UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

Augmenting Hierarchical Visualizations with Topology-Centric Representations and Interactions

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR of PHILOSOPHY

By OMKAR CHEKURI

Norman, Oklahoma

2025

Augmenting Hierarchical Visualizations
with Topology-Centric Representations and Interactions

A DISSERTATION APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY THE COMMITTEE CONSISTING OF

Dr. Chris Weaver, Chair

Dr. Dean Hougen

Dr. Chongle Pan

Dr. Ghulam Jilani Quadri

Dr. Ziho Kang

Acknowledgements

I would like to express my deepest gratitude to everyone who has supported me, both directly and indirectly, throughout this challenging and rewarding journey toward my dissertation.

First and foremost, I am immensely grateful to my advisor, Dr. Chris Weaver. Since our first meeting in the fall of 2019, his unwavering support and guidance have been the cornerstone of my academic and personal growth. Dr. Weaver has witnessed the highs and lows of my journey from the challenges presented by the COVID-19 pandemic. Throughout these experiences, his constant patience and support have been instrumental in shaping me as a scholar. I value most the lessons I learned about the fundamental approach to research, maintaining integrity, and being persistent and passionate about taking on bigger challenges. As a first-generation doctoral student, I could not have asked for more.

I would like to thank Dr. Sridhar Radhakrishnan for his advice and support in all aspects of life during my time at the University of Oklahoma. Dr. Sridhar motivated me to pursue my PhD while serving as my graduate advisor during my Master's degree.

I would like to thank the members of my dissertation committee: Dr. Dean Hougen, Dr. Ziho Kang, Dr. Chongle Pan, and Dr. Ghulam Jilani Quadri. Your expertise and thoughtful feedback have been essential in refining this work. I have known Dr. Ziho Kang since my Master's at OU, where I had the opportunity to enroll in his courses and work directly with him. My first introduction to research in human-computer interaction was nurtured during my cognitive engineering course, and his academic advice, care, and encouragement have had a significant impact. I have known Dr. Hougen since I began my PhD and could always rely on his input and career guidance. I received valuable insights from Dr. Pan during the initial stages of my PhD, and I got to know Dr. Quadri in the later stages; his contagious energy and enthusiasm, along with his support during my dissertation write-up,

were immensely important. I would also like to thank Dr. Christan Grant for serving as a committee member during his time at OU and providing useful feedback.

I am deeply grateful to my family whose patience, support, and encouragement were invaluable throughout my PhD journey. Their unwavering belief in me provided the strength I needed to navigate this challenging path. I especially appreciate their help in managing my responsibilities while I pursued my goals.

My sincere appreciation goes to my PhD colleagues, Yonathan Hendrawan and Dr. Chenguang Xu, whose camaraderie and collaborative spirit enriched my research and graduate study experience. Your insightful discussions, constructive critiques, and mutual support have been a constant source of inspiration and motivation. I had the opportunity to know Yonathan very well; we shared countless hours discussing various topics and advising each other on navigating the challenges of a PhD. Dr. Chenguang Xu's advice on research and career, as a senior graduate student and after, has been invaluable.

I would also like to give special thanks to my friend, Dr. Kanneganti Sai Teja, who was not only my roommate but also a constant source of motivation, always available to offer advice. His encouragement and support have helped me stay focused both academically and professionally. I also appreciate my friend Dr. Xuxin Chen for the many conversations about various aspects of life and for his support of my graduate studies. Finally, I thank the staff of the School of Computer Science for ensuring I was well taken care of during my time at the University of Oklahoma.

Each of you has played a vital role in this journey, and I am profoundly grateful for your contributions. There are many others who have supported me throughout my time at the University. This dissertation is not only a reflection of my work but also a testament to the collective effort, guidance, and support I have received along the way. I look forward to nurturing these relationships moving forward.

Table of Contents

List of Tables

List of Figures

Abstract

As Tufte emphasizes, "above all else, show the data" [131]. Consistently highlighting key aspects of the data in visualizations ensures a more reliable, quick, and insightful understanding of data, particularly when interpreting various aspects of the data. This is particularly applicable for hierarchical data, which have many facets of relational information. A hierarchy is a system that organizes entities by rank, order, or importance, using parent-child relationships at its foundation. These relationships often result in derivation of various other kinds of relationships within the hierarchy, some of which are widely applicable and considered of primary importance across domains, while others are domain-specific and of special importance. Examples of these relationships include siblings, levels and paths. The nodes and edges that comprise these relationships are often collectively referred to as *topologies* by the visualization community [80, 92]. In our research, we further distinguish the topologies of single node and edge elements, referred to as *basic topologies* and those formed by sets of nodes and edges, which we term *composite topologies*. Having the ability to explicitly represent composite topologies and provide support for interactive operations on them is useful for revealing patterns, facilitating analysis, and drawing meaningful conclusions from hierarchical data.

Tree visualizations are a subclass of hierarchical visualizations designed to represent data with strict parent-child relationships in a tree structure. These techniques emphasize visual clarity in conveying hierarchical depth and structural relationships. Existing tree visualization techniques and systems primarily focus on representing node entities and parent-child relationships between node entities, and to some extent supporting operations on nodes and edges. These visualizations often reveal various other composite topologies incidentally through their design. Some relationships, like parent-child relationships, are easier to

interpret in these visualizations than composite topologies because they are explicitly represented. Other relationships represented by composite topologies require deliberate effort by users to interpret their structure from what they see. Explicit representation of composite topologies can allow users to see those topologies directly rather than discerning them from their constituent node and edge representations, opening new avenues for interactive data exploration and analysis of hierarchies.

This research explores how existing tree visualizations represent information and how these visualizations can be extended to support representation of composite topologies and interactive operations on them. We examine the concept of *topologies*: the structures in a hierarchy, comprising nodes and edges, that constitute various relations. We develop a software architecture and a data pipeline to support tree visualization design, along with an implemented tree visualization system called Hieros. We conduct a user evaluation to study the usability and utility of representing various relations and interacting with them in tree visualizations.

This dissertation makes five main research contributions. The first contribution is *C4D3* [31], a JavaScript-based coordination library for building coordinated multiple view visualizations using the D3 [24] library for web-based visualizations. It is based on the Live Properties architecture of the Improvise visualization system [138]. C4D3 aims to adapt and implement the coordination architecture of Improvise as an independent library that can be used with web-based visualization libraries such as D3, and eventually make it possible to build and coordinate tree visualizations via their topologies.

The second contribution is a conceptual model of *representative suitability*, which is used to assess the capability and suitability of tree visualizations to represent the structures and their attributes in hierarchical data. The model aims to help in predicting how likely tree visualizations are to effectively visually represent specific topologies and the information associated with them.

The third contribution is the design of *Hieros*, a tree visualization library with a modular

software architecture and a hierarchical data pipeline. The architecture supports explicit representation of topologies and interactive operations on them at various stages of the visualization pipeline.

The fourth contribution is a reference implementation of *Hieros*, a tree visualization library implemented in the *Improvise* visualization environment to support construction of topology-augmented tree visualizations.

The fifth contribution is a user evaluation to assess the utility and usability of tree visualizations in explicitly representing topologies.

Overall, this dissertation contributes to the systematic understanding of tree visualizations in terms of topological structures that form the building blocks of hierarchies, the building of a visualization system and data processing pipeline to support topology-centric tree visualizations, and demonstration of their application to a variety of knowledge domains.

Chapter 1

Introduction

1.1 Overview

A hierarchy is a way of organizing entities based on their rank, order, or importance. Hierarchies have many applications in various human-designed systems such as evolutionary trees, file systems, tournament brackets, and organizational charts. The semantics of entities and their relationships in hierarchies differ across applications, depending on what qualifies as an entity and what defines a relationship between entities. For example, in a dendrogram, an entity is a cluster formed by its constituent items, and the relationship indicates which items belong to which cluster. In contrast, in a decision tree, an entity is a decision or choice, and the relationship is the path leading to potential outcomes, which may themselves be further decisions. Tree visualization techniques are flexible in the many ways to visually encode such variations in the data and also offer many ways to interact with hierarchies in applications.

1.2 Motivation

Hierarchical data consists of information about entities *(nodes)* and their relationships *(edges)*. These entities and relationships often combine into *composite topologies* that represent more complex parent-child relationships. For instance, *siblings* are all child nodes of a given parent

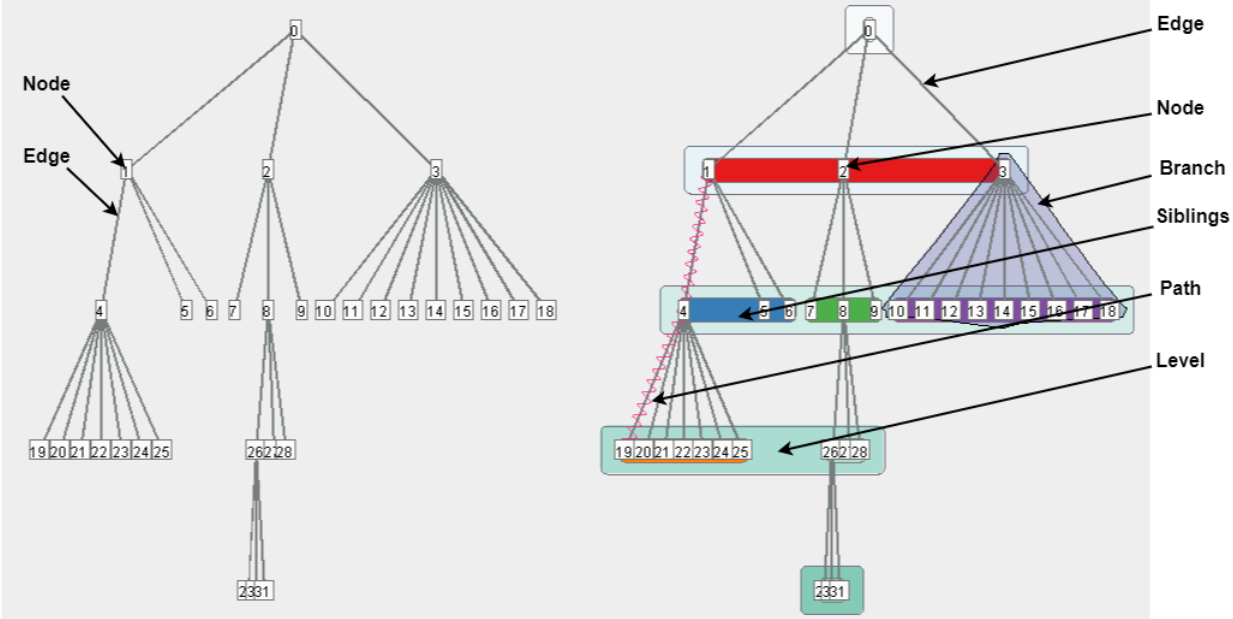


Figure 1.1: Left: Node-link tree visualization explicitly encoding only nodes and edges. Right: Node-link tree visualization explicitly encoding node, edge, level, siblings, path, and branch topologies.

node, a *level* consists of all nodes at a given distance from the root node, a *branch* consists of a given node and all its descendants, and a *path* consists of all the edges between an ancestor node and a descendant node in the tree. More specialized topologies correspond to domain-specific relations. Basic and specialized topologies are typically well-studied in their respective domains, but have received little attention in research on visual representation techniques. Figure 1.1 shows an example of a basic node-link hierarchical visualization that depicts only nodes and edges, as compared to a node-link hierarchical visualization that also explicitly shows level, siblings, branch, and path topologies.

Existing tree visualization designs [113] primarily focus on representing the nodes and edges of hierarchies. In many cases, they also implicitly depict composite structures by highlighting the specific nodes and edges that constitute them. Consequently, tasks involving composite topologies in tree visualizations [92] typically require interpreting information indirectly from their constituent nodes and edges. This presents an opportunity to augment tree visualizations by explicitly representing topologies and supporting new exploratory and

analytical tasks through interaction with these representations. This opportunity raises several challenges:

- Systematically understand the relative strengths and weaknesses of existing tree visualizations in terms of their ability to represent information associated with composite topologies.
- Provide support for representing composite topologies as well as data associated with them.
- Enable users to perform tasks on composite topologies by supporting interactions such as selection, navigation, data editing, and structure manipulation.
- Provide support for coordinating multiple tree visualizations through interaction with the topologies in them.

Existing tree visualization systems and software architectures offer limited support for topologies beyond basic node and edge structures. They generally lack first-class capabilities for representing, querying, and interacting with composite topologies. In visualizations that do have such capabilities, representations and operations are custom designed for specific domains, leaving open opportunities for more systematic understanding and broader generalized application of new representation and interaction techniques.

1.3 Research Path

This dissertation documents research to systematically develop understanding of how to augment tree visualization techniques with explicit representation of commonly occurring composite topologies. The research proceeded in four main stages:

- **Stage One** explored the various commonly occurring topologies in tree visualizations, the kinds of operations one can perform on those topologies, and the different kinds

of information that can be associated with them. We developed a conceptual **model of representational suitability** to assess how well tree visualizations can depict the structure of topologies and information associated with them. Use cases demonstrate how the model helps in selecting tree visualizations more likely to be effective for particular kinds of tasks. (See Chapter 3.)

- **Stage Two** developed a coordination library, **C4D3** [31], to support the development of coordinated multiple view (CMV) visualizations for client-side web applications. The goal was to enable coordination of visualizations more generally on the web, supporting coordination patterns across various types of both hierarchical and non-hierarchical views. The C4D3 library also provides a view abstraction and a coordination mechanism for views built using the popular D3.js JavaScript visualization library [24]. We implemented a gallery of example CMV visualizations to demonstrate the library's utility. (See Chapter 4.)

- **Stage Three** designed and implemented a tree visualization library, **Hieros**, that supports explicit representation of, and operations on, both basic and composite topologies in tree visualizations. We used Hieros to design and implement augmented versions of seven well-known tree visualization techniques that demonstrate a variety of ways to represent and interact with nodes, edges, and composite topologies. (See Chapter 5.)

- **Stage Four** evaluated topology-augmented tree visualizations built using Hieros to assess their usability and utility in explicitly representing topologies and supporting operations on them, offering recommendations for improvement and ideas for future research on tree visualizations. (See Chapter 6.)

1.4 Details of the Research Path

This dissertation systematically studies how to visualize tree data effectively by examining the current and potential role of composite topologies in hierarchical visualizations. It seeks to build understanding of how topologies are currently represented in tree visualizations, the ways in which tree visualizations can visually represent structural and attribute information of topologies explicitly, the interactive operations on tree data that could be performed through those visual representations, how tree visualizations can be coordinated through their topology representations, and develop a visualization system that supports the design and implementation of topology-augmented tree visualizations. Therefore, this research primarily lies at the intersection of visualization theory, design, and system development. We began by grounding our work in visualization theory, followed by the design of visualizations and the development of a visualization system, and concluded with an evaluation of the implemented designs.

The tree visualization design space is huge. Schulz documents over 340 tree visualization techniques [113] to represent hierarchical data. Tree visualization techniques have been organized in many ways [151, 111, 113]. One common way of organization is based on the Gestalt principles of connection, containment, and alignment [111]. Within each Gestalt principle, we can further organize based on the node alignment [113]. We sampled commonly used tree visualization techniques from these two organizations of Gestalt principles and node alignment and study them in terms of providing support for representing topological information. We develop a *model of representational suitability* that helps assess and compare the strengths and weaknesses of the sampled techniques. We examine the base ability and overall suitability of the sampled tree visualizations to effectively represent different kinds of topological structures and their attribute values. The model serves as a guide to designers to choose a suitable visualization for targeted topological tasks.

Complex visualization designs, particularly those with advanced CMV capabilities [140] commonly desired in exploration and analytical applications, are relatively rare in web-based

visualization tasks. Such tools typically connect views directly to implement simple coordination patterns like shared selection. It remains challenging to implement more complex coordination patterns desirable for data exploration in these tools. We re-implemented the Live Properties coordination architecture of Improvise [138], originally implemented in Java, as a JavaScript library to support coordination of visualizations built with D3.js [24]. The resulting C4D3 library provides view abstractions to make D3.js views modular and compatible with the coordination architecture, thereby supporting implementation of complex CMV web-based visualization tools.

Existing techniques for visualizing hierarchical data focus almost entirely on representing and interacting with nodes and edges themselves. Much less attention has been paid to visualizing composite topologies like levels, siblings, paths, and branches. We develop system extensions and view modules in the Improvise visualization environment [138] to support representation and interaction with such composite topologies as first-class objects in tree visualizations. We name the library *Hieros* after the root word for hierarchy. In Hieros, an extended information visualization pipeline [111] maps node and edge data into composite structure information for subsequent visual encoding and layout along with the nodes and edges themselves. We develop support for common operations that occur at various stages of the pipeline to interactively create, select, navigate, and manipulate the composite topologies during data exploration and analysis. Further extension of the pipeline supports coordination of topologies across tree visualizations via the underlying Live Properties [138] architecture in Improvise. We develop seven variations of common tree visualization techniques, augmented with composite topology representations and interaction, as Improvise view modules. We include support for *shared selection*, and *shared navigation* coordination patterns [140] to build coordinated tree visualizations for demonstration and evaluation purposes.

We conduct a user study to assess the usability and utility of the topology-augmented visualizations developed using Hieros. Quantitative measures confirm that participants are able to perform tasks on topologies with reasonable accuracy and speed. Qualitative feed-

back indicates that participants find visualization of the composite topologies themselves to be helpful for understanding those structures and their attribute information, as well as useful for performing tasks on them. They express confidence in interacting with the structures and find the visualizations appealing. We also identify several shortcomings of the topology-augmented visualization designs using Hieros. Participants find it challenging to perform complex tree manipulation tasks, especially involving multiple steps. With so many ways to interact with the tree, participants express that additional visual cues are needed to help understand the interactive state of the visualization between and during task performance. Participants also express difficulty interpreting quantitative information about topologies encoded using color visual channels. This last shortcoming is fundamental to the limited number of visual channels available to encode quantitative information in some tree visualization techniques, as well as the limited effectiveness of color for encoding quantitative information [77]. These results suggest that augmenting visualizations with explicit representations of composite topologies makes interactive exploration and analysis of those topologies in hierarchical data faster and more straightforward overall.

1.5 Thesis Statement and Research Contributions

1.5.1 Thesis Statement

Tree visualization techniques can be augmented to explicitly represent composite topological structures and enable direct interaction with those structures. Explicit representation and interaction with composite topologies visually emphasizes their intrinsic structure and relationships, thereby supporting faster and more effective exploration and analysis of hierarchical data than by visualizing and interacting with those structures solely through their constituent node and edge representations.

1.5.2 Key Contributions

This dissertation describes four primary contributions to the field of information visualization. The first contribution is *C4D3* [31], a JavaScript-based coordination library for views built using D3.js based on the Live Properties architecture in Improvise [138]. Interactive dependencies between views can be specified in a declarative manner, allowing for simple, flexible mixing and matching of different coordinations and creation of coordination patterns.

The second contribution is a conceptual *model of representative suitability*, which is used to assess the capability and suitability of tree visualizations to represent topological structures and their attributes in hierarchical data. The model predicts visualization techniques likely to effectively represent topological structures and the information associated with them.

The third contribution is the design of *Hieros*, a tree visualization architecture to support design and implementation of topology-augmented tree visualizations including their layout algorithms and visual encoding options. Hieros utilizes a hierarchical data pipeline to support interactive operations on topologies along the various stages of an information visualization pipeline [111] specialized for tree visualization. Additionally, Hieros offers a mechanism to coordinate topologies across different tree visualizations.

The fourth contribution is a reference implementation of *Hieros* in the Improvise visualization environment. The implementation includes a representative set of commonly used tree visualization techniques, augmented with composite topology features as Improvise view modules.

The fifth contribution is a user evaluation to assess the utility and usability of representing tree topologies explicitly in hierarchical visualizations.

1.6 Research Questions and Scope

The research presented in this dissertation must answer several questions raised by the thesis.

First, existing web-based visualization libraries currently rely on either low-level coordination abstraction libraries or natural language template-based high-level coordination abstractions for building coordinated multiple views. This either requires the views to be aware of the internal implementation details of other views or restricts the example set of coordination patterns provided through templates. Although simple coordination can be implemented easily in existing libraries using these approaches, it is challenging to implement more complex ways of coordinating views including to support common coordination patterns [140]. How can we design a view-level abstraction for D3.js [24] that offers view modularity and simple, yet flexible composition of coordinated multiple views (CMV) while preserving the expressiveness of individual D3 components?

Second, analysis in different domains may involve topologies in different ways. What kinds of topologies interest users in different domains? What kinds of operations are performed on tree data in different domains? In what ways do users interact with different kinds of topologies, and what kind of data is being modified when performing tree exploration and analysis tasks? In genealogy, for instance, a user might be interested in the number of siblings of a person in a family, or how two distinctly related people are connected. These queries involve operations on siblings and path topologies, respectively.

Third, there are many variations of tree visualization techniques for representing hierarchical data [113]. Broadly, they can be organized based on how parent-child relationships are depicted in terms of the the Gestalt principles of connection, containment, and alignment [111, 151]. The choice of visual encoding of nodes and edges affects how users perceive them differently in different techniques; some choices of visual encoding are more favorable for certain kinds of data [77]. What are the strengths and weaknesses of tree visualization techniques in visually encoding the structural and attribute information of composite topologies?

Fourth, current tree visualization systems use a tree visualization pipeline [111] that exposes only nodes and edges as first-class objects. Extending the existing pipeline could offer

support for explicit composite topology visual representation and interactive operations on those topologies along the pipeline in a first-class manner. What levels of visual representations and interactive operations need to be supported? What are the necessary components of such a visualization pipeline and how can they be integrated into a software architecture for efficient and effective design and use?

