearer, K. (1998). A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, *19*(3–4), 255 - 259. doi: http://dx.doi.org/10.1016/S0167-8655(97)00179-7

Burch, M., & Diehl, S. (2008). Timeradartrees: Visualizing dynamic compound digraphs. *Computer Graphics Forum*, *27*(3), 823–830. doi: 10.1111/j.1467-8659.2008.01213.x

Burch, M., Vehlow, C., Beck, F., Diehl, S., & Weiskopf, D. (2011, December). Parallel edge splatting for scalable dynamic graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, *17*(12), 2344–2353. doi: 10.1109/TVCG.2011.226

Calvert, K., Doar, M., & Zegura, E. (1997). Modeling Internet Topology. *IEEE Communications Magazine*, *35*(6), 160–163.

Cao, C., & Slobounov, S. (2010). Alteration of Cortical Functional Connectivity as a Result of Traumatic Brain Injury Revealed by Graph Theory, ICA, and sLORETA Analyses of EEG Signals. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, *18*(1), 11–19. doi: http://dx.doi.org/10.1109/TNSRE.2009.2027704

Card, S. K., Mackinlay, J. D., & Shneiderman, B. (1999). *Readings in Information Visualization, Using Vision to Think*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Card, S. K., Sun, B., Pendleton, B., Heer, J., & Bodnar, J. (2006). Time Tree: Exploring Time Changing Hierarchies. In *Proceedings of IEEE conference on visual analytics science and technology (IEEE VAST)* (pp. 3–10). IEEE.

Carlson, J. M., & Doyle, J. (1999, Aug). Highly Optimized Tolerance: A Mechanism for Power Laws in Designed Systems. *Physical Review E*, *60*, 1412–1427. doi: 10.1103/PhysRevE.60.1412

Carpendale, S. (2008). Evaluating information visualizations. In A. Kerren, J. Stasko, J.-D. Fekete, & C. North (Eds.), *Information visualization* (Vol. 4950, p. 19-45). Springer Berlin Heidelberg. doi: 10.1007/978-3-540-70956-5_2

Carpendale, S., Cowperthwaite, D. J., Fracchia, F. D., & Shermer, T. (1996). Graph folding: Extending detail and context viewing into a tool for subgraph comparisons. In F. Brandenburg (Ed.), *Graph drawing* (Vol. 1027, p. 127-139). Springer. doi: 10.1007/BFb0021797

Carpendale, S., Cowperthwaite, D. J., Tigges, M. T., Fall, A., & Fracchia, F. D. (1999). The Tardis: A Visual Exploraition Environment for Landscape Dynamics. In *Spie conference on visual data exploration and analysis vi* (p. 110-119).

Carpendale, S., Fall, A., Cowperthwaite, D. J., Fall, J., & Fracchia, F. D. (1996). Case study: visual access for landscape event based temporal data. In *Proceedings of visualization* (p. 425-428). IEEE Computer Society. doi: 10.1109/VISUAL.1996.568148

Cassinelli, A., & Ishikawa, M. (2005). Khronos Projector. In *Proceedings of international conference and exhibition on computer graphics and interactive techniques (SIGGRAPH).* ACM.

Chang, B.-W., & Ungar, D. (1993). Animation: From cartoons to the user interface. In *Proceedings of the 6th annual acm symposium on user interface software and technology* (pp. 45–55). New York, NY, USA: ACM. doi: 10.1145/168642.168647

Chevalier, F., Dragicevic, P., Bezerianos, A., & Fekete, J.-D. (2010). Using text animated transitions to support navigation in document histories. In *Proceedings of chi.* ACM.

Chi, E. H.-h. (2000). A taxonomy of visualization techniques using the data state reference model. In *Proceedings of IEEE symposium on information visualization* (pp. 69–75).

Chi, E. H.-h., Riedl, J., Barry, P., & Konstan, J. (1998, July). Principles for information visualization spreadsheets. *IEEE Compututer Graphics and Applications*, *18*(4), 30–38. doi: 10.1109/38.689659

Choi, B.-T., Lee, S.-H., & Ko, S.-J. (2000). New frame rate up-conversion using bi-directional motion estimation. *IEEE Transactions on Consumer Electronics*, *46*(3), 603-609. doi: 10.1109/30.883418

Collberg, C., Kobourov, S., Nagra, J., Pitts, J., & Wampler, K. (2003). A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 acm symposium on software visualization* (pp. 77–ff). New York, NY, USA: ACM. doi: 10.1145/774833.774844

ColorBrewer. (2002). http://colorbrewer2.org.

Conti, G., Ahamad, M., & Stasko, J. (2005). Attacking information visualization system usability overloading and deceiving the human. In *Proceedings of the 2005 symposium on usable privacy and security* (pp. 89–100).

Cowperthwaite, D. J., Carpendale, S., & Fracchia, F. D. (1996). Visual access for 3d data. In *Conference companion on human factors in computing systems* (pp. 175–176). New York, NY, USA: ACM. doi: 10.1145/257089.257242

Cui, W., Wang, X., Liu, S., Riche, N. H., Madhyasta, T. M., Ma, K.-L., & Guo, B. (2014). Let It Flow : A Static Method to Explore Dynamic Graphs. In *Proceedings of pacificvis*. IEEE.

Cuthill, E., & McKee, J. (1969). Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 24th national conference* (pp. 157–172). New York, NY, USA: ACM. doi: 10.1145/800195.805928

Cytoscape: An open source platform for complex network analysis and visualization. (2013). http://www.cytoscape.org/. (online, accessed:17/02/2014)

Daniel, G., & Chen, M. (2003). Video visualization. In *Proceedings of the 14th IEEE conference on visualization* (pp. 54–). Washington, DC, USA: IEEE Computer Society. doi: 10.1109/VISUAL.2003.1250401

Demšar, U., & Virrantaus, K. (2010). Space–time density of trajectories: exploring spatio-temporal patterns in movement data. *International Journal of Geographical Information Science*, *24*(10), 1527-1542. Retrieved from http://www.tandfonline.com/doi/abs/10.1080/13658816.2010.511223 doi: 10.1080/13658816.2010.511223

Diehl, S., & Görg, C. (2002). Graphs, they are changing - dynamic graph drawing for a sequence of graphs. In M. Goodrich & S.G.Kobourov (Eds.), *Lncs* (Vol. 2528, pp. 23–31). Springer.

Diehl, S., Görg, C., & Kerren, A. (2001). Preserving the mental map using foresighted layout. In D. Ebert, J. Favre, & R. Peikert (Eds.), *Data visualization 2001* (p. 175-184). Springer Vienna. doi: 10.1007/978-3-7091-6215-6_19

Dinkla, K., Westenberg, M., & van Wijk, J. (2012). Compressed adjacency matrices: Untangling gene regulatory networks. *IEEE Transactions on Visualization and Computer Graphics*, *18*(12), 2457-2466. doi: 10.1109/TVCG.2012.208

Doar, M. (1996). A better Model for Generating Test Networks. In *Proceedings of global telecommunications conference* (pp. 86–93).

Doreian, P., Batagelj, V., & Ferligoj. (2005). *Generalized Blockmodeling*. Cambridge University Press.