Fifth, studying the designs of existing tree visualization techniques would help us understand the possibilities for augmenting them with explicit visual representations and interactions on composite topologies. What are the different ways that topologies can be explicitly represented in tree visualizations? What kinds of interactions can be performed on those representations to support operations on their topological structures and attributes? How can those representations also depict interactive state during and between topology-related tasks?

Sixth, what are the benefits and drawbacks to designers of having an architecture and implemented library for creating tree visualizations? How well can such an architecture support design and use of operations on tree data? What are the practical impacts of treating topologies as objects of primary importance in the data analysis process, particularly as targets of visual representation and interaction? How are different tree visualization techniques amenable to topological augmentation, or not?

Seventh, we would like to understand the utility and usability of topology-augmented tree visualizations built using the Hieros library. How efficiently and accurately can users perform operations on topologies in those visualizations? What challenges do they face? How might Hieros be improved to overcome those challenges?

1.7 Organization of the Dissertation

The remaining chapters in the dissertation are organized as follows.

Chapter 2 provides an overview of prior work in coordinated multiple views, representation

of topologies in tree visualizations, tasks performed on tree topologies, and the structural and attribute information associated with topologies. It discusses how existing tree visualization techniques represent topologies and support different kinds of interactive tasks on them.

Chapter 3 presents a model of representative suitability for assessing the capability of a given technique to represent topological structure, its characteristics, and associated attribute information. It describes a general method for applying the model to technique-topology pairings. The method is used to populate a representative sample space of well-known tree visualization types and both common and specialized topologies. The results identify important patterns across techniques and structures, validating past design choices and suggesting future opportunities. These findings are then used to assess the criteria and overall method for determining suitability. Finally, the chapter offers strategies to select tree visualizations that are well-suited to different mixes of hierarchical topologies and tasks.

Chapter 4 presents a coordination library based on the Live Properties architecture for creating coordinated multiple views (CMV) [138]. It describes the architecture of view modules in C4D3, provides guidance on how to create view abstractions to make the views modular and work with the C4D3 coordination layer, presents example applications, and assesses C4D3's applicability to web-based visualization development.

Chapter 5 describes the software architecture of Hieros, its components, and their integration in an implemented library for building topology-augmented tree visualizations. It details extension of the standard information visualization pipeline of Hieros to support visual representation and interactive operations on composite topologies, a variety of visual representations and interactive operations for those different topologies, and key topology coordination patterns supported by Hieros. It presents design variants of seven well-known tree visualization techniques augmented with explicit composite topology

representations and interactions. Finally, it describes the design considerations involved in the development of Hieros and makes an overall assessment of its design capabilities and limitations.

Chapter 6 presents a user study evaluating the usability and utility of topology-augmented tree visualizations implemented using Hieros, discusses the operational limitations of Hieros designs and summarizes directions for improvement and ongoing development.

Chapter 7 concludes with a discussion of the benefits and limitations of the topology-augmented approach in terms of Hieros, its data pipeline, and the example augmented tree visualization designs; an outline of possible extensions and future work; and a summary of the contributions of the dissertation.

Chapter 2

Background and Related Work

This chapter reviews prior work that lays the foundation for this thesis by focusing on tree visualization centered around topologies. It examines how existing tree visualizations integrate information about composite topologies, especially in their visual representation and corresponding support for interactive operations on topologies. It then outlines how existing visualization pipelines for tree visualizations are designed to work with topologies for their visual representation and to support more complex data transformation and querying operations on those topologies. This chapter reviews literature on the visual representation of tree topologies, tasks involving topological data, the data transformation and query support provided by tree visualization systems, and support for coordination of topologies in tree visualizations. It also reviews prior work on coordination architectures and visualization systems that support the development of coordinated multiple views more generally.

2.1 Tree Visualizations

The visualization community continues to show a substantial interest in developing techniques for visualizing hierarchical data [113]. Historically, people have used figurative representations of trees [72] to depict hierarchical data through organic traits of a biological tree such as trunks, branches, shoots, fruits, and leaves. Also referred to as symbolic rep-

resentations, they continue to find practical use, such as for the depiction of phylogenetic trees [102]. Figurative representations continue to inspire research and development of hierarchical visualization techniques with diverse layouts and styling. These techniques utilize visual channels like area, shape, and length to show information more abstractly than in historical forms, allowing them to generalize well across domains and applications.

Many taxonomies of tree representations have been proposed. Schulz [113] categorizes tree visualization techniques as *implicit hierarchies* that use non-drawn metaphors to represent parent-child relationships, *explicit hierarchies* that draw metaphors such as edges to represent parent-child relationships, and *hybrid* representations that do both. Schulz, et al. [115] further classify the implicit, explicit, and hybrid representations as *root-centric*, *parent-centric*, and *hybrid* based on whether layout operations are determined with respect to the tree's root node and/or with respect to each node's parent. Techniques are further categorized on their dimensionality (*2D, 3D, hybrid*) and how nodes are aligned with the root node (*horizontal, vertical, radial, free*). Scheibel, et al. [111] categorize treemap visualization techniques as *space-filling treemap*, *containment treemap*, *implicit edge representation*, and *mapped tree* based on the is-a relationship between nodes. Other categorizations [152, 92] are based on Gestalt principles or structural characteristics such as *connection, containment, alignment, adjacency, indented outline, symbolic,* and *hybrid*. Fung, et al. [49] uses matrix based representation of trees. Some taxonomies sub-categorize techniques in terms of levels and enclosure [135] or forms of implicit connectivity [114]. In Chapter 3, we use a similar categorization to organize common tree visualization techniques for modeling their representational suitability. Later on, this organization helps in sampling the tree visualization space and studying it in terms of how topology information can be represented in the sampled variety of tree visualization techniques. In Chapter 5, we use the same categorization to develop variations of the common tree visualization techniques to explicitly represent topologies.

2.2 Representation of Topologies in Tree Visualizations

Hierarchical visualizations depict topologies either explicitly or implicitly. For instance, linear and radial variants of node-link views [22, 116] represent nodes and edges explicitly; squarified treemaps [117], sunburst charts [122], and icicle plots [147], and circular treemaps [145] represent edges implicitly; and nodes in dendrograms [153] and leaf nodes in rings [128] are represented implicitly. The implicit representations of topologies rely on the Gestalt principles of connection, containment, alignment, etc. in most cases and the particularities of layout in others. Some visualization techniques that represent tree structures go beyond traditional node-edge depictions to visually represent some composite structure by virtue of their layout. Nested pietrees [115] and circular treemaps [145] use buffer space to represent siblings. Other's rely on using an available visual channel of the basic topologies to surface a composite topology, such as by highlighting the nodes of a given level with the same color [66]. Still others utilize coordinated views to show topologies like paths [10].

Some approaches use augmented node-link diagrams to emphasize topological attribute information, as seen in Hierarchical Bundling [61] and TreeVersity [55], which visually encode attributes not only in nodes and along edges, but also within branches to reflect multifaceted hierarchical relationships. Containment-based visualizations, such as rectangular treemaps [118], and their improved variants, including squarified treemaps [27] and Voronoi treemaps [14], represent sets of siblings as regions enclosing constituent items. Alignment-based methods, notably icicle plots and sunburst charts [10], further delineate hierarchical levels and sibling groupings through linear or radial arrangements, while polar treemaps [64] merge treemap principles with polar coordinates to achieve radial space-filling layouts. Adjacency matrix representations, such as matrix-based tree visualizations [51] and hybrid systems like NodeTrix [59], offer alternative views that can be particularly advantageous for large hierarchies by highlighting structural connections in a compact form.

More recent research has integrated focus+context and distortion techniques, exemplified by adaptations of Table Lens [98] and Local Edge Lenses [129], to support dynamic explo-

ration and drill-down of specific tree topologies while preserving overall context. TreeJuxtaposer [81] demonstrates that implicitly visually encoding branches and paths by highlighting their constituent nodes and edges facilitates comparative analyses and detailed exploration of complex hierarchies. Paisley, et al. [91] uses implicit representation of paths for topic modeling. Some domain-specific tree visualization techniques [94] draw a horizontal plane in 3D to represent levels. Bubble Sets [39] use isocontours to highlight sub-trees in a parse tree. Balzer and Deussen [13] use semi-spheres to represent siblings in 3D representation of trees. These various explicit representations are applied in an an ad-hoc or in domain-specific manner, leaving open the opportunity to apply and study explicit representation of composite topologies in tree visualization techniques more generally.

The layout algorithms used to generate the visual representations of tree visualizations often take into consideration particular aspects of tree structure. Reingold and Tilford [99] describe an algorithm for tidier drawing of trees that are aesthetically pleasing and use minimal space. Misue [79] describe a tree layout algorithm that adopts to different aspect ratios of display. He and Zhu [57] describe a level-based tree algorithm that preserves the order of leaf nodes. The algorithms focus mainly on maintaining the aesthetics of the tree and making better utilization of available space. Other algorithms for tree layout try to preserve the characteristics of composite topologies; for instance, Bederson, et al. [19] describe ordered and quantum treemaps that preserve the order of siblings in a space-filling layout. Existing tree visualization techniques generally put emphasis on efficient space utilization or highlighting particular aspects of composite topologies. In Chapter 5, we modified the layout algorithms of common tree visualization techniques to incorporate explicit visual representation of composite topologies.

2.3 Tasks on Topologies in Tree Visualization

Tree visualizations aim to facilitate exploration and analysis tasks on hierarchical data driven by user needs. Brehmer and Munzner's multi-level typology [26] characterizes visualization use by differentiating the *why* and *how* of tasks and providing abstractions for both. Pandey, et al. [92] extended the *why* aspect specifically for tree visualizations, categorizing tasks by abstraction level: *analyze* (high-level, consuming topological information), *search* (mid-level, finding specific topologies), and *query* (low-level, identifying, comparing, or summarizing topologies) to abstract visualization tasks and add specificity when characterizing target topologies on which the visualization tasks occur. The *how* aspect of Brehmer and Munzner's typology [26], which encompasses data manipulation, encoding, and data generation in visualizations, is generally applicable to tree visualizations. Together, these typologies [26, 92] provide a framework for organizing tasks on tree topologies and developing visual representations and interactions for those tasks. Munzner, et al. [81] introduced TreeJuxtaposer, which enables encoding, navigating, selecting, and arranging operations on topologies, especially branches, to support *mid-level* tasks of search and *low-level* tasks of comparison. Graham and Kennedy [52] use a directed acyclic graph visualization to support *low-level* comparison tasks on multiple trees. It allows for comparison of implicit representations of composite topologies such as paths, and siblings. TreVis [4] uses a context tree visualization to support comparison of trees based on the attributes of nodes. It uses distance-based metrics to compare the trees and the visual encoding of nodes to represent similarities and differences. Radial Clustergrams [5] support tasks involving browsing a tree and selecting subsets of nodes using selection and navigation interactions. It also uses a *fisheye distortion technique* [107] to browse parts of the tree without losing global context. InfoSky Visual Explorer [12] supports level-based exploration of document collections. It supports operations on attributes of a document, such as size or age, and maps them to the visual encoding of the nodes, such as color and luminance. These various visualization systems use visual encoding of nodes and edges to implicitly represent composite topologies,

with topology interactions performed on the explicit representation of nodes. Explicitly representing composite topologies could provide ways to compare topologies directly; enable direct select, filter, and navigation interactions on composite topologies; and thus support analyze, search, and query tasks directly on tree topologies. This directness could increase usability by making structural and attribute analysis tasks more efficient and learnable.

Visualization tools like SpaceTree [96] combine data pipelines with interactive querying, allowing users to modify visual representations through filtering, selection, and drill-down based on node attributes and hierarchical levels. Tools such as Treemap 4.0 [37] and SequoiaView [67] provide different filtering methods. Treemap 4.0 primarily employs attribute filters (on e.g., size, age) to visually gray out nodes, alongside structural filters controlled by a dynamic query interface to filter out particular composite tree structures. SequoiaView offers ways to filter files based on attributes such as name, size as well as creation, modification or last access date or any combination of these in addition to sorting and highlighting of files. These developments underscore the significant potential of tree visualization systems that utilize a visualization pipeline to explicitly represent topologies and support a wide array of topology tasks [92] beyond simple node-edge interactions. In Chapter 5, we describe the implementation of a visualization pipeline to support operations on trees through interactions to create, filter, select, navigate, and generally manipulate composite topologies as well as nodes and edges.

2.4 Visualization Pipeline for Topology Support

Generation of a tree visualization typically happens in three phases along a visualization pipeline [111]: *pre-processing* that is responsible for delivering suitable data describing the tree's structure and attributes, *visual mapping* for converting the data into visual structures and details, and *rendering* that converts the visual structures into displayed graphics. Existing tree visualization systems mainly focus on supporting node and edge structures in

these stages. Different systems implement data transformations on these basic structures in different ways. GoTree [71] is a visualization system that describes a computational pipeline for visualization of node and edge data. User interactions create operations that direct the generation of views by modifying the three stages of the pipeline to determine layout specification, node and edge visual encoding specification, and coordinate system specification. Schulz, et al. [115] explore the layout generation process in the visual mapping phase and propose a generative layout approach that follows a six-step layout process to generate rooted tree drawings.

The Data State Reference Model [36] describes a taxonomy for classifying techniques that apply data operators at various stages of the visualization pipeline, and identifies various visualization techniques used by tree visualizations [101, 69, 35, 16] involving different stages of the pipeline. Chapter 5 describes how Hieros extends the information visualization pipeline to map node and edge data into composite structure information for subsequent visual encoding and layout with the nodes and edges themselves. Hieros supports operations at various stages of the pipeline to interactively create, select, navigate, and manipulate these structures during data exploration and analysis.

2.5 Coordination in Tree Visualizations

Coordinated multiple views are often used for exploring and analyzing different facets of the same data or for comparing distinct datasets using multiple visual representations [140]. Coordinated multiple views can be employed for the exploration of hierarchical data in a variety of ways. Common techniques include edge drawing [126], which connects nodes between different trees with edges; coloring [81, 31], in which color highlights identical items across trees; and animation [144], which tracks changes in a tree over time. Another strategy involves comparing topologies across representations by coordinating constituent nodes through selection [31] or by coordinating the constituent nodes of composite topologies, such

as branches [114]. Tree-Panels [127] utilizes a combination of tree representations to display complementary information about the nodes and edges within a hierarchy, although it does not otherwise coordinate them. In Chapter 5, we describe extension of the tree visualization pipeline to enable the coordination of explicit representations of composite topologies across tree visualizations, specifically using *selection* and *navigation* coordination patterns [140] to create coordinated multiple views.

2.6 Coordination in Visualizations

Interactive systems, as surveyed by Grammel, et al. [54] and Mei, et al. [78], have emerged to assist in constructing CMV's without requiring programming expertise. For example, Snap [90] enables users to coordinate views by joining data tables based on relational data models and coupling primary- and foreign-key actions. Despite their expressiveness, these systems have a steep learning curve and require verbose operations, making the creation of coordinations tedious for users. Other systems like Tableau (formerly Polaris) [123], simplify coordination by allowing users to create common coordinations based on predefined templates. MyBrush [68] and Lyra2 [154] support customization and rapid creation of interactions in multiple views. Dataflow systems such as VisFlow [150], provide dataflow operators that can establish coordination along the dataflow, in which interactions in upstream views can change the visual representation of downstream views. The construction can be further simplified with the assistance of natural language interfaces like FlowSense [149], and Gosling [76]. Despite their capability to reduce technical burdens, these methods suffer from limited expressiveness; i.e., they only supporting limited types of coordination and a limited range and variety of coordination patterns.

On the other hand, programming-based tools provide users with the capability to author custom coordination constructions. Visualization programming toolkits and libraries, such as ProtoVis [23] and D3 [24], offer high expressiveness but demand considerable pro-

gramming expertise. To mitigate this, declarative grammars grounded in The Grammar of Graphics [146] have been introduced, enabling the rapid generation of visualizations through concise JSON syntax. However, these grammars often lack specific coordination support, requiring users to switch context between individual visualization elements and coordination specifications, hindering coordination construction. Improvise [138], addresses this need by employing a direct shared-object coordination mechanism (Live Properties) along with an indirect coordination mechanism (Coordinated Queries) with CMV specified through a visual abstraction language.

Coordination capabilities are directly integrated into the majority of desktop and web-based visualization systems. More recently, Nebula [34] provides an abstraction layer for constructing coordinated visualizations using macros to abstract coordination specifications. It offers efficient CMV design through textual specification of supported coordination patterns. Use-Coordination [65] is a reactive library for coordinating views, built using the *ReactJS* framework. Use-Coordination and Nebula offer low-level coordination abstraction and natural language template-based coordination patterns respectively, limiting flexibility to specify coordination patterns and thus making it hard to compose complex CMV's. Chapter 4 describes C4D3 [31], a shared-object coordination library based on the Live Properties architecture [138] of Improvise visualization system. C4D3 provides a mid-level abstraction to implement complex coordination patterns [140] flexibly and easily in web-based visualizations.

2.7 Summary and Gaps in the Literature

The survey above hints at the substantial utility of hierarchical visualizations that can represent information about topologies explicitly, beyond the usual nodes and edges. Existing visualizations do effectively represent the basic hierarchical elements of nodes and edges. They also heavily rely on these elements to represent composite topologies such as levels,

paths, and siblings implicitly. Due to the limited visual encoding channels available for representing nodes and edges, re-purposing these channels for composite topologies presents challenges. These challenges may include losing the ability to represent information about nodes and edges simultaneously, or may create ambiguity, as a given node or edge can be part of multiple composite topologies at the same time, making it unclear which topology is being depicted by the current visual encoding of the node or edge. While hybrid and multiple view approaches can partially address these issues, they typically rely heavily on implicit representations (often domain-specific) rather than offering generalized, first-class support for diverse topological structures. Moreover, user interactions in tree visualizations are generally limited to simpler operations such as panning, zooming, and basic filtering. More advanced interactions that leverage detailed topological structures, such as to highlight siblings across multiple levels, remain ad-hoc and are rarely generalized. These shortcomings highlight the need for a more systematic approach to representing and interacting with composite topologies that benefit from explicitly representing them within tree visualizations. In addition to informing the design of new and augmented tree visualization techniques, the development of software architectures to treat these topologies as first-class visualization objects would enable operations on trees involving complex querying of data at different stages of the visualization pipeline including through coordinated interaction.

Chapter 3

Representational Suitability Model for Tree Visualizations

3.1 Overview

Hierarchical information is of interest in many domains. Researchers and practitioners have developed a variety of visualization techniques to represent and interact with hierarchical data. While many of these techniques have been studied in terms of their effectiveness for performing tasks on such data, the results have been generally inconsistent and difficult to generalize. We present a general model of the suitability of tree visualizations to visually represent both simple and more complex topological structures in hierarchical data and the data attributes associated with them. Our primary goals are to inform researchers about which visualization techniques are well-suited (or not) to represent different aspects of hierarchical information and guide designers in the selection of suitable techniques to support particular domain analysis tasks. The model applies a combination of five heuristic criteria—encoding capability, channel accuracy, layout capability, side effects, and scalability—to assess the suitability of a given technique to represent topological forms, characteristics, and associated attributes. We describe a general method to apply the model to technique-

topology pairings, systematically apply the method to populate a large representative sample space for well-known tree visualization types and common and more specialized topologies, identify important patterns across techniques and structures to validate past design choices and suggest future opportunities, use the results to assess the criteria and overall method for determining suitability, and offer strategies to select tree visualizations that are likely to be well-suited to different mixes of hierarchical topologies and tasks on them.

3.2 Introduction

Consider a user who wants to use a tree visualization to represent the hierarchical organization of a university. The hierarchical data consists of information such as the different colleges in the university and the multiple departments they oversee, the size of various departments, the levels of supervision for various employees, the funds allocated to various departments, etc. There are many varieties of tree visualizations from which the user can choose. Each of these visualizations varies in its design considerations and has its own strengths and weaknesses for representing different aspects of hierarchical information. For instance, squarified treemaps are well-suited for comparing properties of leaf nodes, whereas node-link treemaps are well-suited for understanding the connectivity of all nodes in a tree. Given a variety of tree visualizations and analysis goals, how might a university employee go about making a choice of visualization to represent their organizational data?

Many user studies [121, 15, 67, 136, 74] have been conducted to find and compare the strengths and weaknesses of particular techniques. Many of these user studies rely on providing a curated list of tasks to be performed by participants and make conclusions based on the analysis of measures such as the speed and accuracy with which the participants performed these tasks. Fiedler, et al. [45] survey 69 such studies and argue that these studies, while useful, failed to produce knowledge that can be reliably derived and generalized. A user intending to select a tree representation based on these user studies must synthesize

localized insights from small sets of compared tree representations and the often simple range of visual interpretation tasks considered. More comprehensive coverage of the design space of visual representations and the tasks a user can perform in them is needed.

Visualization design builds on both theoretical foundations and long-standing practical knowledge of how visual perceptual channels can effectively represent data *attributes* in visualizations [20, 38, 77]. However, there is much less knowledge about how those channels are (or are not) effective for representing aspects of data *structure*, particularly for non-tabular data structures, including hierarchical data. Hierarchical data varies across domains in terms of both structural relationships and data attribute characteristics. Effective perception of this data is highly influenced by the choice of tree visualization technique as well as the variation in visual encoding mappings employed within the technique to represent structures and attributes.

The primary contribution of the research described in this chapter is to provide a model of *representational suitability* that helps assess and compare the strengths and weaknesses of commonly used tree representation techniques. We do this by organizing the tree visualization design space into categories of techniques based on their general characteristics of connection, containment, and alignment, much like in related applications and surveys [152, 114]. We then examine their base *ability* and overall *suitability* to effectively represent different kinds of topological structures and their attribute values.