Dosenbach, N., Nardos, B., Cohen, A., Fair, D., Power, J., Church, J., ... others (2010). Prediction of Individual Brain Maturity Using fMRI. *Science*, *329*(5997), 1358–1361. doi: http://dx.doi.org/10.1126/science.1194144

Dunne, C., Henry Riche, N., Lee, B., Metoyer, R., & Robertson, G. (2012). Graphtrail: Analyzing large multivariate, heterogeneous networks while supporting exploration history. In *Proceedings of the SIGCHI conference on human factors in computing*

systems (pp. 1663–1672). New York, NY, USA: ACM. doi: 10.1145/2207676.2208293

Dunne, C., & Shneiderman, B. (2013). Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 3247–3256). New York, NY, USA: ACM. doi: 10.1145/2470654.2466444

Dwyer, T. (2004). *Two-and-a-half-dimensional visualisation of relational networks*. Unpublished doctoral dissertation, University of Sydney.

Dwyer, T., & Eades, P. (2002). Visualising a fund manager flow graph with columns and worms. In *Proceedings of IV* (p. 147-152).

Dwyer, T., Lee, B., Fisher, D., Quinn, K. I., Isenberg, P., Robertson, G. G., & North, C. (2009). A comparison of user-generated and automatic graph layouts. *IEEE Transactions on Visusalization and Computer Graphics*, *15*(6), 961-968.

Eades, P., & Feng, Q.-W. (1997). Multilevel visualization of clustered graphs. In S. North (Ed.), *Graph drawing* (Vol. 1190, p. 101-112). Springer. doi: 10.1007/3-540-62495-3_41

Eades, P., & Hong, S.-H. (2013). Handbook of Graph Drawing and Visualization. In R. Tamassia (Ed.), (chap. Circular Drawing Algorithms). Chapman and Hall/CRC.

Eades, P., Lai, W., Misue, K., & Sugiyama, K. (1991). Preserving the mental map of a diagram. In *Proceedings of compugraphics* (pp. 24–33).

Eklund, J., Sawers, J., & Zeiliger, R. (1999). NESTOR Navigator: A Tool For The Collaborative Construction Of Knowledge Through Constructive Navigation. In *Proceedings of australian world wide web conference.*

Elmqvist, N., Do, T.-N., Goodell, H., Henry, N., & Fekete, J.-D. (2008). Zame: Interactive large-scale graph visualization. In *Proceedings of the IEEE pacific visualization symposium* (p. 215-222).

Elmqvist, N., Dragicevic, P., & Fekete, J.-D. (2008). Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis 2008)*, *14*(6), 1141-1148.

Elmqvist, N., & Fekete, J. (2010). Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. *IEEE Transactions on Visualization and Computer Graphics*, *16*(3), 439-454. doi: 10.1109/TVCG.2009.84

Elmqvist, N., Henry, N., Riche, Y., & Fekete, J.-D. (2008). Mélange: Space folding for multi-focus interaction. In *Proceedings of the ACM conference on human factors in computing systems* (p. 1333-1342).

Eppstein, D., & Wang, J. (2002). A Steady State Model for Graph Power Laws. *2nd International Workshop on Web Dynamics*.

Erdös, P., & Rényi, A. (1959). On Random Graphs. *Publicationes Mathematicae*, *6*, 290–297.

Erten, C., Harding, P., Kobourov, S. G., Wampler, K., & Yee, G. (2003). GraphAEL: Graph animations with evolving layouts. In *Proceedings of symposium on graph*

drawing (pp. 98–110). Springer.

Erten, C., Kobourov, S., Le, V., & Navabi, A. (2004). Simultaneous graph drawing: Layout algorithms and visualization schemes. In G. Liotta (Ed.), *Graph drawing* (Vol. 2912, p. 437-449). Springer. doi: 10.1007/978-3-540-24595-7_41

Everts, M. H., Bekker, H., Roerdink, J. B. T. M., & Isenberg, T. (2009). Depth-Dependent Halos: Illustrative Rendering of Dense Line Data. *IEEE TVCG*, *15*(6), 1299–1306. doi: http://doi.ieeecomputersociety.org/10.1109/TVCG.2009.138

Fabrikant, A., Koutsoupias, E., & Papadimitriou, C. H. (2002). Heuristically Optimized Trade-Offs: A New Paradigm for Power Laws in the Internet. In *Proceedings of colloquium on automata, languages and programming* (pp. 110–122). Springer, Berlin.

Falkowski, T., Bartelheimer, J., & Spiliopoulou, M. (2006). Mining and visualizing the evolution of subgroups in social networks. In *Proceedings of the 2006 ieee/wic/acm international conference on web intelligence* (pp. 52–58). Washington, DC, USA: IEEE Computer Society. doi: 10.1109/WI.2006.118

Farrugia, M., Hurley, N., & Quigley, A. (2011). Exploring temporal ego networks using small multiples and tree-ring layouts. In *Proceedings of achi.*

Farrugia, M., & Quigley, A. (2011). Effective temporal graph layout: A comporative stydy of animations versus statuc display methods. *Information Visualization*, 47–64.

Federico, P., Aigner, W., Miksch, S., Windhager, F., & Zenk, L. (2011). A visual analytics approach to dynamic social networks. In *Proceedings of the 11th international conference on knowledge management and knowledge technologies* (pp. 47:1–47:8). New York, NY, USA: ACM. doi: 10.1145/2024288.2024344

Federico, P., Pfeffer, J., Aigner, W., Miksch, S., & Zenk, L. (2012a). Visual Analysis of Dynamic Networks using Change Centrality. In *Proceedings of asonam* (p. 179 - 183).

Federico, P., Pfeffer, J., Aigner, W., Miksch, S., & Zenk, L. (2012b). Visual analysis of dynamic networks using change centrality. In *Advances in social networks analysis and mining (asonam), 2012 ieee/acm international conference on* (p. 179-183).

Fekete, J.-D., Wang, D., Dang, N., & Plaisant, C. (2003). Overlaying links on treemaps. *Compendium of IEEE Symposium on Information Visualization Conference*.

Feldman, J., & Tremoulet, P. D. (2006). Individuation of visual objects over time. *Cognition*, *99*(2), 131–165.

Fels, S., Lee, E., & Mase, K. (2000). Techniques for interactive video cubism (poster session). In *Proceedings of the eighth acm international conference on multimedia* (pp. 368–370). New York, NY, USA: ACM. doi: 10.1145/354384.354535

Frank, O., & Strauss, D. (1986). Markov Graphs. *Journal of the American Statistical Association*, *81*(395), 832–842.

Freire, M., Plaisant, C., Shneiderman, B., & Golbeck, J. (2010). ManyNets: An Interface for Multiple Network Analysis and Visualization. In *Proceedings of of chi* (pp. 213–222). New York, NY, USA: ACM. doi: 10.1145/1753326.1753358

Friedrich, C., & Eades, P. (2001). The marey graph animation tool demo. In *Proceedings of the 8th international symposium on graph drawing* (pp. 396–406). London, UK, UK: Springer.

Friedrich, C., & Eades, P. (2002). Graph Drawing in Motion. *Graph Algorithms and Applications*, *6*(3), 353–370.