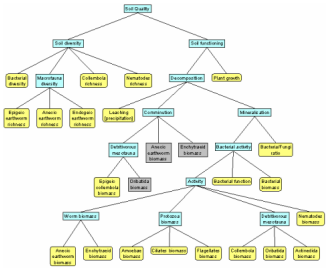
The model is centered around the topological structures of trees likely to be essential for analysis. Users from different domains can abstract their domain-specific tasks in terms of the topological elements of interest, helping the model to generalize across domains. The model also identifies examples of common types of data that these topological structures represent and organizes them into structural attributes and data attributes under each topological structure. This helps visualization designers assess the suitability of candidate tree visualization techniques in ways that consider structures and attributes both individually and in combination. The goal is to facilitate a more effective and efficient selection of vi-

sualization designs for both general and domain-specific applications to hierarchical data analysis.

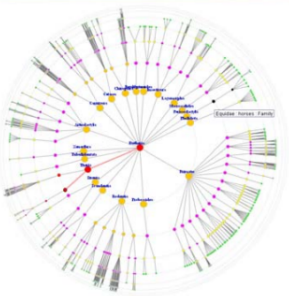
We start with a background on hierarchy visualizations, focusing on the evolution of their utilization of visual channels to support data exploration and analysis tasks, and argue for the need for a more systematic understanding of tree representations in terms of their ability to visually encode data. We then discuss the need to center topology in considerations of the tree data to better understand the kinds of tasks domain users want to perform on trees. We introduce the aspects of the tree representation model in terms of its general structure including an organization of tree representations, an organization of common topological elements in tree data, common types of structural and data attributes, and the kinds of information in need of visual representation. We then discuss heuristics to determine the effectiveness of tree representations. We present a method by which visualization designers can utilize our model to assess the visual representation capabilities of particular tree representations and describe how we applied the method ourselves. We summarize our results in terms of observations of patterns for a sample set of common visualization techniques (Figure 3.1) and topologies. We then analyze the practical application of the method and the model itself in terms of its strengths, limitations, overall applicability, and future research directions.

3.3 Related Work

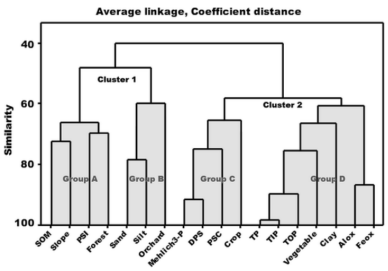
Hierarchical data consists of information about the entities in a hierarchy, the characteristics associated with them, and the relationships between them. The parent-child nature of hierarchical relationships gives rise to a variety of *topological structures* ("topologies") defined by subsets of entities and parent-child relationships. Beyond the elemental topologies of *nodes* (for each entity) and *edges* (for each relationship), commonly encountered topologies include *levels*, *paths*, *sub-trees*, and *siblings* [44, 92]. Other topologies, including more complex and specialized ones, are important in particular domains, such as *cousins* in genealogy.



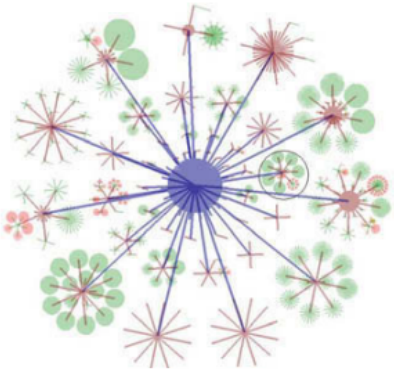
A: top-down node-link [22]



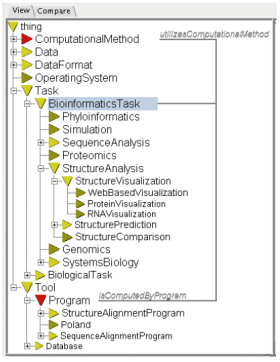
B: radial tree [116]



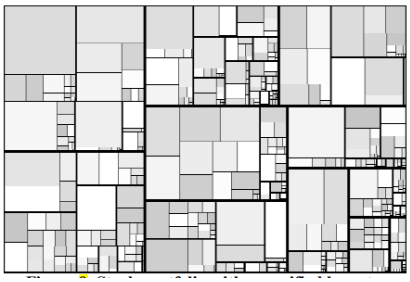
C: cluster dendrogram [153]



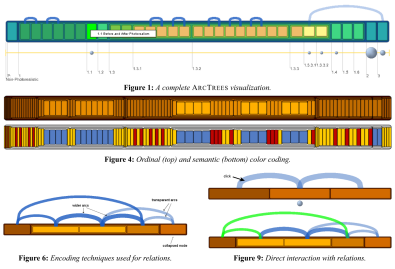
D: rings [128]



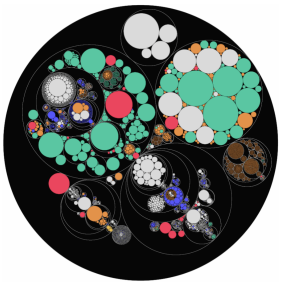
E: indented outline [62]



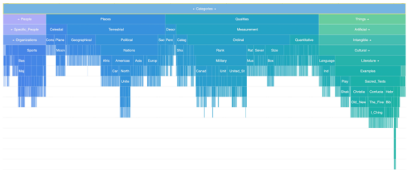
F: squarified treemap [117]



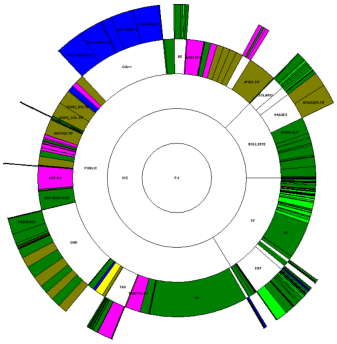
G: ArcTree [84]



H: circular treemap [145]



I: icicle plot [147]



J: sunburst chart [120]

Figure 3.1: Examples of the ten tree visualization techniques sampled in the representational suitability model.

These structures are typically well-studied in their respective domains, but overlooked in visualization taxonomies. Evaluations of tree visualization techniques typically focus on only a few of the simpler topologies as task targets. Our model centers on topology as the focal point in determining representational suitability. It includes basic, common, and a few unusual topologies to provide enough coverage of the design space to be useful for general visualization design.

Earlier studies of effectiveness of tree visualizations, especially those involving the comparison of tree visualization techniques [121, 15], provide a sense of the suitability of tree representations for the tasks considered. However, these studies rarely consider more than a few specific tree representations and provide limited understanding of why particular visual representations are suitable for a task, either individually or collectively. Moreover, coverage typically includes only a small sample of desirable (let alone sensible) tasks that could be performed on trees. It is difficult to sample the space of tasks performed on trees, and to generalize the results, without more comprehensive coverage. Consequently, a systematic understanding of tasks performed on tree representations and a detailed analysis of strengths and weaknesses of those representations is needed to understand and select suitable trees for the tasks and data one wants to support in visualization designs.

In the visualization research community, the evolving understanding (mostly ad hoc) of the relative strengths of various tree representations, for instance the ability of connection-based node-link techniques to represent structures effectively, or the ability of containment-based representations such as treemaps to use space efficiently, motivated the development of *hybrid tree* visualizations [152, 56] and *multiple tree* visualizations [127, 53, 50]. Multiple tree visualizations split data across multiple different views based on their ability to show particular characteristics. Hybrid tree visualizations integrate multiple representation techniques to better represent particular structural relationships. Some hybrid techniques transition parts of the tree from one representation to another [152]. Despite the availability of many ways to visually represent trees, it remains challenging to find a visualization to

suit situation-specific analysis needs in an informed manner. We argue that this is due not to a lack of techniques, but rather to a lack of systematic understanding of representational capabilities. In developing the model, we follow a bottom-up *information-centric* approach to analyze how well visualized tree topologies can represent characteristics of information, in contrast with top-down *task-centric* approaches that analyze performance of tasks on those topologies.

Fiedler, et al. make a similar argument in their survey of user studies [45]. The results of those studies offer observations on which visual representations work best for particular tasks in specific domains, but provide neither a theory-backed explanation of how and why those representations are suitable, nor insights on their generalizability to other tasks. The results of evaluation methods [30] can be difficult to extrapolate beyond the particular techniques considered. For instance, many studies [121, 67, 74] observe that treemaps (Figure 3.1F) are well-suited to show node attributes. Müller, et al. later argue from Gestalt principles that node-link and treemap representations outperform icicle plots for the perception of quantitative node attributes in simple settings, and the reverse in complex settings [83]. More such work is needed to ground evaluation studies in theory, such as to account for attribute type differences like categorical versus ordinal or discrete versus continuous.

Heuristic based evaluations [46, 85, 87] has often been used as a method for assessing visualizations, offering an alternative to longitudinal evaluation, which is logistically challenging, very time consuming, and difficult to implement. While its origins lie in Human-Computer Interaction, notably with frameworks like Nielsen's usability heuristics [86], the information visualization community has increasingly recognized the necessity of specialized heuristics that address the unique challenges of visual data representation and the amplification of cognition [156, 29]. Generic usability heuristics, though relevant, often fall short in identifying problems specific to how visual encodings and layouts impact a user's ability to gain insight and understanding [156]. This recognition has spurred the development of numerous visualization-specific heuristic sets [40, 47, 41, 112, 125]. These frequently draw from estab-

lished theories in perception, cognitive psychology, and graphic design, as reflected in the influential work of Bertin, Tufte, and Ware [20, 131, 137]. Such specialized heuristics aim to systematically evaluate aspects like visual clarity, the effectiveness of data mapping to visual channels, and the avoidance of misleading visual patterns, thereby providing a discount evaluation method [156] that can be applied early in the design process to identify potential issues before they become deeply embedded.

We populate a model of representation capabilities grounded in theories of visual encoding [77], visual grouping of objects in space [93, 143], and general applications of Gestalt principles in visualization design [155, 80, 137]. Despite its incomplete grounding in empirical evidence, this theory-driven approach offers coverage of existing techniques, ready expansion to new techniques, and predictions from observed patterns to motivate further empirical studies.

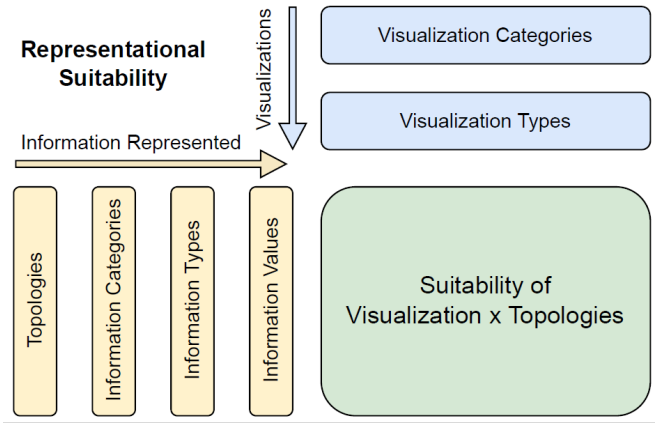


Figure 3.2: The organization of the representational suitability model.

3.4 A Model of Representational Suitability

In this section, we present a model of the suitability of tree visualization techniques to represent topologies in hierarchical data and the information associated with them. The main premise of the model is that visual interactions between a layout of topological elements and the visual channels used to encode information about them constrain the effectiveness of

the overall visual representation for conveying topological structure and details. The model is designed to be progressively and cumulatively applied to new techniques and topologies. We apply the model to analyze the suitability of well-known types of tree visualization techniques for visually representing both commonplace and specialized kinds of topological information.

The structure of the model is shown in Figure 3.2. The first dimension of the model (blue horizontal column headers) organizes tree *visualization types* into perceptual categories. The second dimension (yellow vertical row leaders) organizes the topological *information represented* into a four-level classification scheme based on the structure and attributes of topologies in hierarchical data. The cross-product of the dimensions (green matrix) characterizes the suitability of each visualization type to represent each kind of topological information.

3.4.1 Visualization Types

The model is not inherently limited to particular tree visualization types or how they are categorized. For the purposes of presenting and assessing the model, we focus on ten visualization types (see Figure 3.1) that are representative of the variety of commonly encountered techniques and how they visually represent parent-child relationships. The visualization types are categorized by the Gestalt principles of **connection**, **containment**, and **alignment**. These categories correspond to the three most common ways to encode a parent-child relationship:(1) as an explicit edge between the parent node and the child node, (2) with the child node enclosed in the parent node, and (3) through alignment of the child node with the parent node.

The **connection** category consists of visualization types (Figure 3.1A-E) that represent each parent-child relationship as an explicit edge connecting the child node to its parent node, including axial and radial node-link diagrams. Wang, et al. [135] use a similar category to organize such tree visualization types. We lump several other techniques that are sometimes categorized differently but that utilize explicit edges—dendrograms (Figure 3.1C), rings (Fig-

ure 3.1D), and indented outline (Figure 3.1E)—into the connection category. Dendrograms and rings represent nodes implicitly as a point where cluster edges meet, except for leaf nodes which show additional detail. Indented outlines encode relationships between children and their parents using a mix of connection, containment, and alignment; because connection offers more ways than containment or alignment to represent topological information, we chose to put indented outlines in the connection category.

The **containment** category consists of visualization types that represent parent-child relationships by enclosing child nodes inside parent nodes. Wang, et al. [135] use an *enclosure* category for tree visualizations that enclose children in a parent with 100% utilization of the parent area, such as squarified treemaps (Figure 3.1F) and ArcTrees (Figure 3.1G). (Note that the connecting lines drawn in ArcTrees do not denote parent-child relationships.) Scheibel, et al. [111] organize a taxonomy of tree visualizations based on the characteristics of the spatialization process and how parent-child relationships are encoded, including a category to distinguish space-filling from other containment techniques. We include circular treemaps (Figure 3.1H) as an example of a non-space filling containment technique.

The **alignment** category consists of views that represent parent-child relationships as implicit edges by aligning child nodes around parent nodes either axially or radially. Schulz, et al. [114] discuss this category of visualizations as *implicit representation by alignment*. Icicle plots (Figure 3.1I) and sunbursts (Figure 3.1J) lay out nodes using radial and vertical alignment, respectively. An important characteristic of the *alignment* category is explicit use of rectangular strips or ring segments to represent nodes. These regions offer many ways to encode topological information including embedding of details within strip/segment areas.

The visualization types included in each category are selected to capture variety within each category as well as overall. The types vary in their overall *axis-parallel* versus *radial* character of layout [113] and in how explicitly they visually represent nodes and edges. All categories (not just **alignment**) include at least one visualization type in which node alignment influences interpretation of topological information.

We do not currently include *hybrid* visualization types like those in organizations that aim to cover the entire hierarchical visualization design space [135, 92]. Hybrid techniques often have idiosyncratic designs that integrate primarily connection and containment approaches to represent topological structure and attributes in complementary ways. While hybrid techniques could serve as additional examples of the approaches *individually*, assessing suitability of approaches *in combination* calls for an additional level of modeling (much as models of interaction inside views differ from models of coordination between them). We also do not include visualization types considered to be *general* [135], including *symbolic diagrams* [92] like phylogenetic trees [102], or specialized containment techniques like Voronoi treemaps [14] and map-like visualizations [21] that would add little to explication of the model here. Other categorization schemes could also be applied to the tree visualization space [113] to study how different organizations reveal patterns of representational suitability under the model.

3.4.2 Information Represented

Hierarchical data is rich in topological structure and attribute information. The diversity of tasks that one might want to perform to explore and analyze such information visually is huge. This makes it challenging to choose which information and tasks to include when evaluating tree visualizations. Plaisant [97] argues that data and task selection is an ad hoc process in part because there are so many worthwhile combinations to focus upon. Our model recognizes the need for a systematic way to organize and sample topologies and techniques for empirical study, especially to assess the capability of different techniques to effectively represent particular structures and attributes visually.

The model breaks down the *information represented* in hierarchical visualizations into four levels, as shown in Figure 3.2. The first level distinguishes the various *topologies* one can encounter working with hierarchical data. The second level subdivides each topology into *information categories* that capture the different kinds of structure and attribute information that might be associated with it. The third level further subdivides each information

category into *information types* of those structures and attributes. Finally, the fourth level of *information values* describes specific data values (of the information types) that one would like to visually represent effectively in a tree visualization.

Topologies

We define a *topology* as a structure within a hierarchy consisting of subsets of entities and parent-child relationships that have a meaningful relationship among them.

Elemental Topologies are the basic topological components of trees. A *node* represents an entity, such as a person in a family or a file in a file system. An *edge* represents a parent-child relationship, such as between a mother and son or between a directory and a file in it. Nodes and edges are the blocks used to build more complex topologies.

Composite Topologies are formed by grouping nodes and edges in structurally meaningful ways, either directly (as nodes and edges) or indirectly (as constituents of other composite topologies). Several composite topologies are all but universal in hierarchical data analysis. A *tree* is the entire hierarchical structure as a whole, consisting of all nodes and edges. A *path* is a series of nodes connecting an ancestor to a descendant with the edges connecting them. A *level* is a set of nodes that are equally distanced from the root along their respective paths. A *branch* (also known as a *subtree*) consists of an ancestor node and all of its descendant nodes and edges. Other composite topologies are encountered across domains but are not commonly considered in visualization techniques. For example, *siblings* are all child nodes of a common parent. *Bi-paths* capture the relationship between two descendant nodes through a common ancestor node. There are also topologies that are more frequently encountered in particular domains. For example, *cousins* are the non-sibling nodes of a common grandparent, as in genealogy. Similarly, *leaves* is the set of all nodes without children, such as the non-directory files in a file system. Such topologies are covered by the model, and we include two as examples in our sample set (see Figure 3.3, leftmost column). The set can be extended readily to add general-purpose and domain-specific topologies as desired.

Information Categories

The information that describes a hierarchy can generally be classified as *topological* or *attribute* information according to the tasks performed on it [44, 67, 70]. Topological information describes the structure of hierarchical phenomena; attribute information describes the data associated with those phenomena. Pandey, et al. [92] identify *data attributes* as information associated with nodes and edges and *structural attributes* as information associated with the tree topology itself. Our model encompasses both "siblings" as a coincidence of a parent's childrens' data attributes and *siblings* as a topology in its own right.

Our model centers on topology as the primary focus of visual representation suitability. Tree visualization techniques may be suitable for representing some elemental and composite topologies but not others. The model includes both basic and common composite topologies from other models, and expands the notion of topology beyond basic forms to encompass more complex composite topologies of interest in specialized domains. It expands upon the structure-attribute dichotomy to distinguish attributes inherent to a structure from attributes associated with it. It also recognizes that structures have inherent relationships with other structures, and that all of these various kinds of information can be intermixed in complex ways for visual data analysis. We refer to these respective abstractions of *form*, *feature*, *attribute*, and *blended attribute* collectively as *information categories*.

A topological **form** is a structural description of the topology itself. This information is in a sense "built-in" to the data representation and the resulting visual layout of constituent parts. The choice of visual representation affects how one perceives a topology in terms of those parts (elemental topologies and any subsumed composite topologies) and how they are visually laid out and encoded relative to each other.

A topological **feature** is a characteristic of a topological form. It is information that can be calculated from how the topology is composed and situated in its tree as a whole. For instance, a *path* has a length, a *siblings* has a count, and a *node* has a level. A *level* (form) also has a level (feature). (It is important to distinguish forms from the features often

used to refer to them.) The choice of visual representation affects how well a feature can be observed in the visual layout and the visual encoding of the topology and its parts.

A topological **attribute** is a data value associated with a topology. For instance, a gender might be associated with a *node* that represents a person in a family tree, and a count of page accesses might be associated with a *path* that represents a URL. The choice of visual representation affects how well attributes can be visually encoded with their topology.

A topological **blended attribute** is information that combines form, feature, and/or attribute information from a topology or its constituents. For instance, the average size of files in a folder (recursively) takes the *leaves* (form) of the *branch* (form) under the folder's *node*, sums the file size (attribute) of each *node* in the *leaves* and divides by the count (feature) of the *leaves*. One must consider the potentially very complicated visual interplay between the forms, features, and attributes that contribute to a blended attribute when determining representational suitability. We include two examples in our sample set.

Information Types and Values

Visualization types differ in their suitability for visually representing information from the four categories. We include an assortment of examples from each category with each topology. For a given topology and information category, an *information type* is a type of data in that category that characterizes or derives from the topology. Information types are data types in the usual sense—nominal, ordinal, quantitative, composite objects, data structures, etc.—but are thought of in terms of how well that data is conveyed through the visual representation of the topology in any given visualization type. (For the examples that follow, refer to the third column and corresponding rows in Figure 3.3.)

An information type of a **form** captures how well a topology conveys its topological role or relatedness to other topologies. For example, *association* is how well an edge conveys the parent-child relationship between its parent and child nodes. *Ancestor* is how well a bi-path conveys its relatedness to its ancestor node.

An information type of a **feature** captures how well a topology conveys some aspect of its topological character. For example, *diverging* is how well a tree conveys that it should be interpreted as diverging from root to leaves. *Level* (the type) is how well a *level* (the topology) conveys its distance to the root node. *Order* is how well a siblings conveys the order of the sibling nodes as stored in the tree data.

An information type of an **attribute** captures how well a topology conveys a type of data associated with it. For example, *item* is how well a node conveys its entity's identity. *Category* is how well a *siblings* conveys some classification of its sibling nodes as a set. *Quantitative* is how well a topology (node, edge, etc.) conveys a quantitative data attribute, similar to how well quantities are visually encoded in visualizations generally, but accounting for contextual effects of the topology being visually situated in its tree's layout.

An information type of a **blended attribute** captures how well a topology conveys the necessary form, feature, and attribute information that make up the blend. For example, *part-whole* is how well an edge conveys a child node as a part of its parent along with its sibling nodes. *Time* is how well absolute or relative chronological data associated with the tree is conveyed through positioning or ordering in the tree's layout.