Frishman, Y., & Tal, A. (2004). Dynamic drawing of clustered graphs. In *Proceedings of the IEEE symposium on information visualization* (pp. 191–198). Washington, DC, USA: IEEE Computer Society. doi: 10.1109/INFOVIS.2004.18

Frishman, Y., & Tal, A. (2007). Online dynamic graph drawing. In *Proceedings of the 9th joint eurographics / IEEE VGTC conference on visualization* (pp. 75–82). Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. doi: 10.2312/VisSym/EuroVis07/075-082

Fruchterman, T. M. J., & Reingold, E. M. (1991, November). Graph drawing by force-directed placement. *Softw. Pract. Exper.*, *21*(11), 1129–1164.

Fuchs, J., Fischer, F., Mansmann, F., Bertini, E., & Isenberg, P. (2013). Evaluation of alternative glyph designs for time series data in a small multiple setting. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 3237–3246). New York, NY, USA: ACM. doi: 10.1145/2470654.2466443

Furnas, G. (1986). Generalized Fisheye Views. In *Proceedings of ACM conference on human factors and information systems* (p. 16–23).

Gaertler, M., & Wagner, D. (2006). A hybrid model for drawing dynamic and evolving graphs. In P. Healy & N. Nikolov (Eds.), *Graph drawing* (Vol. 3843, p. 189-200). Springer. doi: 10.1007/11618058_18

Gansner, E., Koren, Y., & North, S. (2005). Topological fisheye views for visualizing large graphs. *IEEE Transactions on Visualization and Computer Graphics*, *11*(4), 457-468. doi: 10.1109/TVCG.2005.66

Gerhard, S., Daducci, A., Lemkaddem, A., Meuli, R., Thiran, J., & Hagmann, P. (2011). The Connectome Viewer Toolkit: An Open Source Framework to Manage, Analyze, and Visualize Connectomes. *Frontiers in Neuroinformatics*, *5*, 3:1–3:15. doi: http://dx.doi.org/10.3389/fninf.2011.00003

Ghani, S., & Elmqvist, N. (2011). Improving revisitation in graphs through static spatial features. In *Proceedings of graphics interface* (pp. 175–182). Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society.

Ghani, S., Riche, N. H., & Elmqvist, N. (2011). Dynamic insets for context-aware graph navigation. In *Proceedings of the 13th eurographics / IEEE- VGTC conference on visualization* (pp. 861–870). Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. doi: 10.1111/j.1467-8659.2011.01935.x

Ghoniem, M., Fekete, J.-D., & Castagliola, P. (2005). On the Readability of Graphs Using Node-Link and Matrix-Based Representations: Controlled Experiment and Statistical Analysis. In *Information Visualization Journal* (Vol. 4, p. 114–135). Palgrave Macmillan.

Ginestet, C., & Simmons, A. (2011). Statistical Parametric Network Analysis of Functional Connectivity Dynamics during a Working Memory Task. *NeuroImage*, *55*(2), 688–704. doi: http://dx.doi.org/10.1016/j.neuroimage.2010.11.030

Gleicher, M., Albers, D., Walker, R., Jusufi, I., Hansen, C., & Roberts, J. (2011). Visual Comparison for Information Visualization. *Information Visualization*, *10*(4), 289–309. doi: http://dx.doi.org/10.1177/1473871611416549

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning* (1st ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Goldman, R. I., Stern, J. M., Engel, J., & Cohen, M. S. (2000). Acquiring simultaneous EEG and functional MRI. *Clinical Neurophysiology*, *111*(11), 1974– 1980.

Görke, R., Kluge, R., Schumm, A., Staudt, C., & Wagner, D. (2012). An efficient generator for clustered dynamic random networks. In *Proceedings of the first mediterranean conference on design and analysis of algorithms* (pp. 219–233). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-34862-4_16

Gove, R., Gramsky, N., Kirby, R., Sefer, E., Sopan, A., Dunne, C., ... Taieb-Maimon, M. (2011). NetVisia: Heat Map and Matrix Visualization of Dynamic Social Network Statistics and Content. In *Proceedings of socialcom/passat* (p. 19-26).

Greilich, M., Burch, M., & Diehl, S. (2009). Visualizing the evolution of compound digraphs with TimeArcTrees. In *Proceedings of the 11th eurographics / IEEE - VGTC conference on visualization* (pp. 975–990). Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. doi: 10.1111/j.1467-8659.2009.01451.x

Gretchen, P. (2011). *A cartographer's toolkit - small multiples.* http://www.gretchenpeterson.com/blog/small-multiples. (online, accessed 24-Jan-2014)

Gruau, F. (1994). *Neural Network Synthesis using Cellular Encoding and Genetic Algorithms.* Unpublished doctoral dissertation, Laboratoire de l'Informatique du Parallelisme, Ecole Normale Superieure de Lyon, France.

Guilmaine, D., Viau, C., & McGuffin, M. J. (2012). Hierarchically Animated Transitions in Visualizations of Tree Structures. In *Proceedings of avi* (pp. 514–521). ACM.

HAC - a java class library for hierarchical agglomerative clustering. (2013). http://sape.inf.usi.ch/hac. ([=online, accessed:03/07/13])

Hadlak, S., Schulz, H., & Schumann, H. (2011). In Situ Exploration of Large Dynamic Networks. *IEEE Transactions on Visualization and Computer Graphics*, *17*(12), 2334-2343. doi: 10.1109/TVCG.2011.213

Hadlak, S., Schumann, H., Cap, C. H., & Wollenberg, T. (2013). Supporting the visual analysis of dynamic networks by clustering associated temporal attributes. *IEEE Transactions on Visualization and Computer Graphics*, *19*(12), 2267-2276. doi: http://doi.ieeecomputersociety.org/10.1109/TVCG.2013.198

Hagmann, P., Cammoun, L., Gigandet, X., Meuli, R., Honey, C., Wedeen, V., & Sporns, O. (2008). Mapping the Structural Core of Human Cerebral Cortex. *PLoS Biology*, *6*(7), e159. doi: http://dx.doi.org/10.1371/journal.pbio.0060159

Hascoët, M., & Dragicevic, P. (2012). Interactive graph matching and visual comparison of graphs and clustered graphs. In *Proceedings of international working conference on advanced visual interfaces (avi)* (p. 522-529). doi: 10.1145/2254556.2254654

Heer, J., & Card, S. K. (2004). DOITrees Revisited: Scalable, Space-Constrained Visualization of Hierarchical Data. In *Proceedings of the international working conference on advanced visual interfaces (avi)* (pp. 421–424). ACM.

Heer, J., Mackinlay, J., Stolte, C., & Agrawala, M. (2008). Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE Transactions on Visualization and Computer Graphics*, *14*(6), 1189-1196. doi: http://doi.ieeecomputersociety.org/10.1109/TVCG.2008.137

Heer, J., & Perer, A. (2011). Orion: A system for modeling, transformation and visualization of multidimensional heterogeneous networks. In *IEEE visual analytics science and technology (vast)*. Retrieved from http://vis.stanford.edu/papers/orion

Heer, J., & Robertson, G. (2007). Animated transitions in statistical data graphics. *IEEE Transactions on Visualization and computer graphics*, *13*, 1240–1247.

Heim, P., Ertl, T., & Ziegler, J. (2010). Facet graphs: Complex semantic querying made easy. In *Proceedings of the 7th international conference on the semantic web: Research and applications - volume part I* (pp. 288–302). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-13486-9_20

Helsgaun, K. (2000). An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational Research*, *126*, 106-130.