In Figure 3.3, each information type is accompanied by a description (fourth column) and the specific *information value* (fifth column) that is to be visually represented in any given visualization type. For example, the *level* topology has a feature of information type *size* with a value defined as the count of items in the level. This means that size as the count of items in the level (the information value) is visually encoded as part of the visual representation *of the level topology itself*. This is in contrast to reading the size by visually scanning and counting the level topology's constituent nodes, which would be a more complex task involving how well the form of the level conveys the form of its node topologies. It is also in contrast to encoding an associated count attribute in the visual representation of the level itself, which might look the same but have meaning unrelated to the level's nodes.

3.4.3 Representational Suitability

Visualization studies have considered a wide range of criteria when evaluating the suitability of visualizations. Numerous heuristics have been proposed, typically organized into higher-level categories such as perception, cognition, interaction, tasks, and usability [47, 134, 41, 112, 125]. Our model narrows this focus to criteria essential for the effective perception of topological detail in static visual representations of hierarchical data, particularly at the scales targeted by most visualization techniques. Following Nielsen, et al. [88] recommendation to use three to five heuristics for evaluation, we identify five key criteria to model representational suitability: layout capability, encoding capability, channel accuracy, channel scalability, and visual side effects.

This selection is strongly grounded in information visualization literature, especially through perceptual and cognitive lenses. As Zuk, et al. [156] emphasize, visualization-specific heuristics are vital for uncovering issues that extend beyond general usability concerns, particularly when the goal is to amplify cognition. Each of our selected heuristics contributes to this objective by addressing different aspects of how structure and meaning are conveyed visually. While our set may not be exhaustive, it offers a theoretically grounded and practically useful framework for evaluating visual effectiveness, moving beyond generic assessments to meet the nuanced demands of hierarchical data representation.

Layout capability is a combination of layout *support* and layout *flexibility*. Layout support considers the spatial organization of topologies in a hierarchical visualization technique and how that organization imposes constraints on effective perception of topological information folded into that organization. Layout flexibility considers how much variation is possible for representing topological information, when those constraints are not absolute, and without breaking the design contract that gives the layout its character. For instance, the window size of a node-link diagram (Figure 3.1A) limits the amount of text and other detail that can be shown horizontally in nodes across a level, as well as the number of levels that can be shown. Nevertheless, the ability to adjust node width, height, and spacing—

up to a point—to suit many different labeling designs gives this visualization type layout flexibility and thus high layout capability overall.

Encoding capability is the ability of a tree representation to visually encode an information type effectively. Tree representations show topological structures implicitly or explicitly, which affects the kinds of data they can encode in those structures. For instance, squarified treemaps (Figure 3.1F) and circular treemaps (Figure 3.1H) implicitly encode edges (through containment), which severely limits the ability to encode attributes of edges. Similarly, dendrograms (Figure 3.1C) subtly encode nodes as vertical lines, with similar limitations. Representations of topological structure often have limited access to visual channels, which in turn limits the kind and amount of information that can be encoded in them, especially in ways that do not interfere with seeing the topological structure itself. For instance, an indented outline (Figure 3.1E) cannot encode quantitative information in the horizontal position of nodes (beyond perhaps small perturbations) because that channel is already used for indentation to convey level in the tree.

Channel accuracy is the accuracy with which a user can perceive the value of an information type through the visual channel(s) used to encode it. Much like in visualization design practice, assessment of accuracy draws from Mackinlay's perceptual ranking of visual channels [77], Gestalt principles as described by Wertheimer [143] and Pashler and Yantis [93], and one's own experience with the visualization types, their layouts, the topologies they emphasize, the graphical character of available marks, and the types of data to encode.

Channel scalability is the number of distinct data values that can be distinguished in the visual channel(s) used to encode that data. It is limited by how the visual encoding is situated in the layout as well as by the visual channels used in the encoding itself. For example, a node-link diagram (Figure 3.1A) might use color hue to differentiate at most 7–11 categories of sibling nodes, while laying out those nodes into hundreds of distinguishable horizontal positions. Encoding any information about topologies more than a few levels deep is effectively impossible in most treemaps (Figure 3.1F,H).

Visual side effects are unintended patterns in the layout and visual encoding of topologies and their constituents. Side effects can be useful in some cases to reinforce desirable visual patterns. In other cases, they can negatively impact perception of patterns, or even introduce artifacts. For instance, the arrangements of sibling nodes from left to right in a node-link diagram (Figure 3.1A) and from top to bottom in an indented outline (Figure 3.1E) emphasize orderings that may or may not exist. The same arrangement in a radial tree (Figure 3.1E) suggests a cyclic relationship without a strongly indicated start or end. The burst style of siblings in rings (Figure 3.1D) strengthens the sense of cycle in them, but weakens it for the other composite topologies, including the tree itself.

3.5 Applying the Model

The previous section describes how the model organizes tree visualization types and topological information, and proposes five criteria to assess the suitability of each visualization type for representing each kind of topological information. This section describes a method to apply the model by progressively incorporating the five criteria into an overall determination of the suitability of each potential pairing. The results are shown in Figure 3.3. To apply the results, a visualization designer should first identify which kinds of topological information are of interest to visualization users, then select suitable visualization types to include in the visualization design; see Section 3.6.5 for more.

Given the model and a set of suitability criteria, there are a number of ways one could go about determining an overall suitability. For instance, one could calculate a weighted average of suitability values determined for each criterion individually. Zuk, et al. [156] argue that "the process of heuristic evaluation may evolve just as the heuristics can evolve over time." Heuristics can be factored into an evaluation to account for their relative importance. To populate the model with suitability values for the sampled visualization types and kinds of topological information included in the research described in this chapter, we chose to

Topologies	Information Category	Information Type	Description of Information Type	Information Value	Connection												Containment				Alignment			
					Node-Link (Top-Down)		Radial Tree		Dendrogram (Top-Down)		Rings		Indented Outline (With Edges)		Treemap		ArcTrees		Circular Treemap		Icicle Plot		Sunburst	
					Possible	Suitability	Possible	Suitability	Possible	Suitability	Possible	Suitability	Possible	Suitability	Possible	Suitability	Possible	Suitability	Possible	Suitability	Possible	Suitability	Possible	Suitability
Node	Attribute	Item	A node representing one item in a hierarchy. Eg. one person in an organization	Identity of the Item	yes	very high	yes	very high	yes	high or none	yes	very high	yes	very high	yes	medium	yes	medium	yes	medium	yes	very high	yes	very high
		Set	A node representing a group of entities in a hierarchy. Eg. A node representing department in an organization hierarchy	Identity of the set	yes	medium	yes	medium	yes	very high	yes	medium	yes	high	yes	very high	yes	very high	yes	very high	yes	very high	yes	very high
		Quantitative	The numerical value associated with the node.	Magnitude of item	yes	low	yes	low	yes	high	yes	very low	no	none	yes	medium	yes	very high	yes	medium	yes	very high	yes	high
		Ordinal	The ordering of nodes perceived across the tree	Relative Magnitude of item	yes	high	yes	low to medium	yes	low	yes	low to medium	yes	medium	yes	medium	yes	high	yes	high	yes	very high	yes	high
		Categorical	The category of value associated with the node.	Type of Item	yes	high	yes	high	yes	very low	yes	medium	yes	medium	yes	medium	yes	high	yes	medium	yes	high	yes	very high
Edge	Form	Association	The basic association between child and a parent	Parent and child nodes	yes	very high	yes	very high	yes	high	yes	high	yes	very high	yes	high	yes	high	yes	high	yes	very high	yes	very high
	Attribute	Quantitative	The importance in terms of the value parent-child relationship carries in addition to the existence of a relationships in a tree.	Weigth of the edge	yes	low	yes	low	no	none	no	none	yes	very low	no	none	no	none	no	none	no	none	no	none
		Nominal	The category of the parent-child relationship in addition to the existence of relationship in a tree.	Label of item	yes	low or none	yes	low or none	no	none	yes	low to medium	yes	very low	no	none	no	none	no	none	no	none	no	none
	Blended Attribute	Part-whole	The relationship in which the child nodes make up parts of the parent node and combining them will constitute a parent node.	Proportion of an item	yes	medium	yes	medium	yes	high	yes	high	yes	high	yes	very high	yes	very high	yes	very high	yes	very high	yes	very high
Tree	Form	Root (Node)	A common ancestor to all the nodes in the tree.	Relative reference to a node item	yes	very high	yes	very high	yes	high or none	yes	very high	yes	very high	yes	very high	yes	very high	yes	very high	yes	very high	yes	very high
	Feature	Diverging	The ability of a tree to start with fewer nodes and grow in size into many nodes.	Increasing branching	yes	high	yes	very high	yes	high	yes	very high	yes	very high	yes	medium to high	yes	medium to high	yes	medium to high	yes	very high	yes	very high
		Converging	The ability of a tree to start with many nodes and shrink in size into fewer nodes.	Decreasing branching	yes	high	no	none	no	none	yes	medium	no	none	no	none	no	none	no	none	no	none	no	none
	Blended Attribute	Time	The chronological information associated with the tree from root node to leaf nodes.	Absolute and relative values of time	yes	high	yes	very high	yes	very high	no	none	yes	high	no	none	no	none	no	very low or none	yes	high	yes	medium to high
		Space	The ability of the tree to be laid out spatially on 2D space such that nodes can convey spatial data.	Position in space	no	none	no	none	no	none	no	none	no	none	no	none	no	none	no	none	no	none	no	none
Path	Feature	Direction	The direction of relation from an ancestor to a descendant. Ex Grand parent relationship vs Grandchild relationship	Relative reference to ancestor and descendant nodes	yes	very high	yes	very high	yes	very high	yes	high	yes	high	yes	very low	yes	medium to high	yes	high	yes	very high	yes	very high
		Path Length	A sequence of edges connecting an ancestor and a descendent.	Count of Item	yes	very high	yes	very high	yes	very high	yes	high	yes	very high	yes	medium	yes	high	yes	high	yes	very high	yes	very high
Level	Feature	Level number	The distinct levels of the tree that begin with the root node and increases in number with each additional connected edge.	Count of items	yes	very high	yes	high	yes	medium	yes	very high	yes	very high	yes	low	yes	very low or medium	yes	medium or high	yes	very high	yes	very high
		Size	The number of nodes in a level.	Count of items	yes	very high	yes	very high	yes	low	yes	low	yes	medium	yes	very low	yes	very low	yes	very low	yes	very high	yes	very high
		Sibling Set Count	Number of groups of siblings in a level.	Count of items	yes	high	yes	high	yes	low	yes	high	yes	medium	yes	very low	yes	very low	yes	very low	yes	high	yes	high
Branch (Subtree)	Feature	Size	Number of nodes in a branch	Count of items	yes	high	yes	high	yes	medium	yes	low	yes	high	yes	low	yes	high	yes	high	yes	medium	yes	medium
		Depth	The length of the longest path in a branch.	Count of items	yes	high	yes	high	yes	medium	yes	very high	yes	high	yes	low	yes	medium	yes	low	yes	very high	yes	very high
Siblings	Feature	Order	The ordinality of the siblings arising from the placement of the sibling nodes in a sibling topological structure.	Relative position of an nodes	yes	very high	yes	low to medium	yes	high	no	none	yes	very high	yes	very low or none	yes	very high	no	none	yes	very high	yes	low to medium
	Attribute	Category	The categorical nature of the siblings rising from the categorical data of the sibling nodes under a parent node in a tree.	Type of items	yes	low or high	yes	very low	no	none	yes	medium	no	none	no	none	yes	medium to high	no	none	yes	very low	yes	very low
Bi-path	Form	Ancestor (Node)	The ancestor that connects the two descendants using the two paths.	Relative reference of node	yes	very high	yes	very high	yes	high or none	yes	very high	yes	very high	yes	very high	yes	very high	yes	very high	yes	very high	yes	very high
	Feature	Bi-Path Length	A path connecting two descendent nodes through a common ancestor.	Count of edges	yes	very high	yes	very high	yes	very high	yes	high	yes	high	yes	very low	yes	medium to high	yes	high	yes	very high	yes	very high

Figure 3.3: The suitability of common tree visualization types to visually represent topology information.

prioritize *encoding capability* as the primary criterion and successively factor in the other criteria as a diminishing series of moderating influences.

For each potential pairing of visualization type and information to represent, we first determined whether or not it is *possible* to visually represent the topological *information value* through available and topologically appropriate visual channels in the visualization type. We then assessed the *suitability* of those visual channels for that purpose. In Figure 3.3, a pair of columns under each visualization type show the corresponding results. The *possible* column simply states whether or not ('yes' or 'no') the tree structures in the visual representation are capable of encoding the given *information value*. The *suitability* column summarizes how effective those visual channels are—as determined by the method—for representing the *information value*. We used a decision tree-based model with multiple levels to determine suitability, with one level per criterion. We ordered the levels on the presumed relative importance of the five *suitability criteria*.

The **first level** of the decision tree applies the *encoding capability* criterion to assess if a technique supports encoding of the information value. The **second level** applies the *channel accuracy* criterion to assign a value of *very low*, *low*, *medium*, *high*, or *very high*, then progresses to the next level. In both levels, for cases with a value of *medium*, *high*, and *very high*, the method progresses to the next level, and for other cases, the method terminates with a value of *very low*, *low* or *none*. In the **following three levels**, whenever application of the corresponding criterion—*visual side effects*, *layout capability*, then *channel scalability*—produces a value of *low*, it assigns that value to the overall suitability and proceeds to the next level, otherwise it adjusts the running value to *medium*, *high*, and *very high*, factoring in the criterion applied. There are two exceptions. In the **fourth level**, the *layout capability* criterion terminates with an overall suitability value of *none* if the layout does not support visual encoding of the considered topology. In the **fifth level**, the channel scalability criterion produces a range of suitability values in some instances. The method also terminates whenever the value of suitability after any level is *low*, *very low*, or *none*,

because the remaining criteria are of decreasing importance and will have minimal additional impact.

3.6 Discussion

Our motivation to develop a model of representational suitability grew in part from our own experiences working with tree visualization techniques. It is difficult to assess the capabilities of any given technique against the many available ones [113] in the absence of more established theoretical understanding of the tree visualization design space [45]. Moreover, significantly comprehensive coverage through empirical studies of the practical effectiveness of tree visualization techniques for representing different kinds of topological information may be slow in coming. In contrast, the sampling of both visualization types and topological information in the populated model is indicative of significantly many use cases across a variety of domains.

The populated model also serves a broader purpose of piecing together existing understanding of the capabilities of tree visualization techniques, and by serving that purpose may facilitate systematic study of the tree visual design space. The model also calls for a wider range of general evaluation methods centered on visual representations of topology. The rest of this section analyzes the results, method, and model, and discusses outcomes and implications for ongoing work.

3.6.1 Observations and Overall Patterns

After we finished applying the model of representational suitability to arrive at values for *possible* and *suitability* for the tree representation techniques shown in Figure 3.3, we started looking for patterns in the columns for each tree visualization type and in groups of columns categorized under our tree organization principles of *connection*, *containment*, and *alignment*. We also looked for patterns in individual rows, and in groups of rows corresponding to each

topological structure. The kinds of patterns that we are more interested in are outliers, distributions, and commonalities and differences between columns and rows. Several key observations that we made by referencing values of *possible* and *suitability* follow.

Individual Tree Variance — The *suitability* columns in Figure 3.3 suggest that there is quite a bit of variation across visualization types in terms of their ability to encode some kinds of topological information *at all*. For instance, the possibility to show almost all kinds of information is *yes* in *top-down node-link* techniques, but *no* for many kinds of information in *circular treemap* techniques. The other techniques provide support in between these extremes. Much of the variation is concentrated in particular topologies, particularly to show parent-child and sibling relationships in *edges*, *trees*, and *siblings* topologies.

Inter-Category Tree Variance — Looking at cell values across all columns in Figure 3.3 reveals that the visualization types in the *alignment* category generally have more instances of high suitability compared to *connection* and *containment* techniques. Between those two categories, the *connection* category shows a stronger pattern of high suitability than *containment*. These observations confirm that visualization types can be meaningfully grouped into categories defined by Gestalt principles like in our model, and hint that there may be other meaningful ways to group them. Comparison of patterns between visualization types may also help in formulating and assessing alternative categorizations. Our observations are consistent with the results obtained by Barlow, et al. [15]. Turo and Johnson [132] discuss similar issues with how aspect ratios impact comparison of nodes, especially siblings, in treemaps. The superior overall suitability of *alignment* techniques might be due to their ability to show both structure and attributes with reasonably stable aspect ratio in which leaf nodes can utilize visual channels effectively.

Intra-Category Tree Variance — Within the observed patterns of *inter-group variance* among the visualization categories, there are also noticeable differences in suitability of visualization types within each category resulting from variation in their layout and encoding capabilities. For instance, the axial layout techniques generally perform better than

the radial layout techniques within each category. There is also variation among radial layouts and among axial layouts. For example, radial trees (Figure 3.1B) perform better than rings (Figure 3.1D), and ArcTrees (Figure 3.1G) perform better than squarified treemaps (Figure 3.1F), both of which are in keeping with decreased stability of node positions and sizes.

General-Purpose versus Specialized Trees — Looking across the row in Figure 3.3 for any given information value, the *possible* columns suggest that some information, like the length of a path or the number of items in a level, can be encoded in most of the techniques we considered, but other information like a category or magnitude associated with a node are possible in only some techniques. The *suitability* columns indicate that while some techniques like *top-down node-link* (Figure 3.1A) can encode most kinds of information, suitability is *very high* or *high* only for some of them. This pattern suggests that some techniques are more general-purpose and can be used in many circumstances with at least reasonable suitability, but other techniques like ArcTrees (Figure 3.1G) are more special-purpose, with many extreme values of *very high* or *none* suggesting which purposes a technique is likely to support.

Topological Complexity — Scanning the rows for topologies *node*, *edge*, *path* and comparing them to the rows for *level*, *tree*, and *siblings*, we can see that there are more suitability values of *very low* and *none* as the topologies grow in structural complexity. This indicates that the ability to effectively encode information about a complex topology *as a whole* (e.g., by coloring an entire *level* rather than its constituent *node*s) is more challenging for some techniques than others. For instance, *connection* techniques that represent parent-child relationships explicitly are generally much more suitable for encoding information about composite topologies than *containment* techniques that represent parent-child relationships implicitly.

Intra-Topology Variance — Looking at the rows for a given topological structure, for instance *node* or *siblings*, we can see quite a bit of variation in *possible* and *suitability* values

among the rows themselves. This suggests that considering individual kinds of information may be insufficient to determine the overall suitability of a technique to represent different kinds of information about a given topology. One should consider all of the topological information types that one needs to represent, especially for topologies that exhibit high intra-topology variance over many visualization types (as *siblings* does) or in particular categories of visualization (as *node* does in the *connection* category). The populated model reveals significant patterns of variation in how suitable different tree visualization types are for representing different kinds of topological information. Despite sampling only common tree visualization types and a small subset of possible kinds of topological information, the populated model clearly surfaces patterns that comport with practical understanding and focused studies of the capabilities of tree visualization techniques. This hints at both the descriptive and predictive power of the model. The results also suggest that the model successfully surfaces not only overall suitability but also nuanced understanding of how well different techniques represent diverse hierarchical structures and attributes. Both are needed to inform visualization design choices. In the next section, we tease apart some of these nuances by analyzing the method that led to the results in the populated model.

3.6.2 Assessing the Method

The results in the populated model are heavily influenced by the method we chose to assess the five criteria individually and in combination. The method applies a decision tree in which each level of the tree considers one criterion to answer a particular question about suitability. The application of the criteria at the top of the decision tree draws from more established theory and practical understanding than levels farther down, in which the criteria have less such understanding to draw from. The criteria become harder to assess and produce less definitive conclusions, resulting in increasing uncertainty in the running suitability value. In some cases, this uncertainly calls for passing down a range of suitability values. Tolerating uncertainty in this way relaxes the difficulty of interpreting the criteria and reduces the

complexity of applying the model overall, while still leading to informative conclusions. It also allows for modification of the method to add or remove criteria (especially at lower levels to avoid disrupting better understood criteria at higher levels), strengthen or weaken the influences of individual criteria, or change their order of consideration. This separation of method from model makes the model and its application both extensible and flexible.

In the method, we first assess encoding capability and visual channel accuracy to establish a base determination of suitability. The successive assessments of the other three criteria can restrict that determination, but not expand it. This prioritization of the first two criteria over the other three acknowledges both the theoretical, empirical, and practical understanding of visual encoding and visual channels and the relative lack of such understanding about tree layouts, their impact on scalability, and the visual side effects they can induce. We also prioritize last the criteria that are the most uncertain to assess, and thus are the most likely to produce suitability values between the five allowed values. This approach mitigates the propagation of increasingly ambiguous determinations as criteria are successively assessed in subsequent levels.

Some visual representations encode a topology differently in different parts of the layout, possibly resulting in multiple values of suitability for the same topology information. For instance, in a squarified treemap (Figure 3.1F), leaf nodes have an area to encode quantitative attributes, but non-leaf nodes are occupied by their descendent leaf nodes and cannot encode data in a similar way, producing two different values of suitability for the same kind of node topology information. Similarly, criteria like *channel scalability* can produce a range of the five allowed values, in keeping with how a technique might be suitable, but differently so, at different channel scales. As with criteria that produce ambiguous determinations, criteria that produce multiple and/or ranged determinations should be positioned later (preferably last) in the decision tree.

The method prioritizes criteria in the levels of the decision tree according to our best estimation of their relative importance in visual representation and thus their likely influ-

ence on suitability. It also prioritizes them on their likelihood to produce a definitive result, multiple results, or a range of results. A different or extended set of criteria would call for modification of the method to prioritize and/or weight them accordingly. Regardless, applying the method produces umbrella results that account for many representational factors, and does so on techniques that are exemplars of their visualization types. The results should be interpreted holistically.