Henry, N., & Fekete, J.-D. (2006, September-October). MatrixExplorer: a Dual-Representation System to Explore Social Networks. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization 2006)*, *12*(5), 8 pages.

Henry, N., & Fekete, J.-D. (2007). MatLink: Enhanced matrix visualization for analyzing social networks. In *Proceedings of the 11th IFIP TC 13th international conference on human-computer interaction - volume part* (pp. 288–302). Berlin, Heidelberg: Springer.

Henry, N., Fekete, J.-D., & McGuffin, M. J. (2007, November). NodeTrix: a hybrid visualization of social networks. *IEEE Transactions on Visualization and Computer Graphics*, *13*(6), 1302–1309. doi: 10.1109/TVCG.2007.70582

Henry-Riche, N., & Dwyer, T. (2010). Untangling euler diagrams. *IEEE Transactions on Visualization and Computer Graphics*, *16*(6), 1090-1099. doi: 10.1109/TVCG.2010.210

Henry Riche, N., Dwyer, T., Lee, B., & Carpendale, S. (2012). Exploring the design space of interactive link curvature in network diagrams. In *Proceedings of the international working conference on advanced visual interfaces* (pp. 506–513). New York, NY, USA: ACM. doi: 10.1145/2254556.2254652

Herman, I., Melançon, G., & Marshall, M. S. (2000, January). Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on*

Visualization and Computer Graphics, *6*(1), 24–43. doi: 10.1109/2945.841119

Holme, P., & Saramäki, J. (2012). Temporal networks. *Physics Reports*, *519*(3), 97 - 125. (Temporal Networks) doi: http://dx.doi.org/10.1016/j.physrep.2012.03.001

Holten, D. (2006, September). Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, *12*(5), 741–748. doi: 10.1109/TVCG.2006.147

Holten, D., & van Wijk, J. (2009). A user study on visualizing directed edges in graphs. *Proceedings of the 27th ACM SIGCHI international conference on Human factors in computing systems (CHI)*, 2299–2308.

Honey, C., Kötter, R., Breakspear, M., & Sporns, O. (2007). Network Structure of Cerebral Cortex Shapes Functional Connectivity on Multiple Time Scales. *PNAS*, *104*(24), 10240–10245. doi: http://dx.doi.org/10.1073/pnas.0701519104

Hoppe, K., & Rodgers, G. J. (2013, Oct). Mutual selection in time-varying networks. *Phys. Rev. E*, *88*, 042804. doi: 10.1103/PhysRevE.88.042804

Huang, M. L., Eades, P., & Cohen, R. F. (1998, April). *WebOFDAV: Navigating and visualizing the web on-line with animated context swapping* (Vol. 30) (No. 1-7). Amsterdam, The Netherlands, The Netherlands: Elsevier Science Publishers B. V. doi: 10.1016/S0169-7552(98)00054-3

Huang, M. L., Eades, P., & Wang, J. (1998). On-line animated visualization of huge graphs using a modified spring algorithm. *Journal of Visual Languages Computing*, *9*(6), 623 - 645. doi: http://dx.doi.org/10.1006/jvlc.1998.0094

Hurter, C., Ersoy, O., Fabrikant, S., Klein, T., & Telea, A. (2013). Bundled visualization of dynamic graph and trail data. *IEEE Transactions on Visualization and Computer Graphics*(99). doi: 10.1109/TVCG.2013.246

Hurter, C., Ersoy, O., & Telea, A. (2012, June). Graph bundling by kernel density estimation. *Computer Graphics Forum*, *31*, 865–874. doi: 10.1111/j.1467-8659.2012.03079.x

Hurter, C., Tissoires, B., & Conversy, S. (2009). FromDaDy: Spreading Aircraft Trajectories Across Views to Support Iterative Queries. *IEEE Transactions on Visualization and Computer Graphics*, *15*(6), 1017-1024. doi: http://doi.ieeecomputersociety.org/10.1109/TVCG.2009.145

Hägerstrand, T. (1970). What about people in regional science? In (Vol. 24, p. 6-21). Springer. doi: 10.1007/BF01936872

Jansen, Y., & Dragicevic, P. (2013, Dec). An interaction model for visualizations beyond the desktop. *IEEE Transactions on Visualization and Computer Graphics*, *19*(12), 2396-2405. doi: 10.1109/TVCG.2013.134

Jianu, R., Demiralp, C., & Laidlaw, D. (2009). Exploring 3D DTI Fiber Tracts with Linked 2D Representations. *IEEE Transactions on Visualization and Computer Graphics*, *15*(6), 1449–1456. doi: http://dx.doi.org/10.1109/TVCG.2009.141

Johnson-Laird, P. N. (1983). *Mental models: towards a cognitive science of language, inference, and consciousness*. Cambridge, MA, USA: Harvard University Press.

Jónsson, K. V., Vigfússon, Y., & Helgason, O. (2012). Simulating large-scale dynamic random graphs in OMNeT++. In *Proceedings of the 5th international icst conference on simulation tools and techniques* (pp. 311–318). ICST, Brussels, Belgium, Belgium: Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (ICTS).

Kamada, T., & Kawai, S. (1989). An algorithm for drawing general undirected graphs. *Information Processing Letters*, *31*(1), 7 - 15. doi: http://dx.doi.org/10.1016/0020-0190(89)90102-6

Kapler, T., & Wright, W. (2004). Geotime information visualization. In *Proceedings of IEEE information visualization* (p. 25-32). Washington, DC, USA: IEEE Computer Society. doi: 10.1109/INFOVIS.2004.27

Khan, A., Mordatch, I., Fitzmaurice, G., Matejka, J., & Kurtenbach, G. (2008). Viewcube: A 3d orientation indicator and controller. In *Proceedings of the 2008 symposium on interactive 3d graphics and games* (pp. 17–25). New York, NY, USA: ACM. doi: 10.1145/1342250.1342253

Kids britannica. (2014). http://www.akira.ruc.dk/~keld/research/LKH. (online, accessed:3-Jul-2014)

Kitano, H. (1990). Designing Neural Networks Using Genetic Algorithms with Graph Generation System. *Complex Systems*, *4*, 461-476.

Kjellin, A., Pettersson, L. W., Seipel, S., & Lind, M. (2010). Different levels of 3D: An evaluation of visualized discrete spatiotemporal data in space-time cubes. *Information Visualization*, *9*(2), 152-164. doi: 10.1057/ivs.2009.8

Kleinberg, J. (2000). Navigation in a Small World - It is easier to find Short Chains between Points in some Networks than Others. *Nature*, *406*(6798), 845–845.

Kraak, M. J. (2003). The Space-Time Cube revisited from a Geovisualization Perspective. *Proceedings of the 21st International Cartographic Conference*, 1988-1996.

Kramer, M., Eden, U., Lepage, K., Kolaczyk, E., Bianchi, M., & Cash, S. (2011). Emergence of Persistent Networks in Long-Term Intracranial EEG Recordings. *Journal of Neuroscience*, *31*(44), 15757–15767. doi: http://dx.doi.org/10.1523/JNEUROSCI.2287-11.2011

Kristensson, P., Dahlback, N., Anundi, D., Bjornstad, M., Gillberg, H., Haraldsson, J., ... Stahl, J. (2009, July). An evaluation of space time cube representation of spatiotemporal patterns. *IEEE Transactions on Visualization and Computer Graphics*, *15*(4), 696-702. doi: 10.1109/TVCG.2008.194

Kveladze, I., & Kraak, M.-J. (2012). What do we know about the space-time cube from cartographic and usability perspective? In *Proceedings of the 2012 AutoCarto international symposium on automated cartography.*

Lam, H., Bertini, E., Isenberg, P., Plaisant, C., & Carpendale, S. (2012). Empirical studies in information visualization: Seven scenarios. *IEEE Transactions on Visualization and Computer Graphics*, *18*(9), 1520-1536. doi: 10.1109/TVCG.2011.279

Lamping, J., Rao, R., & Pirolli, P. (1995). A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI*

conference on human factors in computing systems (pp. 401–408). New York, NY, USA: ACM Press/Addison-Wesley Publishing Co. doi: 10.1145/223904.223956

Lee, B., Plaisant, C., Parr, C. S., Fekete, J.-D., & Henry, N. (2006). Task taxonomy for graph visualization. In *Proceedings of beliv workshop: Beyond time and errors: Novel evaluation methods for information visualization* (p. 1-5). ACM.

Lerman, K., Ghosh, R., & Kang, J. H. (2010). Centrality metric for dynamic networks. In *Proceedings of the eighth workshop on mining and learning with graphs* (pp. 70–77). New York, NY, USA: ACM.

Li, K., Guo, L., Faraco, C., Zhu, D., Chen, H., Yuan, Y., ... others (2012). Visual Analytics of Brain Networks. *NeuroImage*, *61*(1), 82–97. doi: http://dx.doi.org/10.1016/j.neuroimage.2012.02.075

Liiv, I. (2010). Seriation and matrix reordering methods: An historical overview. *Statistical Analysis and Data Mining*, *3*(2), 70–91. doi: 10.1002/sam.10071

Liu, Z., Navathe, S., & Stasko, J. (2011). Ploceus: Network-based visual analysis of tabular data. *IEEE Conference onVisual Analytics Science and Technology*, 41-50. doi: 10.1109/VAST.2011.6102440

Liu, Z., & Stasko, J. (2010, November). Mental models, visual reasoning and interaction in information visualization: A top-down perspective. *IEEE Transactions on Visualization and Computer Graphics*, *16*(6), 999–1008. doi: 10.1109/TVCG.2010.177

LKH. (2012). http://kids.britannica.com/elementary/art-163974/The-borders-of-Yugoslavia-changed-greatly-during-the-1900s. (online, accessed:20-Aug-2013)

MacEachren, A. M. (1995). *How Maps Work*. Guilford Press.

Mackay, W., & Beaudouin-Lafon, M. (1998, apr). DIVA: Exploratory data analysis with multimedia streams. In *Proceedings of ACM SIGCHI confernece on human factors in computing systems (chi)* (p. 416-423). ACM.

Marey, É.-J. (1878). *La méthode graphique dans les sciences expérimentales et particulièrement en physiologie et en médecine*. G. Masson.

McCloud, S. (1994). *Understanding comics*. HarperCollins.

McFarland, D., & Bender-deMoll, S. (n.d.). *SoNIA - Social Network Image Animator*. http://www.stanford.edu/group/sonia/. (online, accessed:24-Jan-2014)

McGuffin, M. J., Tancau, L., & Balakrishnan, R. (2003, October). Using deformations for browsing volumetric data. In *Proceedings of IEEE conference on visualization (VIS) 2003* (pp. 401–408).

Medina, A., Matta, I., & Byers, J. (2000, April). On the Origin of Power Laws in Internet Topologies. *ACM SIGCOMM Computer Communication Review*, *30*(2), 18–28. doi: 10.1145/505680.505683

Melançon, G. (2006). Just how dense are dense graphs in the real world?: A methodological note. In *Proceedings of the 2006 avi workshop on beyond time and errors: Novel evaluation methods for information visualization* (pp. 1–7). New York, NY,

USA: ACM. doi: 10.1145/1168149.1168167

Misue, K., Eades, P., Lai, W., & Sugiyama, K. (1995). Layout adjustment and the mental map. *Journal of Visual Languages Computing*, *6*(2), 183 - 210. doi: http://dx.doi.org/10.1006/jvlc.1995.1010

Moberts, B., Vilanova, A., & van Wijk, J. (2005). Evaluation of Fiber Clustering Methods for Diffusion Tensor Imaging. In *Proceedings of the conference on visualization (vis)* (pp. 65–72). IEEE. doi: http://dx.doi.org/10.1109/VISUAL.2005.1532779

Moody, J. (2001). Peer influence groups: identifying dense clusters in large networks. *Social Networks*, *23*, 261–283.

Moody, J., McFarland, D. A., & Bender-DeMoll, S. (2005). Dynamic Network Visualization: Methods for Meaning with Longitudinal Network Movies. *American Journal of Sociology*(110), 1206-1241.

Moscovich, T., Chevalier, F., Henry, N., Pietriga, E., & Fekete, J.-D. (2009, April). Topology-Aware Navigation in Large Networks. In *Proceedings of the 27th ACM SIGCHI international conference on Human factors in computing systems (CHI))* (p. 2319–2328). New York, NY, USA: ACM.

Mrvar, A., & Batagelj, V. (2004). *Analysis and visualization of temporal networks using pajek.* http://vlado.fmf.uni-lj.si/pub/networks/doc/seminar/SFI-SAS04tn.pdf. (online, accessed:24-Jan-2014)

Munzner, T. (1998, July). Exploring large graphs in 3D hyperbolic space. *IEEE Computer Graphics and Applications*, *18*(4), 18–23. doi: 10.1109/38.689657

Munzner, T. (2009, November). A nested model for visualization design and validation. *IEEE Transactions on Visualization and Computer Graphics*, *15*(6), 921–928. doi: 10.1109/TVCG.2009.111

Nelson, S., Cohen, A., Power, J., Wig, G., Miezin, F., Wheeler, M., ... Petersen, S. (2010). A Parcellation Scheme for Human Left Lateral Parietal Cortex. *Neuron*, *67*(1), 156–170. doi: http://dx.doi.org/10.1016/j.neuron.2010.05.025

Neumann, P., Schlechtweg, S., & Carpendale, S. (2005). ArcTrees: Visualizing relations in hierarchical data. In *Proceedings of the 7th joint eurographics / IEEE VGTC conference on visualization* (pp. 53–60). Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. doi: 10.2312/VisSym/EuroVis05/053-060

Nicosia, V., Tang, J., Mascolo, C., Musolesi, M., Russo, G., & Latora, V. (2013). Graph metrics for temporal networks. In P. Holme & J. Saramäki (Eds.), *Temporal networks* (p. 15-40). Springer. doi: 10.1007/978-3-642-36461-7_2

Noack, A. (2007). Energy models for graph clustering. *J. Graph Algorithms Appl.*, *11*(2), 453-480.

NodeXL: Network overview, discovery and exploration for excel. (n.d.). nodexl.codeplex.com. (online, acessed:24-Jan-2014)

North, C. (2006, May). Toward measuring visualization insight. *IEEE Comput. Graph. Appl.*, *26*(3), 6–9. doi: 10.1109/MCG.2006.70

North, S. C. (1996). Incremental layout in DynaDAG. In *Proceedings of the symposium on graph drawing (GD)* (pp. 409–418). Springer.