The descriptive and predictive power of the model is limited by the discrimination between suitability values supported by a five-level scale (*very low* to *very high*). When we first applied the method with a two-level scale of *low* and *high*, we were unable to discern patterns of variation. Expanding the scale to five levels revealed much stronger patterns of variability. Even so, we observed that the method could tolerate a three-level scale (with *medium*) or even the two-level scale for some criteria without much loss of detail in suitability results. Using a less precise scale was helpful when criteria were particularly hard to assess due to complexity or variety in the visualization type or kind of topological information under consideration. The ultimate choice of five levels matches our ability to qualitatively assess both individual criteria and overall suitability without imposing too little or too much precision on the determination and interpretation of suitability values.

3.6.3 Assessing the Criteria

The model prescribes five criteria for assessing the suitability of visualization techniques to represent information. Each of the five criteria focuses on a specific aspect of how well a technique can layout and encode particular kinds of structure or attribute information, if at all. Although they were developed for the purpose of assessing visual representation of tree structures in tree visualizations, the criteria themselves are agnostic to data structure and visualization type, and could be applied equally well to visualizations of tabular data, time data, graph data, etc. Nonetheless, we focus here on how well the criteria work for tree data and visualizations.

The criterion of **encoding capability** expresses the requirement of a technique to effectively convey structure and attribute information about a topology. The *form* of a topology can be represented through explicit or implicit encoding of its constituent elements and compositional structure, but the representation of a *feature* or *attribute* must build upon the representation of its *form*, and thus requires a way to explicitly encode its information type and value. For instance, one can encode the form *association* of an edge topology in a treemap implicitly (through containment), but cannot encode an *attribute* of the edge due to the implicit encoding of its *form* (at least *not in the edge topology itself*; one could encode the *attribute* in the representation of the *form* of the edge's parent node). Some visual channels for explicit encoding are intrinsic to a technique's design, like the area of nodes in a squarified treemap (Figure 3.1F), and may or may not be available to encode data. Other visual channels not intrinsic to the design can be utilized flexibly to represent feature and attributes, like color saturation, hue, and texture in the treemap's nodes. Even if a visual channel supports explicit encoding, such that the model predicts high representational suitability, one needs to make sure not to overload the channel with multiple information values (whether from the same or different topologies).

The criterion of **channel accuracy** for a *feature* or *attribute* is applied by considering how it can be visually encoded as quantitative, categorical, or nominal data in the visual channels of marks [77, 80] in the usual way. However, one must also account for how the data is encoded as a part of the representation of an elemental or composite topology, including whether it can be perceived unambiguously within both its immediate topological context and in the visual representation as a whole. For a *form*, channel accuracy is assessed by how well one can identify it within its tree and distinguish its structure (through its constituent elements or otherwise) from those of other topologies. It is similar to assessment of encoding capability but can also account for additional encoding beyond a technique's usual layout of the structure, such as by drawing a bubble around an entire level, path, or branch.

The criterion of **layout capability** is applied by considering how visual channels are

used to represent the constituent elements of a topology within a layout, how consistent those uses are for the elements, and the extent to which those uses can vary while preserving the perceived structure of the topology. Layout capability for any given topology can be difficult to assess because the layout of the topology is embedded in the layout of the tree as a whole. One can observe how the topology's constituents are laid out relative to each other, and how the topology affects the appearance of other topologies (and vice versa). For instance, the radial positioning of nodes in a radial tree (Figure 3.1B) has high layout capability for levels and siblings because their nodes are consistently located at the same radius, but lower layout capability for paths and branches due to angular splaying of edges from parent node to child nodes, especially in wide branchings.

The criterion of **visual side effects** is difficult to apply because it is something of an abstract catch-all for different phenomena that can reinforce or disrupt perception of structure or attributes, and the number and character of those phenomena are largely unexplored. In practice, the visual side effects criterion overlaps heavily with the channel accuracy and layout capability criteria. While those two do partially factor in effectiveness of visual encodings individually and relative to each other, they do not directly consider reinforcement or disruption phenomena as such. Sometimes, visual side effects are artifacts that suggest non-existent topological information. A particularly common side effect is the perceived ordering of nodes in *level* and *siblings* topologies. This side effect is very strong in top-down node-links (Figure 3.1A) but weak to absent in squarified treemaps (Figure 3.1F). In applying the criterion, we considered only side effects that create such perceptual anti-patterns among sets of elemental topologies or composite topologies relative to each other.

In applying the method to populate the table, the criterion of **channel scalability** was rarely a significant factor in determining overall suitability. Most visualization types readily support encoding the different data values in the kinds of data sets they are designed to support. However, there were conspicuous exceptions. Some visualization types would do well on the first four criteria only to show weakness or fail at lower than expected scalability.

For example, the squarified treemap (Figure 3.1F) and the icicle plot (Figure 3.1I) can represent many *siblings* topologies without difficulty, but indented outlines (Figure 3.1E) scale very poorly in this regard due to limited vertical space to show siblings' nodes. Despite being the last factor to consider, scalability can be critical and should not be overlooked, especially when assessing the suitability of a less studied visualization type to represent a complex topology.

Even when criteria are well understood, they can be difficult to apply well. All of the criteria likely require substantial expertise in practical visualization design to apply. Comprehensiveness, consistency, and correctness are needed to produce results that can serve as reasonable and meaningfully accurate predictions of suitability. Inadequate factoring of poorly understood criteria at the early stage of the method can produce ambiguous or inconclusive results. It is advisable to start with well-understood and easily applicable criteria as in the method used. A weighted approach might help by making application of the criteria more independent, but would require determination of weights to use in lieu of the cascading weighting baked into our method's decision tree. Nevertheless, we expected application of the criteria and the method to generalize well over the sample set of visualizations, and found that to be the case in collecting the results shown in Figure 3.3.

3.6.4 Assessing the Model

Well-known collections of tree visualization techniques [113, 72] attest to their rich history and diversity. There are many useful ways to organize them into categories and taxonomies. The categorization in our model is not new or even particularly interesting in this regard, nor is it meant to be; we chose it as a simple example for purposes of demonstrating and assessing the model as a whole. It could be readily extended to facilitate exploration of suitability patterns within and between other groupings of techniques. For example, the *connection* category can be further classified into level-based and non-level-based techniques. The *containment* category could be further broken down by layout type [135] or space utilization [111].

Other surveys and taxonomies have *general* [111] or *hybrid* [113] categories that could extend the model's categorization and sample set of visualization types. In this way, the model is not so much a producer of a categorization scheme as a consumer of existing and new ones. There is ample opportunity for future work to explore application of the model to more diverse collections of visualizations and their various categorizations. Importantly, the model hints at ways to extend the tree visualization design space by visualizing information not only in the visual representations of nodes and edges, but also in the visual representations of composite topologies themselves, turning all topologies into first-class visual citizens.

There is a similarly diverse space of potentially interesting and useful topological structures to explore. The small set of common composite topologies sampled in the populated model is just a starting point. The set can be readily extended to include additional topologies from particular domains and from analyses of hierarchical data in general. Increased capacity to design suitable visualizations to explore and analyze specialized and even idiosyncratic topologies might even facilitate new thinking about complex structures in hierarchical phenomena. Adding an assortment of blended attributes and even compositions of compositions are important directions to extend the model and inform ongoing development of the space of tree visualization techniques. Visualization designers could choose from a range of visualization types and topological forms to meet general and domain-specific analysis needs.

Beaudouin-Lafon [17] suggests that models of interaction can be evaluated based on their descriptive power, evaluative power, and generative power. Our model's **descriptive power** is in its ability to suggest suitable ways to visualize information about topological forms, features, and attributes both individually and in combination. Its **evaluative power** comes from its ability to assess different visualization types and categories in terms of their ability to show specific topological information and to provide recommendations of alternatives for particular hierarchical data types and tasks. Its **generative power** comes from its ability to extend to new visualization types and topological structures. The model can apply to new views of known topological structures and to new topological structures in existing views.

For each topological structure, the model can readily extend to new features, attributes, and a potentially extremely rich space of blended attributes that surfaces inter-topological relationships as targets for visual representation.

3.6.5 Putting the Model into Practice

Tree visualization revolves around topologies. Any given visualization technique may support a variety of tasks, each focusing on particular topological information. Visualization designers need to select techniques that suitably represent that information to support exploration and analysis tasks. The main purpose of our model is to inform that selection. However, selecting suitable techniques can be challenging. Needs can range from a simple task on one common, basic topology to a complex suite of tasks spanning many unusual, complex topologies. In this section, we present a series of increasingly complex scenarios to suggest how the model can be put into practice, and extrapolate to future opportunities for development and application.

One Task, One Topology — A historian works with genealogical data, and often wants to see the number of generations separating a living person from a noted ancestor. The designer looks up *Path, Feature, Path Length* in the populated model, scans the row for suitable visualizations, sees several options with *very high* suitability, selects icicle plot, and proceeds to find/implement one for the historian to use. In this scenario, the designer simply needs to find a suitable visualization for only one kind of topological information.

Multiple Tasks, One Topology — A software maintainer wants to track the number of files three levels deep in a source code tree and the number of files at that level in their respective parent directories. The designer looks up *Level, Feature, Size* and *Level, Feature, Sibling Set Count* in the populated model, scans the two rows for suitable visualizations, sees four options with *very high* suitability to represent the first number and *high* for the second, selects sunburst, and proceeds to find/implement one for the maintainer to use. In this scenario, the designer needs to weigh different suitability values for two kinds of information

to find one visualization that is most suitable overall.

One Task, Multiple Topologies — The historian wants a new feature to see the living person, the noted ancestor, and the separation in generations between them. This task involves information about node and path topologies. The designer starts by looking up *Node, Attribute, Item* and *Path, Feature, Direction* in addition to *Path, Feature, Path Length*. The designer looks at their suitability values in the column for each visualization type, considers the visualizations with *very high* suitability, decides to switch to a radial tree, and proceeds to find/implement one for the historian to use. In this scenario, the designer needs guidance on selecting one visualization that can show all of the information for the task simultaneously. In general, a designer would aim to support as many tasks as possible in as few views as possible, preferably one. Because tree views often exhibit significant variation in suitability for different topological information, finding a single view that works for multiple kinds of information may be difficult or impossible in practice. The designer could use multiple tree visualizations that are collectively suitable to support the task. The designer would need to carefully consider the additional difficulty of performing tasks across multiple views. Modeling suitability to account for such difficulties is an important topic for future work.

Multiple Tasks, Multiple Topologies — The maintainer wants to switch between seeing the size of all files, the kind of edits last made in each package, and the modification times of all files in the source tree. These three tasks involve information about node, siblings, and tree topologies, respectively. The designer looks up *Node, Attribute, Quantitative* for the first task, *Siblings, Attribute, Category* for the second, and *Tree, Blended Attribute, Time* for the third. Scanning the three rows, the designer sees that several visualizations have no suitability for at least one task, and the rest have mixes of suitability. Good candidates appear to be node-link, radial tree, and either icicle plot or sunburst, but each is most suitable for a different task. The designer compromises and selects node-link and icicle plot as two complementary visualizations that together support all three tasks sufficiently. In

this scenario, the designer could choose the most suitable view for each task, fewer views that cover the tasks less well, or a single view that covers some tasks well but others poorly. Strategies to minimize the number of views while supporting as many tasks as possible and maximizing collective suitability is another topic for future work.

The model works well as a predictor of the effectiveness of *static* visual representations for seeing topological information. However, the above scenarios challenge the predictive capabilities of the model and emphasize the need to develop strategies to support multiple tasks on multiple topologies across multiple views. Looking forward, a clear next step is to develop ways to interact with that topological information. Interaction designs are needed to specify which topological information to show and to navigate within and between topological structures inside each visual representation. Interactions could support direct manipulation editing of attributes directly within the visualizations [105] and possibly even extend to general editing of the topological structures themselves. Given the frequent need to utilize multiple views to represent multiple aspects of hierarchical data, another key direction of future work is to develop models and methods of coordination [100, 140] to connect interactions with topologies across multiple views. Finally, the interactions and coordinations will also need models of suitability to describe and predict which interactions and coordinations to use for exploration and analysis with different kinds of topological information in different types of tree visualizations.

3.7 Summary

We developed and applied a model to assess the suitability of visualization techniques for visually representing structures and attributes in hierarchical data. The main contribution and strength of the model is its ability to predict how likely visualization techniques are to effectively represent specific topological structures and the information associated with them. The model helps to understand the representational capabilities of tree visualization

techniques for both basic and composite topological structures, and offers guidance for selecting specific techniques well-suited to particular information seeking tasks. The model is also extensible in its ability to encompass new tree visualization techniques and new kinds of topological structures encountered in different domains, and flexible in its ability to accommodate alternative criteria for assessing suitability. We populated the model with common visualization techniques and topologies, revealing strong patterns of similarities and differences in the suitability of the techniques, suggesting opportunities to apply the techniques individually and collectively to support diverse exploration and analysis needs. Moving forward, we plan to build on the model to develop interactive support for editing topological structures directly inside visualizations and use the results to inform task design for user studies of tree visualization techniques.

Chapter 4

C4D3: A View-level Abstraction and Coordination Library for Building Coordinated Multiple Views

4.1 Overview

D3 [24] is a popular and effective library for the development and deployment of visualizations in web pages. Numerous applications testify to its accessibility and expressiveness for representing and manipulating page content. By eschewing toolkit-specific abstraction, D3 gains representational transparency, but at a cost of modularity. Relatively few D3 applications compose multiple views or support coordinated interaction beyond basic navigation and brushing. Rather than relegate view composition to custom integration code, we overlay D3 with a view-level abstraction that utilizes a general parameter sharing model to offer simple yet flexible composition of coordinated multiple views (CMV) while preserving the expressiveness of individual D3 components. Coupling of event handling to shared parameters recasts modeling of interactive state and simplifies the declarative specification of interactive dependencies between views. We present an example of an extensively coordinated visualiza-

tion, illustrative code to show CMV construction in C4D3, and demonstrate how view-level abstraction can reduce the code needed to compose complex D3 visualizations.

4.2 Introduction

Web-based interactive visualizations are increasingly used for data exploration and analysis. While a single interactive view is often sufficient to present one perspective on a data set, coordinated multiple views (CMV) with interactive capabilities are usually required to support multifaceted perspectives for comprehensive analysis. Yet, it remains challenging to build CMV in popular low-level web-based libraries due to their lack of coordination abstractions [34]. Higher-level design tools like Tableau and Power BI are conversely limited to predefined sets of relatively simple coordination templates.

Data-Driven Documents (D3) has emerged as a widely adopted library for embedding visualizations within web pages [24]. D3 offers an accessible, expressive, declarative language for representing and transforming data in the Document Object Model (DOM) of a web page. Furthermore, it enables manipulation of that data by associating interaction events with transformations, thereby supporting a wide variety of interaction visualization designs.

The tendency in web-based visualization design is to favor simpler designs suited for communication and explanation. Composite designs, like those with CMV constructions commonly utilized in exploration and analytical applications, are relatively rare. This is not exclusive to visualizations built using D3. Visualizations built using other libraries and toolkits [42, 23, 110, 108] exhibit the same tendency. This scarcity can generally be attributed to limitations in language and feature support, as well as to the inherently more complex nature of designs comprising multiple views.

The scarcity of complex CMV constructions using D3 may arise from its goal to avoid imposing a toolkit-specific abstraction of visual representation. Unlike many visualization libraries, D3 does not explicitly define the concept of a "view"; rather, it leaves the defini-

tion of a view largely to developer discretion. This flexibility allows D3 to transform page structure and styling in ways that may not align with commonly recognized view types. However, D3's constraints on interaction handling and encapsulation of event processing make it challenging to combine different interaction channels that rely on the same events, especially when conflicting events occur [108] (such as how D3 relies on mouse click and drag for both brushing and panning). Composite visualizations typically involve not only multiple views, but also multiple coordinated interactions between those views to support diverse manipulations of data and how it appears. As the number of views and the complexity of interactions increase, so does the challenge of managing dependencies between D3 fragments corresponding to each "view" and its associated portion of the DOM. Vega-lite [108] and the interaction semantics introduced in Reactive Vega [110, 109] seek to address these concerns, but do not directly address the issue of composability of CMV in visualization designs. Selection parameters in Vega-Lite convert user interactions into data queries that share state between views. However, these selection interactions are restricted to point and intervals, limiting expressiveness when trying to design custom selection interactions, such as selection of arbitrary data points in views, or when implementing non-selection-based complex data queries that depend on navigation.

C4D3 contributes a view encapsulation and interaction abstraction library implemented on top of D3. It aims to support expressive design and elegant specification of coordinated multiple views by adopting the three main design principles of the Live Properties architecture in the Improvise visualization system [138]. First, modular view implementation facilitates the design and reuse of common and custom view types in CMV constructions. Second, clean separation of view implementation and coordination design makes building and modifying complex CMV constructions simpler and easier than with monolithic approaches. Third, lifting the storage and management of interaction state from view-local to visualization-global, while keeping the interpretation of state and changes to it view-centric, facilitates flexible and meaningful sharing of state to implement expressive and understand-

able coordination designs. We followed these principles to develop C4D3 with the objectives of increasing transparency, extensibility, modularity, composability, consistency, and consonance of web-based CMV visualization design.

In this chapter, we present C4D3's coordination architecture based on Live Properties, a JavaScript implementation of that architecture, the design and implementation of a view wrapper to adapt D3 rendering and interaction handling code for coordination, and several common view types implemented as D3 code within the wrapper. We demonstrate C4D3's capabilities via a reproduction of a classic CMV visualization with extensive navigation and selection coordinations, illustrate the process of implementing a simple CMV design with C4D3, and assess our progress toward achieving our objectives for C4D3. We conclude that C4D3 can support an incrementally expanding collection of D3 components as view modules that can be quickly and easily instantiated and interconnected to implement expressive CMV patterns. While implementing particular views for use in a multiple view visualization design in C4D3 does usually require writing at least some custom D3 code for each view, the effort is similar to writing different components in D3 without C4D3, and the modularity of the C4D3 view wrapper amortizes the effort needed by allowing substantial code reuse between similar views, much like how D3 designers often modify existing D3 examples rather than start from scratch.

In the remainder of this chapter, we begin by discussion of the most relevant work in this area followed by presenting an example Ions visualization (Figure 4.1) built using C4D3 to demonstrate the kinds of applications possible. Next, we list the development objectives considered during the creation of C4D3 and provide an explanation of its architecture, as well as the various steps involved in the design and implementation of CMV examples using a simplified food visualization CMV example (see Figure 4.4). We then describe the various stakeholders of visualization and how they make and use different parts of the library. We then provide another illustrative Covid-19 visualization example (see Figure 4.9) describing the design process with regard to the developmental objectives

considered. Finally, we discuss our findings and developmental objectives. The interactive examples and code for the Ions Visualization and Covid-19 visualization can be found online at https://omkarchekuri.github.io/c4d3.

4.3 Related Work

C4D3 introduces a view abstraction and a general purpose coordination layer adapted from the Improvise architecture [138] for specifying and building coordinated visualizations as data flow graphs. As an implemented library, C4D3 provides coordination capabilities that build on the existing visual representation capabilities of D3, thus applying a separation of concerns between D3's visualization rendering code and C4D3's coordination management. This separation allows for swapping out D3 code with other libraries or combining visual representations from multiple libraries to compose a CMV visualization.

Research on coupled interaction have been pivotal in the evolution of information visualization. Constraint-based UIs like Garnet [82] and Rendezvous [60], and seminal visualization systems such as Tioga-2 [9], Visage [104], Spotfire [6], and XGobi [28] heavily influenced work on CMV systems including IVEE [8], DEVise [73], Snap-Together Visualization [90], GeoVISTA *Studio* [124], CViews [25], and Improvise [138]. This long history underscores the persistent challenge of balancing design accessibility and expressiveness and bridging capabilities of visualization systems [43].

More recently, Nebula [34] provides an abstraction layer for constructing coordinated visualizations using macros to abstract coordination specifications. It offers efficient CMV design through textual specification of supported coordination patterns. The extensibility of coordinations and implementation of complex coordination patterns becomes challenging when users are restricted to the existing natural language patterns backed by higher-level language parsers. Text specification allows one-way binding from the source visualization that accepts the interaction to the destination that shows the result. This makes specification

accessible and more natural, but limits design expressiveness. The architecture is also limited to interactions that trigger updates only upon completion. Nebula is similar to C4D3 in how it separates coordination from visual representation, despite their substantial differences in coordination specification and expressiveness.

User interface frameworks like React offer a component-based architecture in which UI elements are treated as self-contained components organized in a hierarchy. React uses a combination of strategies to coordinate view components in the component hierarchy. It uses props, callbacks, and context to coordinate the parent component with the child component. Additionally, libraries like Redux offer a pattern for organizing global state and ensuring predictable state management for JavaScript applications built on React. Although these libraries together support state management for building views (as UI components) that can be coordinated using shared state, they do not provide an abstraction layer for specifying coordination between views. They rely on developers to implement custom logic to manage complex view interactions and state changes across coordinated views.

CViews [25] offers a coordination model in which objects in a coordination space are directly linked to views through translation functions. Use-Coordination [65] provides a reactive library based on this coordination model, implemented in the *ReactJS* library. One shortcoming of the coordination abstraction layer in this model is that views are directly dependent on coordination objects; the translation functions tightly couple coordination objects to view implementations. In contrast, Live Properties [138] provides a level of indirection such that coordination objects are not directly coupled to views but instead are linked to properties of the views. Coordination objects can provide information of any type for views to share and translate for individual use. This indirection expands coordination to allow sharing not only of interaction characteristics, but also of behavior and appearance, thereby providing an abstraction layer that is flexible enough to express a wide variety of view couplings far beyond those usually encountered.