O'Madadhain, J., Fisher, D., & Nelson, T. (2003). *JUNG - Java Universal Network/Graph Framework.* http://jung.sourceforge.net/. (online, accessed:24-Jan-2014)

Pajek - Program for Large Network Analysis. (1997). http://pajek.imfm.si/doku.php. (online, acessed:20-Dec-2013])

Perer, A., & Sun, J. (2012). MatrixFlow: temporal network visual analytics to track symptom evolution during disease progression. In *Proceedings of the aima annual symposium.* American Medical Informatics Association (AIMA).

Peterson, E. (2011). Time spring layout for visualization of dynamic social networks. In *Proceedings of the IEEE network science workshop (NSW)* (pp. 98–104). IEEE.

Petrovic, N., Jojic, N., & Huang, T. S. (2005). Adaptive video fast forward. *Multimedia Tools and Applications*, *26*(3), 327–344.

Peuquet, D. J. (1994). Its about time - a conceptual-framework for the representation of temporal dynamics in geographic information-systems. *Annals of the Association of American Geographers*, *84*(3), 441-461.

Pietriga, E. (2005). A toolkit for addressing HCI issues in visual language environments. In *IEEE symposium on visual languages and human-centric computing (vl/hcc)* (p. 145-152). IEEE.

Pietriga, E., Cubaud, P., Schwarz, J., Primet, R., Schilling, M., Barkats, D., ... Vila Vilaro, B. (2012). Interaction Design Challenges and Solutions for ALMA Operations Monitoring and Control. In *Proceedings of the conference on astronomical telescopes and instrumentes.* SPIE. doi: 10.1117/12.925180

Plaisant, C. (2004). The challenge of information visualization evaluation. In *Proceedings of the working conference on advanced visual interfaces* (pp. 109–116). New York, NY, USA: ACM. doi: 10.1145/989863.989880

Plaisant, C., Grosjean, J., & Bederson, B. (2002). SpaceTree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation. In *Proceedings of infovis* (pp. 57–64).

Pohl, M., Reitz, F., & Birke, P. (2008). As time goes by: integrated visualization and analysis of dynamic networks. In *Proceedings of the working conference on advanced interfaces (avi)* (pp. 372–375). ACM.

Pretorius, A. J., & Wijk, J. J. V. (2006). Visual analysis of multivariate state transition graphs. *IEEE Transactions on Visualization and Computer Graphics*, *12*(5), 685-692. doi: http://doi.ieeecomputersociety.org/10.1109/TVCG.2006.192

Purchase, H. C. (1997). Which aesthetic has the greatest effect on human understanding? In G. DiBattista (Ed.), *Graph drawing* (Vol. 1353, p. 248-261). Springer. doi: 10.1007/3-540-63938-1_67

Purchase, H. C. (2000). Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interacting with Computers*, *13*(2), 147-162. doi: 10.1016/S0953-5438(00)00032-1

Purchase, H. C. (2002). Metrics for Graph Drawing Aesthetics. *Journal of Visual Languages Computing*, *13*(5), 501 - 516. doi: http://dx.doi.org/10.1006/jvlc.2002

.0232

Purchase, H. C., Carrington, D., & Allder, J.-A. (2002). Empirical evaluation of aesthetics-based graph layout. *Empirical Software Engineering*, *7*(3), 233–255.

Purchase, H. C., Hoggan, E., & Görg, C. (2007). How important is the "mental map"? — an empirical investigation of a dynamic graph layout algorithm. In M. Kaufmann & D. Wagner (Eds.), *Graph drawing* (Vol. 4372, p. 184-195). Springer.

Purchase, H. C., & Samra, A. (2008). Extremes are better: Investigating mental map preservation in dynamic graphs. In *Proceedings of the 5th international conference on diagrammatic representation and inference* (pp. 60–73). Springer.

Reda, K., Tantipathananandh, C., Johnson, A., Leigh, J., & Berger-Wolf, T. (2011). Visualizing the evolution of community structures in dynamic social networks. In *Proceedings of the 13th eurographics / IEEE- VGTC conference on visualization* (pp. 1061–1070). Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. doi: 10.1111/j.1467-8659.2011.01955.x

Rensink, R. A. (2002). Change detection. *Annual review of psychology*, *53*(1), 245–277.

Richiardi, J., Eryilmaz, H., Schwartz, S., Vuilleumier, P., & Van De Ville, D. (2011). Decoding Brain States from fMRI Connectivity Graphs. *NeuroImage*, *56*(2), 616–626. doi: http://dx.doi.org/10.1016/j.neuroimage.2010.05.081

Roberts, J. C. (2007). State of the art: Coordinated & multiple views in exploratory visualization. In *Proceedings of the international conference on coordinated and multiple views in exploratory visualization* (pp. 61–71). Washington, DC, USA: IEEE Computer Society. doi: 10.1109/CMV.2007.20

Robertson, G., Card, S. K., & Mackinlay, J. D. (1993, April). Information visualization using 3D interactive animation. *Communications of the ACM - Special issue on graphical user interfaces*, *36*(4), 57–71. doi: 10.1145/255950.153577

Robertson, G., Fernandez, R., Fisher, D., Lee, B., & Stasko, J. (2008, November). Effectiveness of animation in trend visualization. *IEEE Transactions on Visualization and Computer Graphics*, *14*(6), 1325–1332.

Rosling, H. (2006). *Stats that reshape your worldview.* http://www.ted.com/talks/hans_rosling_the_good_news_of_the_decade.html. (online, accessed:24-jan-2014)

Rossling, H. (2007). *GapMinder.* http://www.gapminder.org/. (online, accessed:24-Jan-2014)

Royer, L., Reimann, M., Andreopoulos, B., & Schroeder, M. (2008, 07). Unraveling protein networks with power graph analysis. *PLoS Comput Biol*, *4*(7), e1000108. doi: 10.1371/journal.pcbi.1000108

Rufiange, S., & McGuffin, M. (2013). Diffani: Visualizing dynamic graphs with a hybrid of difference maps and animation. *IEEE Transactions on Visualization and Computer Graphics*, *19*(12), 2556-2565. doi: 10.1109/TVCG.2013.149

Saffrey, P., & Purchase, H. C. (2008). The "mental map" versus "static aesthetic" compromise in dynamic graphs: a user study. In *Proceedings of the 9th conference on australasian user interface - volume 76* (pp. 85–93). Australian Computer

Society, Inc.

Sallaberry, A., Muelder, C., & Ma, K.-L. (2013). Clustering, visualizing, and navigating for large dynamic graphs. In *Proceedings of the 20th international conference on graph drawing* (pp. 487–498). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-36763-2_43

Sanz-Arigita, E., Schoonheim, M., Damoiseaux, J., Rombouts, S., Maris, E., Barkhof, F., ... Stam, C. (2010). Loss of 'Small-World' Networks in Alzheimer's Disease: Graph Analysis of fMRI Resting-State Functional Connectivity. *PloS one*, *5*(11), e13788. doi: http://dx.doi.org/10.1371/journal.pone.001378

Saraiya, P., North, C., & Duca, K. (2010). Comparing benchmark task and insight evaluation methods on timeseries graph visualizations. In *Proceedings of the 3rd BELIV workshop: BEyond Time and Errors: Novel evaluation methods for information visualization* (pp. 55–62). New York, NY, USA: ACM. doi: 10.1145/2110192.2110201

Sarkar, M., & Brown, M. H. (1992). Graphical fisheye views of graphs. In *Proceedings of the ACM SIGCHI conference on human factors in computing systems* (pp. 83–91). New York, NY, USA: ACM. doi: 10.1145/142750.142763

Schaffer, D., Zuo, Z., Greenberg, S., Bartram, L., Dill, J., & Roseman, M. (1998). Navigating hierarchically clustered networks through fisheye and full-zoom methods. *ACM Transactions on Computer-Human Interaction*, *3*, 162–188.