C4D3 builds on D3's capabilities to make coordination capabilities accessible to the wide

community of D3 users with implementation cost at or below that of D3 on its own. It provides flexible coordination between views by abstracting the coordination from the visual representation and interaction with it, while still maintaining the flexibility and expressiveness to author open-ended coordination patterns. It also optimizes view updates in response to interaction by only updating the affected parts of the DOM, thereby allowing intermediate updates during protracted interactions and with it a sense of interactive immediacy (depending on the latency of the updates themselves).

4.4 Example: Cubic Ion Trap Visualization

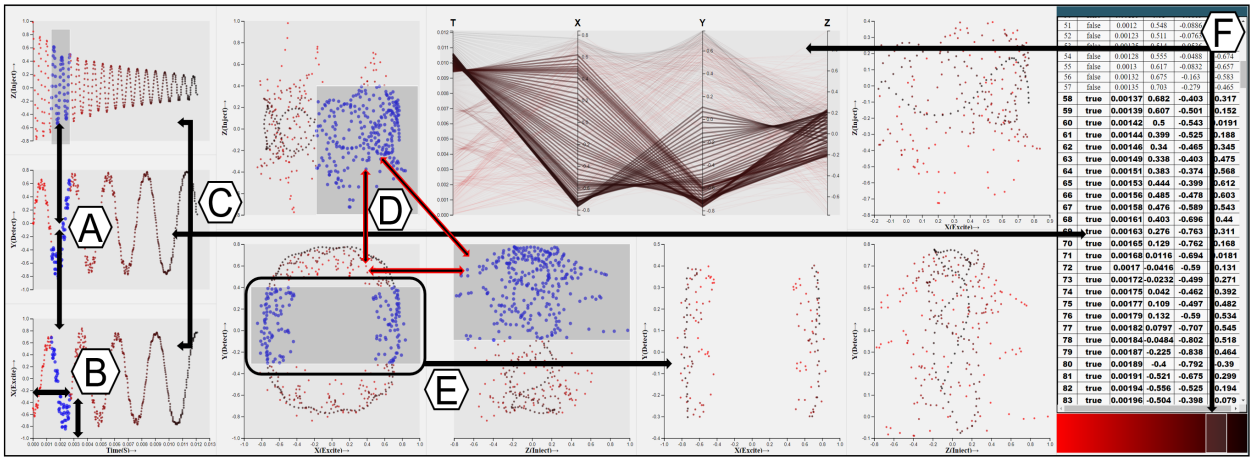


Figure 4.1: Reproduction of various coordinations in the Improvise cubic ion trap visualization [138] with 12 C4D3 views: (A) three time series plots synchronize scrolling horizontally; (B) independent vertical scrolling in those plots; (C) shared selection between scatter plots and the table; (D) overview SPLOM showing three sides of the cube; (E) detail SPLOM zoomed to ranges selected in the overview; (F) a selected range in a color slider filters items in a parallel coordinate plot.

Figure 4.1 shows a reproduction of the Improvise cubic ion trap visualization comprising 12 coordinated views created with C4D3. The C4D3 library provides a variety of view modules, each tailored to exhibit distinct behaviors and appearances using D3 internally for visual representation and interaction handling. The example employs two types of scatter plot module: one to provide an overview, and the other to present a detailed view. It also

employs a parallel coordinate plot module, a table view module, and a gradient plot module. A significant portion of a view module consists of D3 code for constructing and updating the view, while other code sections pertain to methods for updating the view's internal state stored in the view wrapper. To create new types of views as view modules in C4D3, one makes a copy of the wrapper template and customizes it to include the desired D3 code for each view. Each of the individual views are appended to the DOM and positioned based on the view parameters that are provided when creating the views.

To construct the CMV for the cubic ion trap data, we first need to identify the data relationships to explore. First, how does the behavior of ion cloud change over time in the three cube dimensions? For that we need to be able to view the behavior of the ion cloud in three different axis across time and also have the ability to select points of interest to see how they relate to each other spatially. We create three different scatter plots shown on the left most column of views in Figure 4.1, that display the position of ion cloud in each dimension over time and also provide the ability to select points in any of the plots to be highlighted in other plots. Second, what are the patterns of ion cloud motion in the spatial dimensions taken pairwise? For that we create three scatter plots that show the x-y, y-z, and x-z orthogonal projections of the cube onto its 2D faces. These views are laid out in the columns 2 and 3 of Figure 4.1. To see the details of a volume of interest within the cube, we create three more corresponding scatter plots of x-y, y-z and x-z to show the selected zoomed volume. These views appear in columns 4 and 5 in Figure 4.1. We also create a parallel coordinate plot, spanning columns 3 and 4 to see correlations between time and the individual spatial dimensions. Third, a gradient plot in column 6 maps time to color and allows filtering on time by selecting a range of color. Fourth, we coordinate time in the gradient plot with the time axis in the parallel coordinate plot to give us the ability to coordinate them on a selected range of time. Finally, we create a table view to show the raw data and select items to highlight at particular times. The table view (shown in column 6) is coordinated with the gradient plot to highlight rows in the table for times selected in the

gradient plot.

In C4D3, the code for CMVs is written in a declarative manner by supplying input data and view-specific parameters to the view modules and binding coordination objects between them. Arbitrary subsets of views can share appearances and behaviors by binding different objects that store and manage the desired appearance and behavior states of the visualization as a whole. For instance, it is possible to select a subset of views with similar or dissimilar appearances, such as scatter plots and a table view, to highlight the same data point when hovered over with a mouse in any of these views. In another scenario, the same or different subsets of views can be selected and coordinated by having selected items in one view highlighted in other views. A more complex example, shown in Figure 4.1, coordinates a range selection in the gradient view with an item highlighting filter applied in the parallel coordinate plot. Views can also be instantiated from the same view module to be coordinated through common design elements. For example, scatter plots that share the same data can yoke their individual axes to the visual extent of the data as each one displays it. Mixing and matching coordination strategies allows for versatile and open-ended ways to coordinate views, offering designers ample flexibility in achieving their desired outcomes.

4.5 View Abstraction and Coordination in C4D3

Visualization designers create and use many different kinds of interactive data views. Although the introduction of fundamentally new view types is less and less frequent, the visualization community continues to study known views, their variations, how to implement them, and how to compose them into larger structures including coordinated multiple views. There is particular interest in helping users create CMV designs themselves. Balancing expressiveness and accessibility of specification is a key challenge in this effort.

With C4D3, we approach this challenge by adopting a level of view abstraction based on the *parameter set model* of visual exploration [63]. This model treats visualization state as a

collection of parameters, some or all of which can be modified through interaction. The Live Properties architecture in the Improvise visualization system [138] builds on the parameter set model by adding a model for defining views in terms of properties (slots), binding those properties to variables (parameters), and broadcasting update notifications between views whenever any of them changes a variable interactively. Instead of coordinating views through low-level interactions (too mechanical) or high-level semantics (too abstract), Live Properties coordinates views through a middle-level of concrete, meaningful state objects. Navigation interactions translate into ranges, angles, and distances. Selection interactions are captured as an array of brushing hits indexed to data items. Inside view implementations, translating common visualization interactions into such state objects is straightforward, as is parameterizing the visual encoding of data as a function of the state objects. Coordination in Live Properties is *accessible* because the state object data types, and how they relate in a view, are easy to understand. It is *expressive* because views can be interactively coupled with each other in a myriad of ways well suited to both standard and custom coordination design.

C4D3 combines a re-implementation of the Live Property architecture in JavaScript with a wrapper for encapsulating the JavaScript code of D3 visualizations as a Live Property view type. To implement a new view type, a visualization developer makes a copy of the wrapper. They then insert code to create the properties of a view when it is instantiated. Next, they insert the D3 code fragments that perform event handling and visual encoding appropriate to that view type. Finally, they modify the fragments to translate interaction events into property changes and calculate visual encodings from property values. Designers can adapt existing C4D3 views or create view variations by adding relevant Live Properties to support coordination in the widely available rich set of D3 examples.

The view wrapper also provides the functionality needed to propagate interaction events and property updates to keep the view updated. Figure 4.2 shows the internal components of C4D3 views and how they are connected through variables for coordination. Each view has event listeners attached to the DOM corresponding to various mouse, keyboard, and touch

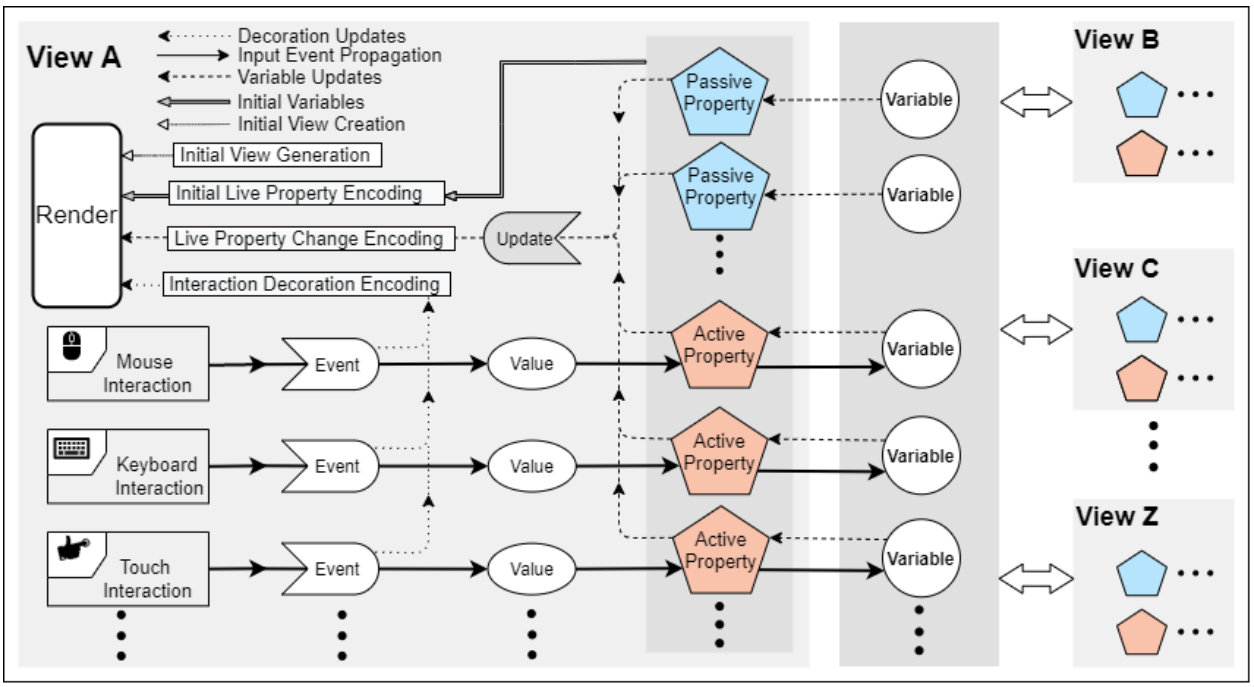


Figure 4.2: The internal structure of C4D3 views and their external coordination through variables. Interaction events propagate via properties to change variable values. Changed variables broadcast update notifications though all connected properties to trigger view updates.

events. Each property of a view registers with relevant listeners to receive notification of events. When it receives an event notification, it modifies the value of its bound variable (if it has one). The variable broadcasts the value change to all bound properties, including the originating view itself. Each property that receives a variable change notification triggers a draw event to update its parent view. When a view updates itself, it accesses current property values as needed. The originating view does not necessarily trigger a draw event directly in response to the interaction, but this can be done to draw a visual side-effect of interaction, such as to show the brushing shape itself. This approach allows clean decoupling of event handling code from property update code inside views. This approach also allows variables to be initialized (and even updated by other means) in a way that keeps all views up to date at all times, including upon first display. A property can also be tagged as active or passive to indicate whether interaction in its parent view can affect it or not.

This approach to view abstraction and coordination in C4D3 has several advantages.

First, it makes the implementation of individual views entirely independent of each other. Views communicate only indirectly through variables, and have no direct connection to or even knowledge of each other. Second, it eliminates the need to write custom code inside views to implement *the coordinations themselves*. Third, as a consequence of the previous two, it substantially increases the scalability of coordination constructions in terms of the number of views, the number of shared state objects, and the overall amount of interactive interdependence that can be implemented with little code. The specification of the structure of coordination between views is effectively externalized from the views themselves. If one can implement a view type in C4D3, then any number of instances of that view type can be created and coordinated. One can create new interactive designs in D3 and wrap each in the C4D3 wrapper in a way that effectively separates the concerns of visual representation design and coordination specification.

C4D3 provides a variety of data types to define interactive state values, such as plot ranges and item selections. Views create, interpret, and utilize property values based on their type. For instance, if a view uses a bit array property to brush and highlight selected items, the visualization designer can create a variable of that type to bind to that view's property, then bind it to other properties of any views to coordinate them through that typed value. Types helpfully communicate each view as a property-based API to designers, and constrain how property values can be expected to effect visual representation and interaction handling in different views. Views can use their property values for representation and/or interaction as they need to, including to parameterize an internal data transformation pipeline. As a result, C4D3 supports flexible coordination well beyond basic navigation and selection, through the addition of view variants that define and utilize their properties in sophisticated ways.

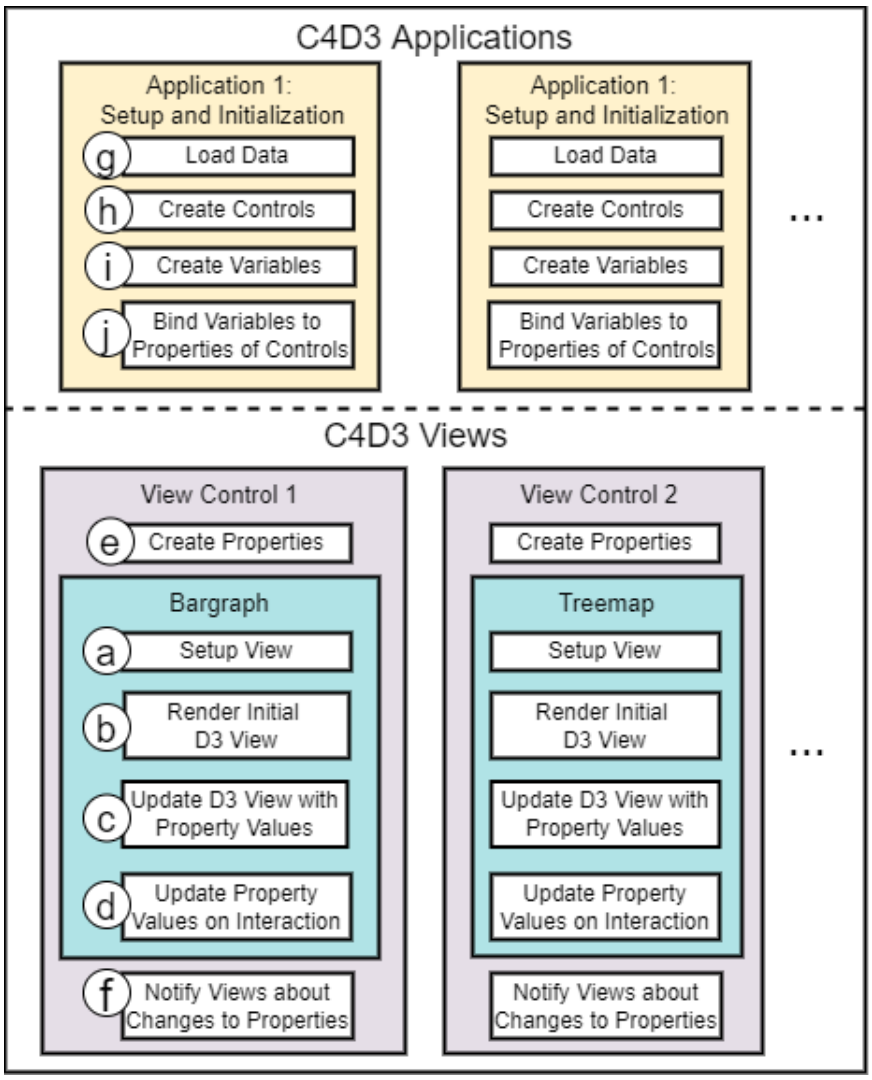


Figure 4.3: Code organization in C4D3: (a)–(d) view modules; (e)–(f) view wrappers; (g)–(j) CMV applications. See the main text for descriptions of the code for labels (a)–(j).

4.6 Implementing CMV in C4D3

C4D3 is implemented as a JavaScript library that consists of six distinct software layers to build CMVs: (1) the D3 library handles DOM specification and interaction handling of views; (2) modules built on D3 set up a view, render and update it, and update its properties upon a change in state of the view; (3) view wrappers interface view modules to the coordination library; (4) a coordination library manages coordinations between the views; (5) an API layer provides abstraction for interacting with individual view controls to support creation

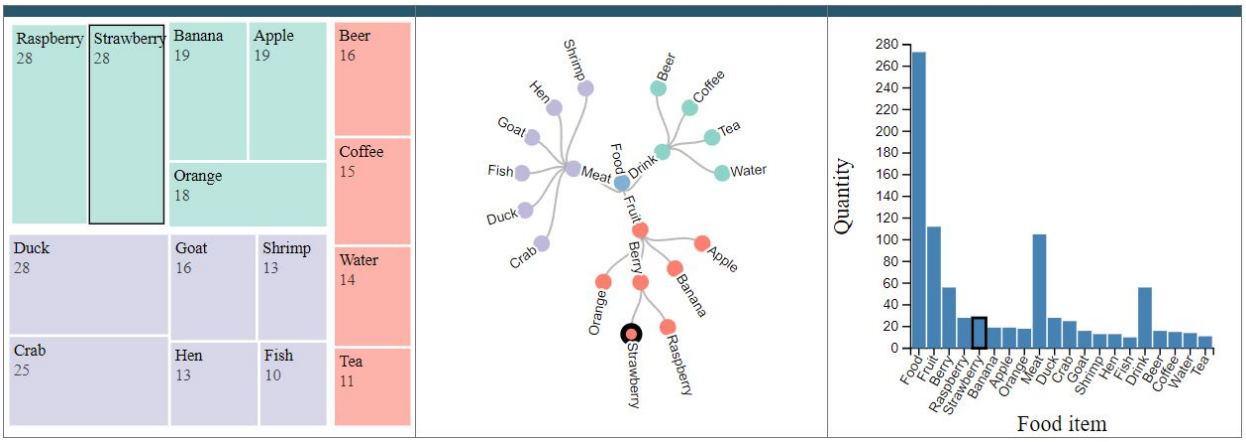


Figure 4.4: A CMV visualization with three views: a treemap, a radial tree, and a bar chart. The views are coordinated to highlight the item hovered over in any of the views.

of views and specification of coordinations; and (6) CMV applications build on the API layer. Figure 4.3 summarizes the general structure of the code one needs to write to implement an application using these layers. First, implement view modules (Figure 4.3a–d). Next, implement the corresponding view wrappers (Figure 4.3e–f). Finally, set up and initialize the application itself (Figure 4.3g–j).

Figure 4.4 shows a simple example that visualizes a taxonomy of food with three views: a treemap, a radial tree, and a bar chart. Each view has an indication property that determines which item to highlight when that item is hovered over by the mouse in any of the views. The properties are bound to a common variable to establish the coordination. In the rest of this section, we present the complete code to specify the example and explain its construction.

4.6.1 Coding View Modules

A CMV application can be built using existing view modules, including several common examples provided by the C4D3 library, and by writing any needed additional view modules and their corresponding view wrappers. View modules are responsible for setting up the initial view representation (Figure 4.3a–b), updating the views (Figure 4.3c), and translating interactions in views into property changes (Figure 4.3d). The majority of code that needs to be written when creating a new view module is the D3 code (Figure 4.3b–c).