Scholz, J., Klein, M., Behrens, T., & Johansen-Berg, H. (2009). Training Induces Changes in White-Matter Architecture. *Nature Neuroscience*, *12*(11), 1370–1371. doi: http://dx.doi.org/10.1038/nn.2412

Schreiber, F. (2003). Comparison of Metabolic Pathways using Constraint Graph Drawing. In *Proceedings of the 1st asia-pacific bioinformatics conference on bioinformatics (apbc)* (pp. 105–110). Australian Computer Society, Inc.

Seoung, S. (2012). *The Connectome - How Brain's Wiring makes us Who We are*. New York, NY: Houghton Mifflin Harcourt Publishing Company.

Shanbhag, P., Rheingans, P., & des Jardins, M. (2005). Temporal visualization of planning polygons for efficient partitioning of geo-spatial data. In *Proceedings of the proceedings of the 2005 IEEE symposium on information visualization* (pp. 28–). Washington, DC, USA: IEEE Computer Society. doi: 10.1109/INFOVIS.2005.32

Shen, Z., Ma, K.-L., & Eliassi-Rad, T. (2006, November). Visual analysis of large heterogeneous social networks by semantic and structural abstraction. *IEEE Transactions on Visualization and Computer Graphics*, *12*(6), 1427–1439. doi: 10.1109/TVCG.2006.107

Sheny, Z., & Maz, K.-L. (2007). Path visualization for adjacency matrices. In *Proceedings of the 9th joint eurographics / IEEE VGTC conference on visualization* (pp. 83–90). Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. doi: 10.2312/VisSym/EuroVis07/083-090

Sherbondy, A., Akers, D., Mackenzie, R., Dougherty, R., & Wandell, B. (2005). Exploring Connectivity of the Brain's White Matter with Dynamic Queries. *IEEE TVCG*,

11(4), 419–430. doi: http://dx.doi.org/10.1109/TVCG.2005.59

Shi, L., Wang, C., & Wen, Z. (2011). Dynamic network visualization in 1.5D. In *Proceedings of the 2011 IEEE pacific visualization symposium* (pp. 179–186). Washington, DC, USA: IEEE Computer Society.

Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE symposium on visual languages* (pp. 336–). Washington, DC, USA: IEEE Computer Society.

Shneiderman, B. (2003, November). Why not make interfaces better than 3D reality? *IEEE Computer Graphics and Applications*, *23*(6), 12–15. doi: 10.1109/MCG.2003.1242376

Shneiderman, B., & Aris, A. (2006). Network visualization by semantic substrates. *IEEE Transactions on Visualization and Computer Graphics*, *12*(5), 733-740. doi: 10.1109/TVCG.2006.166

Six, J. M., & Tollis, I. G. (2013). Handbook of Graph Drawing and Visualization. In R. Tamassia (Ed.), (chap. Symmetric Graph Drawing). Chapman and Hall/CRC.

Slocum, T. A., Robert S. Sluter, J., Kessler, F. C., & Yoder, S. C. (2004). Qualitative evaluation of MapTime, a program for exploring spatiotemporal point data. *Cartographica: The International Journal for Geographic Information and Geovisualization*, *39*, 43-68.

Smith, G., Stuerzlinger, W., Salzman, T., Watson, B., & Buchanan, J. (2001). 3d scene manipulation with 2d devices and constraints. In *Graphics interface* (Vol. 1, pp. 135–142).

Sole, R., & Valverde, S. (2004). Information Theory of Complex Networks: On Evolution and Architectural Constraints. In E. Ben-Naim, H. Frauenfelder, & Z. Toroczkai (Eds.), *Complex networks* (Vol. 650, p. 189-207). Springer. doi: 10.1007/978-3-540-44485-5_9

Sporns, O. (2011). *Networks of the brain*. Cambridge, Massachusetts: MIT Press.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, *10*(2), 99–127.

Stein, K., Wegener, R., & Schlieder, C. (2010). Pixel-oriented visualization of change in social networks. *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, *6*, 233-240. doi: http://doi.ieeecomputersociety.org/10.1109/ASONAM.2010.18

Suchorzewski, M. (2011). Evolving Scalable and Modular Adaptive Networks with Developmental Symbolic Encoding. *Evolutionary Intelligence*, *4*, 145-163.

Sugiyama, K., Tagawa, S., & Toda, M. (1981). Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, *11*(2), 109-125. doi: 10.1109/TSMC.1981.4308636

Tamassia, R. (Ed.). (2013). *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC.

Tang, A., Greenberg, S., & Fels, S. (2008). Exploring video streams using slit-tear visualizations. In *Proceedings of the working conference on advanced visual*

interfaces (pp. 191–198). New York, NY, USA: ACM. doi: 10.1145/1385569.1385601

Thomason, M., Dennis, E., Joshi, A., Joshi, S., Dinov, I., Chang, C., ... Gotlib, I. H. (2011). Resting-State fMRI Can Reliably Map Neural Networks in Children. *NeuroImage*, *55*(1), 165–175. doi: http://dx.doi.org/10.1016/j.neuroimage.2010.11.080

Todd, J. T. (2004). The visual perception of 3D shape. *Trends in cognitive sciences*, *8*(3), 115–121.

Tominski, C., Abello, J., van Ham, F., & Schumann, H. (2006). Fisheye tree views and lenses for graph visualization. In *Proceedings of the IEEE conference on information visualization* (pp. 17–24). Washington, DC, USA: IEEE. doi: 10.1109/IV.2006.54

Truong, B. T., & Venkatesh, S. (2007). Video abstraction: A systematic review and classification. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, *3*(1), 3.

Tufte, E. R. (1986). *The visual display of quantitative information*. Cheshire, CT, USA: Graphics Press.

Tufte, E. R. (1990). *Envisioning information*. Graphics Press.

Tukey, J. W. (1993). *Exploratory Data Analysis: Past, Present, and Future* (Tech. Rep.). Princton, NJ: Princton University, Department of Statistics.

Turdukulov, U. D., Kraak, M.-J., & Blok, C. A. (2007). Designing a visual environment for exploration of time series of remote sensing data: In search for convective clouds. *Computers and Graphics*, *31*(3), 370–379.

Tversky, B., Morrison, J. B., & Betrancourt, M. (2002). Animation: Can it Facilitate? *International Journal of Human-Computer Studies*, *57*(4), 247-262.

van den Elzen, S., Holten, D., Blaas, J., & van Wijk, J. (2013). Dynamic Network Visualization with Extended Massive Sequence Views. *IEEE Transactions on Visualization and Computer Graphics*, *PP*(99), 1-1. doi: 10.1109/TVCG.2013.263

van Ham, F., & Perer, A. (2009, November). Search, Show Context, Expand on Demand: Supporting Large Graph Exploration with Degree-of-Interest. *IEEE Transactions on Visualization and Computer Graphics*, *15*(6), 953–960. doi: 10.1109/TVCG.2009.108

van Ham, F., Schulz, H.-J., & Dimicco, J. M. (2009). Honeycomb: Visual analysis of large scale social networks. In *Proceedings of the 12th IFIP TC international conference on human-computer interaction: Part ii* (pp. 429–442). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-03658-3_47

van Ham, F., & van Wijk, J. (2004). Interactive visualization of small world graphs. In *Information visualization, 2004. infovis 2004. IEEE symposium on* (p. 199-206). doi: 10.1109/INFVIS.2004.43

van Wijk, J. J. (2008). Unfolding the earth: Myriahedral projections. *The Cartographic Journal*, *45*(1), 32-42. doi: 10.1179/000870408X276594

Van Wijk, J. J., & Van Selow, E. R. (1999). Cluster and calendar based visualization of time series data. In *Proceedings of the 1999 IEEE symposium on information visualization* (p. 4-9). Washington, DC, USA: IEEE Computer Society.

Vehlow, C., Reinhardt, T., & Weiskopf, D. (2013). Visualizing fuzzy overlapping communities in networks. *IEEE Transactions on Visualization and Computer Graphics*, *19*(12), 2486-2495. doi: 10.1109/TVCG.2013.232

Vidal, J., Freyermuth, S., Jerbi, K., Hamamé, C., Ossandon, T., Bertrand, O., ... Lachaux, J. (2012). Long-Distance Amplitude Correlations in the High Gamma Band Reveal Segregation and Integration within the Reading Network. *The Journal of Neuroscience*, *32*(19), 6421–6434. doi: http://dx.doi.org/10.1523/JNEUROSCI.4363-11.2012

Viégas, F. B., Wattenberg, M., & Dave, K. (2004). Studying cooperation and conflict between authors with history flow visualizations. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 575–582). New York, NY, USA: ACM. doi: 10.1145/985692.985765

von Landesberger, T., Kuijper, A., Schreck, T., Kohlhammer, J., van Wijk, J., Fekete, J.-D., & Fellner, D. (2011). Visual analysis of large graphs: State-of-the-art and future research challenges. *Computer Graphics Forum*, *30*(6), 1719–1749. doi: 10.1111/j.1467-8659.2011.01898.x

Vrotsou, K., Forsell, C., & Cooper, M. (2010, December). 2D and 3D representations for feature recognition in time geographical diary data. *Information Visualization*, *9*(4), 263–276. doi: 10.1057/ivs.2009.30

Walshaw, C. (2001). A multilevel algorithm for force-directed graph drawing. In J. Marks (Ed.), *Graph drawing* (Vol. 1984, p. 171-182). Springer. doi: 10.1007/3-540-44541-2_17

Wanger, L. R., Ferwerda, J., & Greenberg, D. P. (1992). Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications*, *12*(3), 44–58.

Ware, C. (2004). *Information visualization: Perception for design* (2nd ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Wattenberg, M. (2002). Arc diagrams: Visualizing structure in strings. In *Proceedings of the IEEE symposium on information visualization (infovis'02)* (pp. 110–). Washington, DC, USA: IEEE Computer Society.

Wattenberg, M. (2006). Visual exploration of multivariate graphs. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 811–819). New York, NY, USA: ACM. doi: 10.1145/1124772.1124891

Watts, D., & Strogatz, S. (1998). Collective Dynamics of 'Small-World' Networks. *Nature*, *393*(6684), 440–442.

Waxman, B. (1988). Routing of Multipoint Connections. *Journal on Selected Areas in Communications*, *6*(9), 1617–1622.

Wikimedia. (2013). *Stroke order project.* http://www.tinyurl.com/strokeOrderProject. (online; accessed 24-Jan-2014)

Willems, N., van de Wetering, H., & van Wijk, J. J. (2009). Visualization of vessel movements. In *Proceedings of the 11th eurographics / IEEE- VGTC conference on visualization* (pp. 959–966). Aire-la-Ville, Switzerland: Eurographics Association. doi: 10.1111/j.1467-8659.2009.01440.x

Willems, N., van de Wetering, H., & van Wijk, J. J. (2011). Evaluation of the visibility of vessel movement features in trajectory visualizations. In *Proceedings of the 13th eurographics / IEEE- VGTC conference on visualization* (pp. 801–810). Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. doi: 10.1111/j.1467-8659.2011.01929.x

Wong, P. C., Foote, H., Mackey, P., Perrine, K., & Chin Jr., G. (2006). Generating Graphs for Visual Analytics through Interactive Sketching. *IEEE Transactions on Visualization and Computer Graphics*, *12*(6), 1386–1398. doi: 10.1109/TVCG.2006.91

Worsley, K., Chen, J., Lerch, J., & Evans, A. (2005). Comparing Functional Connectivity via Thresholding Correlations and Singular Value Decomposition. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *360*(1457), 913–920. doi: http://dx.doi.org/10.1098/rstb.2005.1637

Xu, Z., Ke, Y., Wang, Y., Cheng, H., & Cheng, J. (2012). A model-based approach to attributed graph clustering. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data* (pp. 505–516). New York, NY, USA: ACM. doi: 10.1145/2213836.2213894

yEd Graph Editor. (2012). http://www.yworks.com/en/products_yed_about.html. (online, accessed:12-Feb-2014)

Yee, K.-P., Fisher, D., Dhamija, R., & Hearst, M. (2001). Animated exploration of dynamic graphs with radial layout. In *Proceedings of the IEEE symposium on information visualization 2001 (infovis'01)* (p. 43-50). Washington, DC, USA: IEEE Computer Society.

Yeger-Lotem, E., Sattath, S., Kashtan, N., Itzkovitz, S., Milo, R., Pinter, R. Y., ... Margalit, H. (2004). Network motifs in integrated cellular networks of transcription–regulation and protein–protein interaction. *Proceedings of the National Academy of Sciences of the United States of America*, *101*(16), 5934-5939. doi: 10.1073/pnas.0306752101

Yi, J.-S., Elmqvist, N., & Lee, S. (2010). TimeMatrix: Visualizing Temporal Social Networks Using Interactive Matrix-Based Visualizations. In *International journal of human-computer interaction* (Vol. 26, p. 1031-1051).

Yi, J.-S., Kang, Y. a., Stasko, J., & Jacko, J. (2007, November). Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, *13*(6), 1224–1231. doi: 10.1109/TVCG.2007.70515

Zaman, L., Kalra, A., & Stuerzlinger, W. (2011). The effect of animation, dual view, difference layers, and relative re-layout in hierarchical diagram differencing. In *Proceedings of graphics interface* (pp. 183–190).

Zhou, Y., Cheng, H., & Yu, J. X. (2010). Clustering large attributed graphs: An efficient incremental approach. *Data Mining, IEEE International Conference on*, *0*, 689-698. doi: http://doi.ieeecomputersociety.org/10.1109/ICDM.2010.41

Ziegler, J., Kunz, C., & Botsch, V. (2002). Matrix browser: Visualizing and exploring large networked information spaces. In *Extended abstracts on ACM SIGCHI conference on human factors in computing systems* (pp. 602–603). New York, NY, USA: ACM. doi: 10.1145/506443.506504

List of Figures

List of Tables