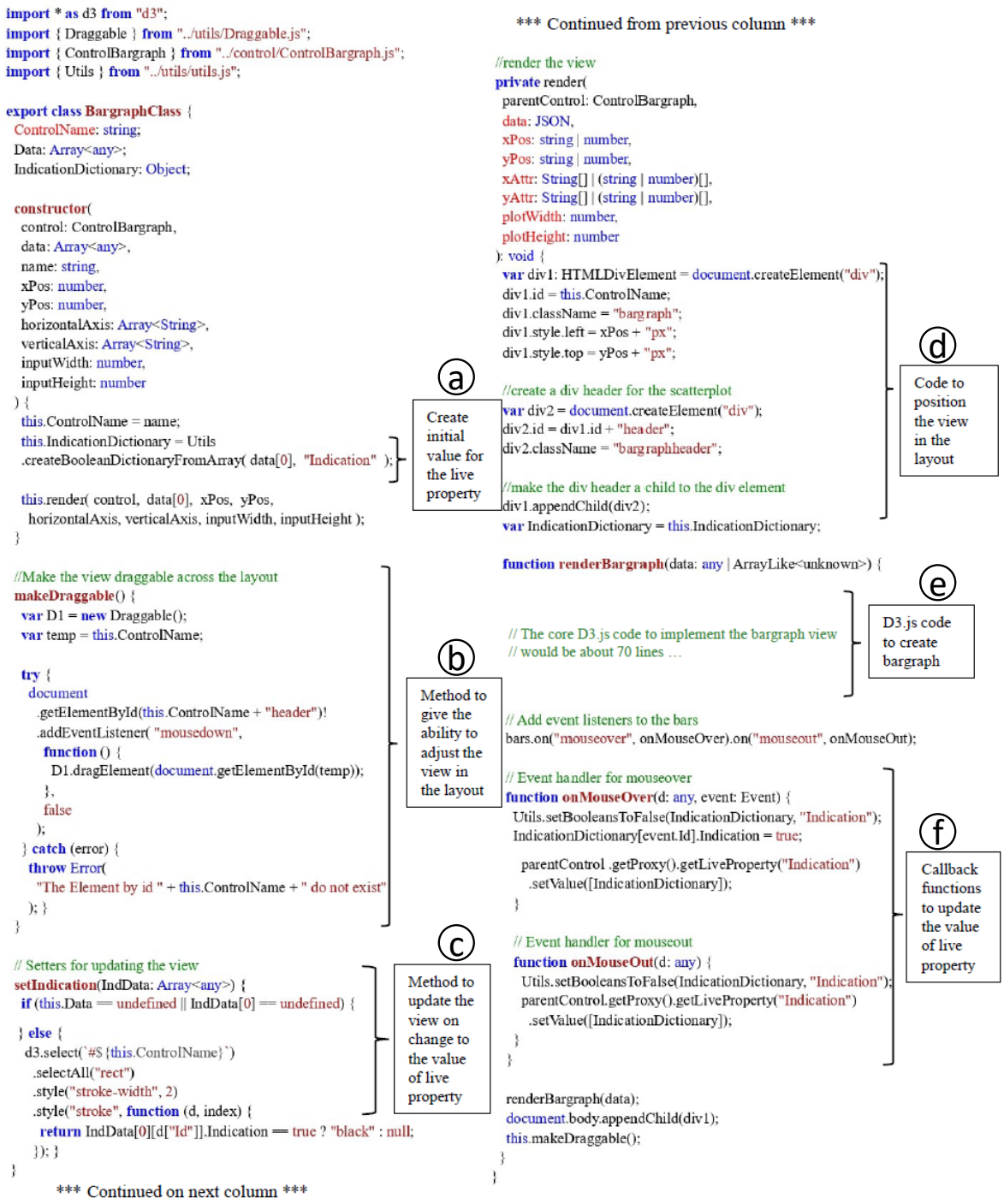


Figure 4.5: Module code for the example bar chart, showing: (a) initialization of properties, (b) a utility method to enable interactive positioning of the view, (c) a method to update the view upon property changes, (d) code to position the view in the layout, (e) D3 code to render the view, and (f) callback functions to update the property upon interaction in the view.

Figure 4.5 shows the module code for the bar chart view in Figure 4.4. The code in Figure 4.5a creates an empty boolean array, initialized with false values, and sized to match the input data, for storing the values of an *indication* property to track the mouse hover. Figure 4.5d shows the code to create the initial bar chart view using the vanilla D3 code for a bar chart shown in Figure 4.5e (hidden for brevity). The code shown in Figure 4.5f updates the value of the property when mouse interactions (onMouseOver and onMouseOut) occur, changing the property value by altering the bit values of appropriate DOM elements to true or false. The property change also notifies the view wrapper and triggers any needed updates to the view, as shown in Figure 4.5c. View modules must separate the code for each of their property types into distinct update methods. The module code for the treemap and radial tree views is structured the same way and was implemented by copying the bar chart module code then modifying the property update code as in Figure 4.5c and the D3-specific code as in Figure 4.5e.

4.6.2 Coding View Wrappers

Each view module has a corresponding view wrapper (also referred to in the API as a view control). Creating a wrapper for a new view module focuses on determining which properties a view needs to have for coordination with other views. The wrapper defines and maintains the properties that encapsulate the appearance and behavior of the view (Figure 4.3e) and acts as an interface between the view module and the coordination layer by updating the view when a change is detected in any of its properties, either by the result of interaction with the view itself or by the result of coordination with other views (Figure 4.3f).

Figure 4.6 shows the wrapper code for the example bar chart view in Figure 4.4. The code in Figures 4.6a–c creates the *indication* property (called "live_property_indication"), instantiates the view for that property, and notifies the view of incoming changes to that property from outside the view, respectively. In Figure 4.6a, the property is tagged as an active property ("true" on the third line) that can be changed by the view itself, as opposed

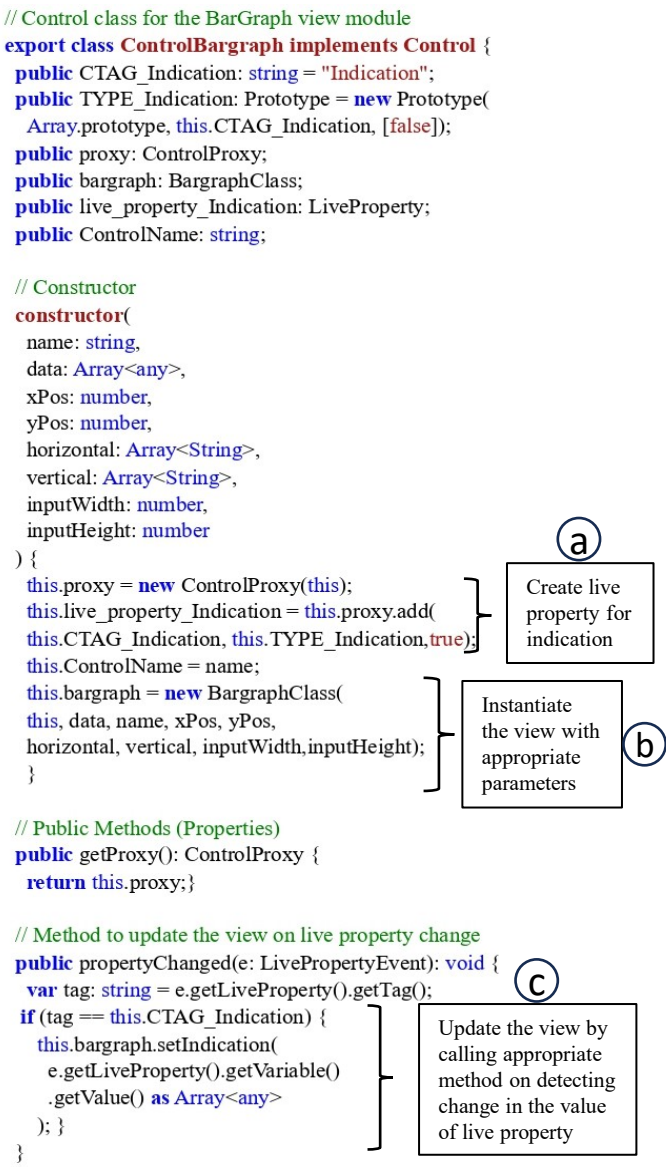


Figure 4.6: Wrapper (control) code for the example bar chart, showing: (a) initialization of properties, (b) initialization of the view, and (c) notifying the view of property changes.

to a passive property (which would be "false" on that line) to indicate that the property can change due to coordination with other views but is not affected by interaction in its own view. The wrapper code for the treemap and radial tree views is structured the same way and was implemented by copying the bar chart wrapper code then modifying the code in Figure 4.6b to instantiate each respective view with its needed parameters.

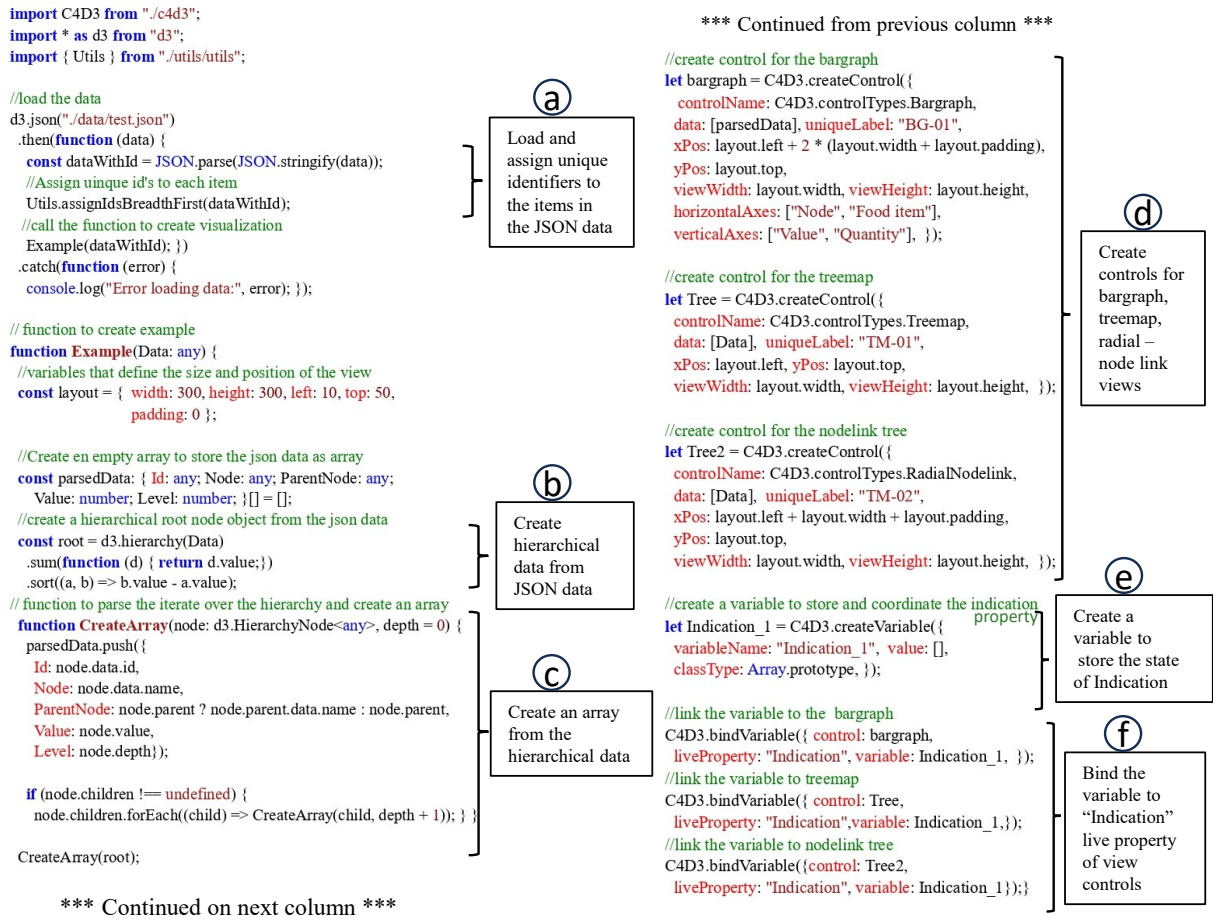


Figure 4.7: Application code for the example in Figure 4.4, showing: (a)–(c) loading and preparing data, (d) instantiating views, (e) instantiating variables for the hover coordination, (f) binding the variable to the properties of views to establish the coordination.

4.6.3 Coding CMV Applications

Creating a CMV application involves initializing the constituent views and specifying coordination using the C4D3 API layer, broken down into the four steps shown in Figure 4.3g–j. First, load the data and create parameters that define the size and position of the views, such as width, height, and position. Second, instantiate the view controls with this data and the parameters. (Alternatively, the layout of the CMV can be managed using a CSS library like Bootstrap to support a responsive grid system for arranging individual views, further simplifying the code for individual view setup.) Third, create variables to store the state of the views as property values. Fourth, bind the variables to the appropriate properties of the

view controls to specify coordination.

For the example shown in Figure 4.4, we first load and prepare the data (Figure 4.7a–c), then create layout parameters and instantiate the view wrappers for the bar chart, treemap, and radial tree views (Figure 4.7d). Next, we create a variable called *Indication* to store the coordinated interaction state of hovering shared by the views (Figure 4.7e). We use a simple array as the property type to represent the shared hover interaction state, with *True* and *False* in the array to represent whether a given data item is being hovered over. Finally, we coordinate the views by binding the *Indication* variable to the *Indication* property of all three view wrappers (Figure 4.7f).

Evolving a CMV application to include additional views and/or coordinations is a straightforward matter of instantiating more view wrappers (adding blocks to Figure 4.7d), creating more variables (adding blocks to Figure 4.7e), and/or binding variables to more view properties (adding blocks to Figure 4.7e). A coordination design can be rapidly and incrementally revised by mixing and matching variables to views through their properties. Incremental creation of new view types or modification of existing ones can be performed by copying and editing view module and wrapper code before returning to the application code to incorporate those view types and coordinate them.

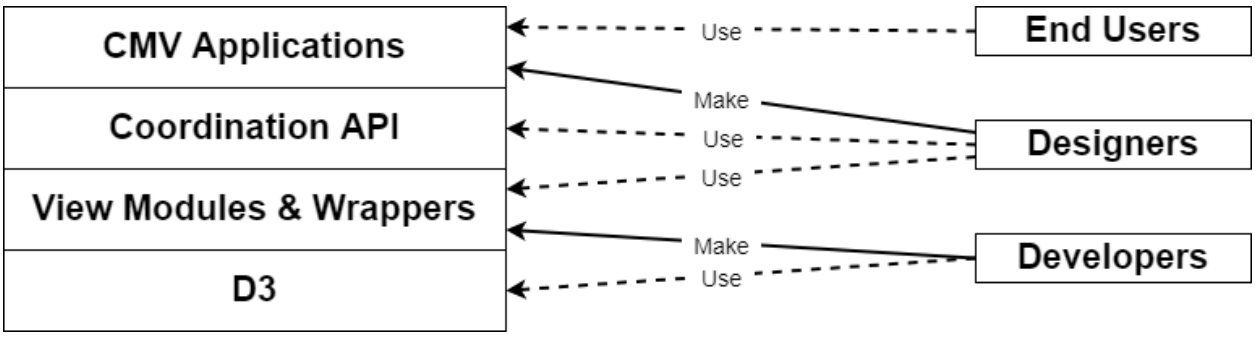


Figure 4.8: A software layer diagram of C4D3 showing how different stakeholders make and use the four different layers.

4.6.4 C4D3 in Visualization Design and Use

We designed C4D3 to support three main groups of stakeholders .

- **Visualization Developers** use the *D3* [24] library to develop *View Modules* & *Wrappers*. They can extend the existing view modules and view wrappers provided by C4D3 or create their own custom view types.

- **Visualization Designers** make use of the *View Modules* & *Wrappers* and the *Coordination API* to develop CMV applications. Their role lies in translating the the needs of end users into view and coordination choices. They can also work with visualization developers to create new view modules or modify existing view modules as needed to realize their CMV application designs.

- **Visualization End Users** are the consumers of developed CMV applications like the ones shown in Figures 4.1 and 4.9. They communicate their needs to the visualization designers who ultimately create CMV applications for them.

4.7 Coordination Patterns

With multiple views in a visualization, users can see and interact with data from multiple perspectives simultaneously. Coordination allows users to combine the interactions of different views to compose more complex queries than would be possible in any single view. To support common data exploration strategies [119], many visualizations employ basic navigation and selection coordinations [89], such as brushing [18]. Techniques like compound brushing [33] and cross-filtering [141, 142] employ coordination constructions that interplay with data processing and querying in more complex ways.

While our current view abstraction focuses on supporting basic forms of coordination, it is sufficient to realize many common coordination patterns [140], and visualization designers can vary and synthesize them to suit specific application needs. Transparency of D3 code in

views also leaves the door open for designers to implement direct view dependencies if they so desire. Here we describe a few of the general coordination patterns that were readily designed into the above C4D3 examples of ion motion (Figure 4.1) and food types (Figure 4.4).

- *Synchronized scrolling* in Figure 4.1A coordinates the horizontal range of time visible in three time series plots. Each plot has a pair of live properties to represent its visible horizontal and vertical ranges, each corresponding to a translated axis value in D3. Changes to either range are applied to the corresponding axis and the points in the plot are repositioned accordingly.

- A *scatterplot matrix* (SPLOM) shown in Figure 4.1D is a more complex construction of range bindings. Three data dimensions are laid out in a traditional stair-step arrangement of plots. X, Y, and Z range variables are bound to the range properties of the plots so as to create the expected synchronization of scrolling across all three views.

- A *perceptual slider* shown in Figure 4.1F coordinates a color slider and a parallel coordinate plot. The two views share a range variable for a selected range of color in a color gradient. The two views also share a color gradient for sake of visual consistency. In this case, we implemented both as general-purpose views but tweaked their respective D3 code to use the same gradient by coincidence.

- *Shared selection* shown in Figure 4.1 supports brushing between the three time series plots and the table view. The three plots in the SPLOM are similarly coordinated, but via a second selection variable.

- *Shared indication* shown in Figure 4.4 similarly coordinates hover interactions across several views, in this case of a food item. As in shared selection, data items hovered over in any of the coordinated views is highlighted in all views.

4.8 Example: COVID-19 Visualization

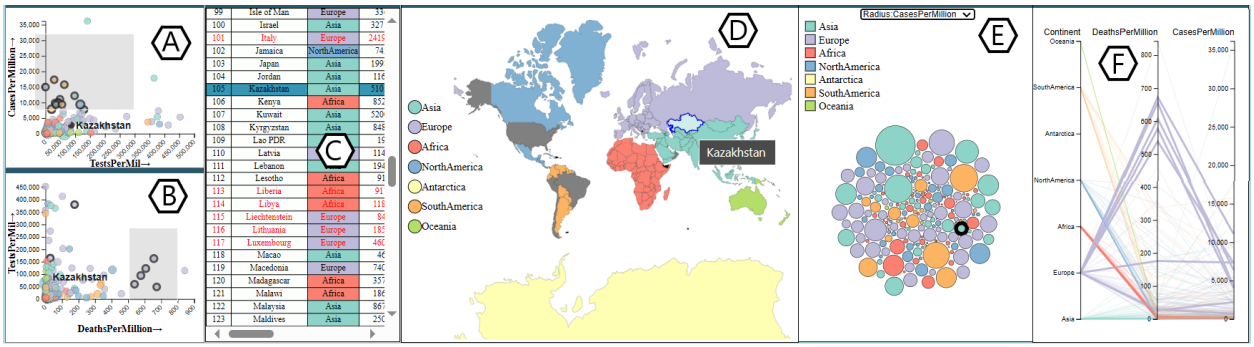


Figure 4.9: Visualization of COVID-19 data [148] with six views (A–F): two scatterplots, a table view, a choropleth map, a bubble chart, and a parallel coordinate plot. Views encode different attributes of countries using a common color scheme for continents. Coordination between views includes range navigation (AB), selection (AD, BCF), and indication (ABCDE).

Figure 4.9 shows a novel visualization with six different coordinated views of COVID-19 data built using C4D3. The data consists of case and fatality statistics on July 8th, 2020 from the Worldometer website [148]. This section describes in detail the implementation of the views, interactions, and coordinations of the visualization and assesses how C4D3 supported its development.

The visualization consists of 6 views of five different types.

- **Scatter Plot:** The visualization includes two instances of a scatter plot module. The top scatter plot (Figure 4.9A) displays the tests per million variable (horizontal axis) versus the cases per million variable (vertical axis) while the bottom scatter plot (Figure 4.9B) shows the deaths per million variable (horizontal axis) versus tests per million variable (vertical axis). Both scatter plots support selection of items by brushing a rectangular region. Items within the brushed regions of the recently interacted scatter plot are selected and highlighted with a black stroke.

- **Table View:** Figure 4.9C shows a table view of the entire dataset. Continents are color-coded based on the categorical colors assigned to each continent, matching the

legend in the choropleth map and the bubble chart. Selection in the table view is performed by double-clicking a row to select a single element. Clicking on a row and dragging the mouse slowly upward or downward selects multiple items. If the items are previously selected, they will become un-selected and vice-versa. Selected rows are displayed with red text. Hovering over a row (indication) changes its background to a teal color.

- **Choropleth Map:** Figure 4.9D shows a choropleth map of the world using a Mercator projection. A legend shows how countries are color coded by continent. The map supports selection and indication interactions. Double-clicking a country selects it and highlights in gray. Hovering over a country indicates it with a blue border and reduced opacity.

- **Bubble Chart:** Figure 4.9E shows a bubble chart with a selection menu. A legend shows how countries are color coded by continent. The bubbles represent countries, and their radius is drawn corresponding to the selected menu option: cases per million population, tests reported per million population, or deaths per million of population in each country. Double-clicking on a bubble selects it and highlights it in gray. Hovering over a bubble highlights it with a white border and reduced opacity.

- **Parallel Coordinate Plot:** Figure 4.9F shows a parallel coordinate plot of countries by continent, cases per million, and deaths per million of population. Each line representing a country is colored according to the categorical color assigned to its continent. This view does not allow direct selection or hovering. The plot highlights items selected via its coordinated views, but does not support selection or indication itself (as an example of using passive properties for one-way selection).

The visualization coordinates the six views in several ways, creating coordination patterns.

- The two scatter plots are coordinated through a range variable. The horizontal range of the top scatter plot is synchronized with the vertical range of the bottom scatter plot. This coordination demonstrates *synchronized scrolling*.

- The top scatter plot (Figure 4.9A) and the choropleth map (Figure 4.9D) are coordinated on their item selections. Any selection made in either of the views is reflected in the other view, demonstrating a *shared selection* coordination pattern.

- The bottom scatter plot (Figure 4.9B), table view (Figure 4.9C), and parallel coordinate plot (Figure 4.9F) are coordinated on their item selections. Any selection made in either the table view or the scatter plot is reflected in all three views, demonstrating a second, independent *shared selection* coordination pattern.

- The Scatter plots, table view, choropleth map, and bubble chart are coordinated by indication. Any item indicated by hovering the mouse over it in any of the views is highlighted in all of the views. This coordination is a multi-way *shared indication* coordination pattern.

The development of the CMV example shown in Figure 4.9 benefits from the capabilities of both D3 and C4D3. D3 [24] supports the development of six views in a transparent manner by exposing both the visual encoding specification and the interaction mechanism. Each view *transparently* associates properties with their roles in visual encoding and interaction. The visualization exploits C4D3's *extensibility* by first developing a view module and wrapper for the scatter plot with range, selection and indication properties, and then extending the same view wrapper by removing the range property for the remaining views. C4D3's *modularity* aided in view development by starting with scatter plot views, then incrementally adding more view types and integrating them into the coordination patterns. While the designs of the view types in the example vary widely by extracting the similar behaviors of *selection* and *hovering* across the views into sensible value types, the small number of properties could be coordinated across them to achieve high *composability*. All views respond to selection,

indication, and hovering interactions through live property updates rather than local view updates, ensuring full *consistency* in all coordinations of the interactions.

4.9 Discussion

We developed a specialized coordination library that integrates well with the popular data and visual representation capabilities of D3, enabling flexible combination of views and coordination to build simple or complex CMV visualizations. In the following section, we discuss the design choices we made for C4D3 and assess achievement of development objectives.

4.9.1 Modular View Support

Building CMVs using C4D3 can take advantage of substantial code reuse. A large fraction of the code in view wrappers is shared between views with a small portion needed to add new properties for coordination. Implementation of new views nevertheless requires structuring D3 code in a different, modular way to support coordination, as discussed in Section 4.6.1.

The complexity of code in a view depends mainly on how expressive we want the view to be to present itself as a collection of properties available for coordination with other views. New C4D3 designers should start with CMV examples that support simple coordination patterns, such as shared selection and shared navigation, and incrementally develop support for more complex coordination patterns.

Despite its name and concentration on D3, the implementation of C4D3 does not inherently restrict designers from substituting an alternative charting library for D3. However, the alternatives would likely vary widely in how they can be used to manipulate the DOM to express interaction state and visual appearance as a function of coordinated property values.

CMV view initialization in C4D3 is abstracted from the low-level details of view creation, giving designers the ability to build complex CMVs in a declarative manner without getting embroiled by the low-level implementation details of views. The separation of concerns and

abstraction offered by properties and shared interaction state appears to provide a strong correspondence between what users want for visual data querying and how designers can specify suitable view inter-dependencies in CMV constructions.

A major challenge in developing support for complex coordination on top of visualization libraries like D3 is their limited range of interactions available to modify DOM elements (brush, drag, zoom, pan, lasso) and the ways they can conflict. This limitation makes it much harder to implement multiple interactions in D3 to support multiple properties in a C4D3 view, either by supporting a coordination pattern by spreading out needed interactions across extra views, or forcing developers to write custom code to process low-level mouse and keyboard events to sufficiently distinguish interaction forms and map them into particular facets of coordinated interaction state.

4.9.2 Modular View Update Mechanism

In general, D3 views are constructed such that mouse events trigger callbacks that update the DOM, making code look clean and simple. This is true for simple and direct interactions. However, when changes are complex and indirectly triggered by coordinated views, updating in this manner makes the code monolithic and adds complexity to the already lengthy code for some D3 views. To manage this complexity, C4D3 decouples the code responsible for re-rendering views into separate callback functions, simplifying the D3 code and making it more modular. This delegates the responsibility of mouse events to only update properties, allowing each view to interpret changes to properties and modify themselves accordingly.

This approach also allows views to support coordinations that span multiple properties in an easily understandable manner. For example, if a view receives shared selection state in its properties from multiple views via multiple variables, it can adjust its appearance based on the combined values of any or all variables. Achieving this level of coordination without C4D3 would be highly complex in D3 alone.

C4D3 can also support sharing data as a coordination parameter, allowing it to be loaded

once instead of multiple times across views. This approach not only improves data handling, but also enables the development of views that support data annotations through interactions and share the same modified data across views, ensuring consistency of annotated data across coordinated views.

4.9.3 Assessment of Objectives

Our first objective is to preserve **transparency** of D3 code in view implementations. Parameterizing views via minimal sets of properties needed to share coordination state provides accessibility. Giving each view responsibility over how to use property values to determine appearance and behavior provides expressiveness. A visualization designer can choose a visual encoding appropriate to the data and application, and implement interaction as needed to accomplish many common visual querying tasks. Item selection in a view can involve clicks, rubber bands, lassos, or other selection gestures. Indication by drawing or hovering is similarly flexible. In both cases the view simply performs the usual hit tests on data to populate a bit array to share via a selection variable.

Specifying views without connecting them directly offers both **extensibility** and **modularity** in view development. D3 practitioners can take advantage of the view abstraction to create new view types, view variants, and coordinate them incrementally as needed to realize their designs. Design variations can be explored by quickly changing the set of available variables and their bindings to views being considered, often with little or no modification to the D3 code itself. Doing so depends on how well the aspects of appearance and behavior needed for variation are exposed as properties in available view types. Overall, C4D3 provides a sharp separation of concerns for view design. Starting from available view types, new view types can readily exploit redundancy in C4D3 code, and often in D3 code as well. The view types in the examples all share code that is essentially the same except for D3 fragments that handle visual encoding specifics. Event handling code, such as for panning, can be reused in the same way as in the world of D3 examples. View variants can offer custom combinations

of interactions for use in special design cases. Curating collections of view types is likely to be an integral activity of C4D3 designers, much like in the D3 community.

View designers specify how live property values affect a view's appearance and behavior, and how interactions in a view affect live property values. The choice of live properties to design into a view type is flexible, yet often calls for only a small set of quite obvious and sensible value types. Live properties tend to be compatible (and often identical) across views, even quite different ones. Consequently, the **composability** of views is high.

When an interaction occurs in a view, it could update itself by responding directly to the interaction, indirectly to a live property notification, or both. In C4D3, views translate most interactions into one or more live property changes. Updates happen indirectly via propagation of variable changes, including in the view in which an interaction originated. By updating themselves independent of where interactive modifications originate, views effectively track the current coordination state of the visualization, and thus maintain visual **consistency** within themselves and between each other. Asynchronous propagation of events in C4D3 makes visual consistency between views *eventual*, but on current systems updates appear immediate for data sizes and view counts like in the examples.

The main objective of C4D3 is **consonance** in CMV design; that is, to support harmonious mixing and matching of multiple coordinations between diverse views. The three examples represent different C4D3 design cases. The first (Figure 4.1) recreates an existing CMV visualization with many, but basic views. The second (Figure 4.4) and third (Figure 4.4) creates new CMV visualizations with fewer, but mixed, views. Together, they show off a variety of navigation and selection coordination patterns. To build a C4D3 visualization, the designer creates variables to model interactive state, creates views to access and modify that state, then binds variables to views to coordinate them. These steps are simple, flexible, and easy to write as C4D3 code. The number of variables and views, and more importantly the number of bindings, is typically quite small—tens of each—even in complex CMV constructions like the ions example. Even so, there is still much practical work to be

done, particularly to expand beyond the small set of essential view and property value types currently included in the library. It also remains unclear how to adapt certain D3 event handling, notably for drag interactions, to the properties model.

Overall, C4D3 lets designers focus on expressing the state, behavior, and appearance of views as shareable parameters at an abstraction level that provides a closeness of mapping between desired view interdependencies and the code needed to express them, leaving most of the complexity of handling coordination proper to the library itself.

4.10 Summary

We present a working prototype of a view-level abstraction for coordinating multiple views. We adapted the Live Properties architecture as a thin coordination layer for use with the D3 web-based visualization library, and applied it to create examples that demonstrate its effectiveness to create complex CMV visualizations. Thinking about views as collections of abstract shareable properties, rather than directly shareable values, may encourage general thinking of coordination on a more abstract level. This middle-level abstraction appears to allow designers to think about coordination at a higher level of abstraction like with Nebula [34], while keeping the flexibility benefits of a lower level of mechanism [65]. Moving forward, we will explore migration of the view abstraction to serve as a CMV abstraction layer for other visualization libraries such as Vega-Lite [108] and *Plotly.js*. More broadly, we aim to support access to the full visualization data processing pipeline, starting with dynamic queries [7] and extending C4D3 capabilities to include a functional reactive language for visualization pipelines like in Improvise [138]. Another possibility is to extend C4D3 into a client-server architecture for desktop, mobile, and web clients using the coordination architecture as the basis for remote interaction including collaborative interaction via coordination [139].

Chapter 5

Software Architecture and Implementation of Hieros

5.1 Overview

Tree visualization systems [37, 96, 71] and toolkits [42] benefit from utilizing the visualization pipeline [29, 111] to support operations on trees at various stages. Visualization pipelines provide a structured framework for understanding how raw data is transformed into interactive visual representations that support human cognition and decision-making. However, the underlying pipelines in most tree visualization systems rely heavily on node and edge information, and primarily expose operations on the attributes associated with these elements. While some systems offer limited support for structural and attribute-based operations on topologies such as paths, levels, and branches, they do not treat these structures as first-class entities. Furthermore, the architectures and data pipelines of these systems are often not described in detail, making the underlying implementation ambiguous in many cases. A visualization pipeline that treats composite topologies as first-class objects, along with nodes and edges, would make it more straightforward to implement operations on them at different stages of the pipeline.

In this chapter, we present Hieros, a software architecture and implemented library that supports the representation of topologies and interaction with them as first-class objects in tree visualizations. Development of Hieros represents a punctuation point in our work to develop a tree visualization system to test our hypothesis: Having the ability to represent composite topologies in trees in a first-class manner improves the utility of tree visualizations by providing ways to represent information associated with them and also support various operations on them in a more straightforward manner.

Hieros extends the information visualization pipeline to include composite topologies mapped from the nodes and edges and supports representing aspects of attribute and structural information about those topologies. The Hieros architecture also supports operations on topologies along the pipeline to interactively create, select, navigate, and manipulate the topologies. We make the following contributions from our research on developing the Hieros library: a tree visualization library that enables the creation of tree visualizations and supports interactive operations on them; a pipeline for hierarchical data processing that treats tree topologies as first-class objects and provides support for designing operations to be applied along the pipeline; and seven variations of a representative sample of existing tree visualization techniques—linear node-link tree, radial node-link tree, squarified treemap, circular treemap, sunburst chart, icicle plot, rings that can represent various topologies to differing extents, as well as support interaction with them in different ways. This work confronts critical challenges in order to test our hypothesis. We pursue four main research goals (R1–R4) to address them.

- **R1 - Hierarchical Data Pipeline:** Understand how to design a visualization pipeline that supports the creation, layout, and visual encoding of composite topologies in tree visualizations, as well as support operations on these topologies at suitable stages of the pipeline. This includes designing the software architecture and developing its components to support the pipeline.

- **R2 - Designing Operations on Tree Topologies:** Understand the various kinds

of operations that can be performed on tree visualizations to support tasks at different stages of the pipeline.

- **R3 - Designing Tree Visualizations:** Design visualizations to represent nodes, edges, and composite topologies along with their associated structural and attribute information, including to develop layout algorithms that incorporate composite topology representations. This also includes assessing support for composite topologies across a representative set of seven visualizations.

- **R4 - Designing Interactions:** Design interactions on topologies to demonstrate support for various operations. This includes designing the visual encoding of topologies to reflect both the interactive state of the visualization and the progress of multi-step interactions.

By achieving the above goals, the Hieros library enables designers to flexibly represent various types of tree topologies from hierarchical domains and provides interactive support for performing operations on these topologies, ultimately facilitating analytical and exploratory tasks on tree visualizations.

We make the following contributions from our research by developing the Hieros library:

- A **software architecture** that supports the representation of and interaction with composite topologies as first-class objects in tree visualizations, along with a visualization pipeline that enables operations at various stages to interactively create, select, navigate, and manipulate topologies.

- An **implemented library called Hieros** that supports the creation of topology-augmented tree visualizations.

- **Seven variations of existing tree visualization designs** linear node-link view, radial node-link view, squarified treemap, circular treemap, sunburst chart, icicle plot, rings—that represent topologies and support interactive operations on them.

In this chapter, we begin with an illustrative example of a tree visualization developed using Hieros to demonstrate various composite topologies. We then describe the overall architecture and the visualization pipeline of Hieros. Next, we outline key components of the software architecture, followed by a discussion of the design of several tree visualization techniques, the interactive operations they support, and several key ways of coordinating tree visualizations on topologies. Finally, we discuss design and implementation considerations involved in creating Hieros and the implemented tree visualizations.

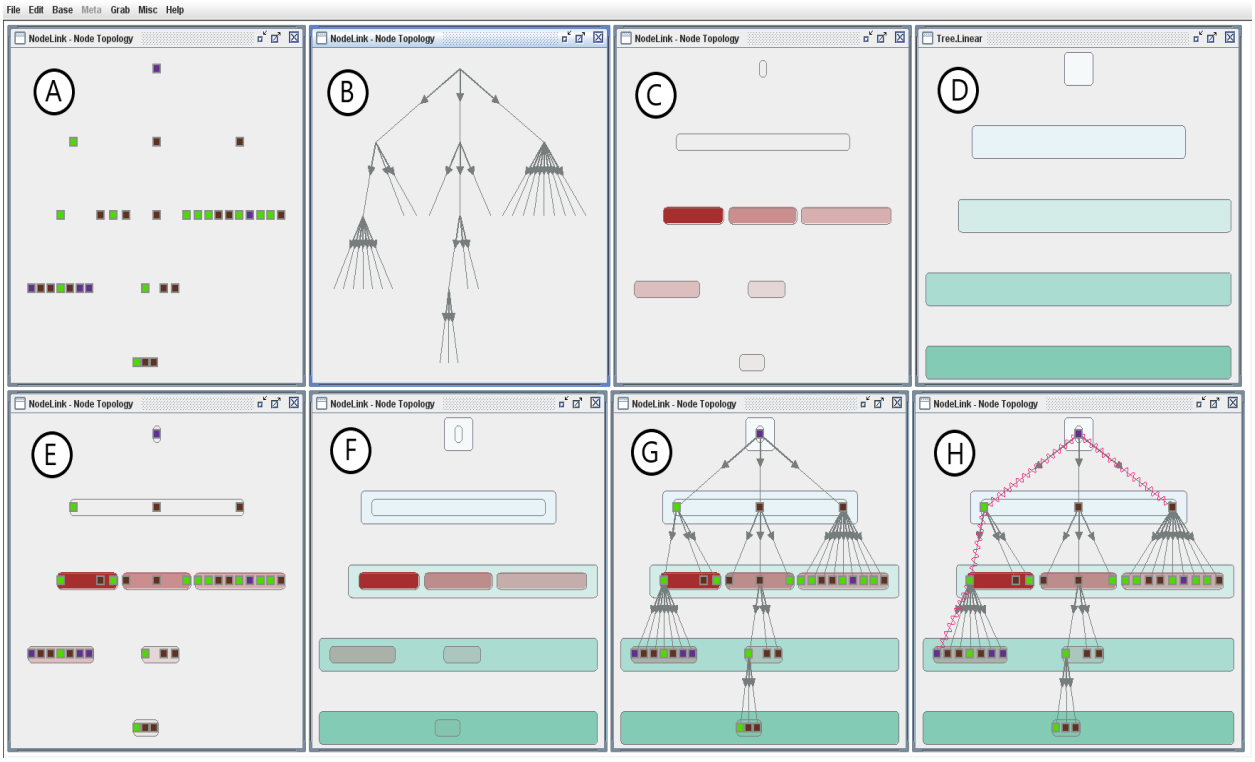


Figure 5.1: An example of a node-link visualization capable of showing different combinations of topologies explicitly. Clockwise from top left. (A) nodes; (B) edges; (C) siblings; (D) levels; (E) nodes and siblings; (F) siblings and levels; (G) nodes, edges, siblings, levels; (H) nodes, edges, siblings, levels, paths.

5.2 Hieros Visualization Example

Figure 5.1 illustrates a visualization of a hierarchy, composed of eight views that explicitly represent different combinations of topologies: nodes, edges, siblings, levels, and paths. The layout algorithm for the node-link view is modified so that node positions are first calculated and then used as anchor points to determine the positions of other topologies within the visualization. Views then make use of the visual encoding specification of the topologies to map the structural and attribute information of the topologies to visual encoding channels such as color, shape, etc. This example is only used to illustrate how common topologies can be drawn as first-class objects independent of the others. Even so, some topologies are only meaningful when shown in combination with others. For instance, nodes are the basis for all other topologies in the visualization; it is difficult to interpret the other topologies without visible nodes for reference.

Figure 5.1A displays the nodes as rectangles. Figure 5.1B uses the positional information of the nodes to render the edges. Figure 5.1C draws siblings topology using node positions as anchor points. Figure 5.1D illustrates level topologies within the hierarchy. Figures 5.1E–G depict various combinations of nodes, edges, siblings, and levels. Finally, Figure 5.1H incorporates all four of those topologies, along with a path topology, which represents a sequence of edges connecting any two nodes.

In this example, users can interact directly with the visual representations of nodes, edges, siblings, and levels. For path and branch topologies, users use modifier keys and interact with the underlying nodes to show them. By treating topologies as first-class objects in our node-link tree, both structural and attribute information can be encoded using their respective visual channels. Categorical node data is represented by node color. The mean of a numerical attribute of a siblings group determines the color of the siblings topology. The directionality of edges, i.e., from parent to child, is encoded with arrowheads. The level number is represented by color in the rectangle for each level topology. The path topology is visually distinguished using a squiggly line along the edges it traces from end to end.

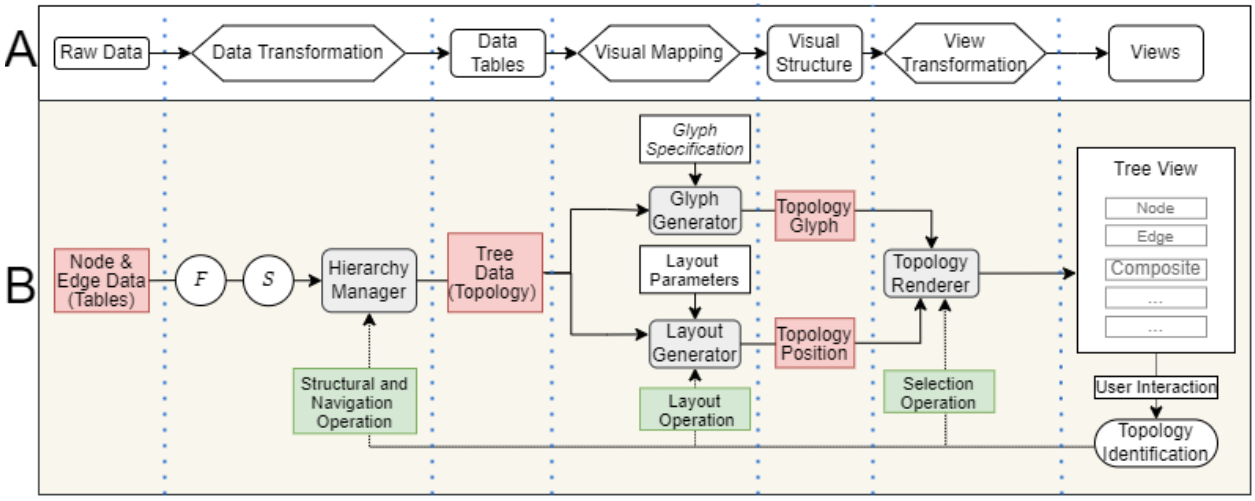


Figure 5.2: Mapping of various stages between the (A) information visualization pipeline as described by Card, et al. [29] and (B) visualization pipeline of Hieros. Backward paths represent how interactions on the topologies affect operations in the Hieros pipeline.

5.3 Software Architecture of Hieros

Figure 5.2A illustrates the information visualization pipeline [29], showing the stages involved in transforming raw data into visualizations. The three stages are *Data Transformation*, which converts raw data into data tables; *Visual Mapping*, which transforms data tables into visual structures; and *View Transformation*, which renders the visual structures to the screen as view. Variations of the pipeline [111] have been created for tree visualizations to account for *Node* and *Edge* topologies. We extended the pipeline [111] to support representation of composite topologies and operations on them.

Figure 5.2B shows the visualization pipeline of *Hieros* showing the corresponding stages involved in transforming raw tabular data into tree visualizations in Hieros. The three stages are *Data Transformation*, which converts raw node and edge tabular data into tree-structured data. The tree-structured data consists of a hierarchical data structure with additional data structures for the composite topologies; *Visual Mapping*, which transforms the tree data into *Topology Glyphs* and generates *Topology Positions* for each topology; and *View Transformation*, which renders the *Topology Glyphs* at their respective *Topology Positions*. Interactions performed on the topologies in a view trigger interactive operations at prior

stages of the pipeline, enabling manipulation of hierarchical data through the creation and transformation of topologies, layout modifications, and selection/highlighting of the rendered topologies.

The overall architecture of the *Hieros* library is shown in Figure 5.3. *Hieros* consists of modular components designed to support the representation of tree topologies and operations on them. The library is built within the *Improvise* visualization environment [138]. It currently leverages the *Improvise* user interface for loading *Node and Edge Data*, specifying visual encodings of topologies, configuring layout parameters, and providing a coordination model for constructing coordinated multiple views. The architecture includes various components aligned with the stages of the information visualization pipeline. The following section describes the key components of the *Hieros* library in detail, with reference to Figures 5.2 and 5.3.

5.3.1 Key Components of the Architecture

Hierarchy Manager

The *Node and Edge* tables are initially loaded using the *Improvise* visualization environment and serve as the entry point to the visualization pipeline of *Hieros*. In the next stage, *Data Transformation*, filter *(f)* and sort *(s)* operations are applied to the *Node and Edge* tables to exclude or reorder nodes based on attribute information. The transformed data is then passed to the *Hierarchy Manager* module, as shown in Figures 5.2 and 5.3. This module validates the input and removes any nodes not connected to the root of the hierarchy. It then converts the transformed node and edge tables into a tree data structure, generating additional data structures for the necessary topologies for downstream visualization and interaction.

The *Hierarchy Manager* module also provides functionality to support *structural* and *navigation* operations on the tree data structure, such as reordering nodes within a sibling topology, creating topologies (e.g., paths and branches), calculating derived attributes (e.g.,

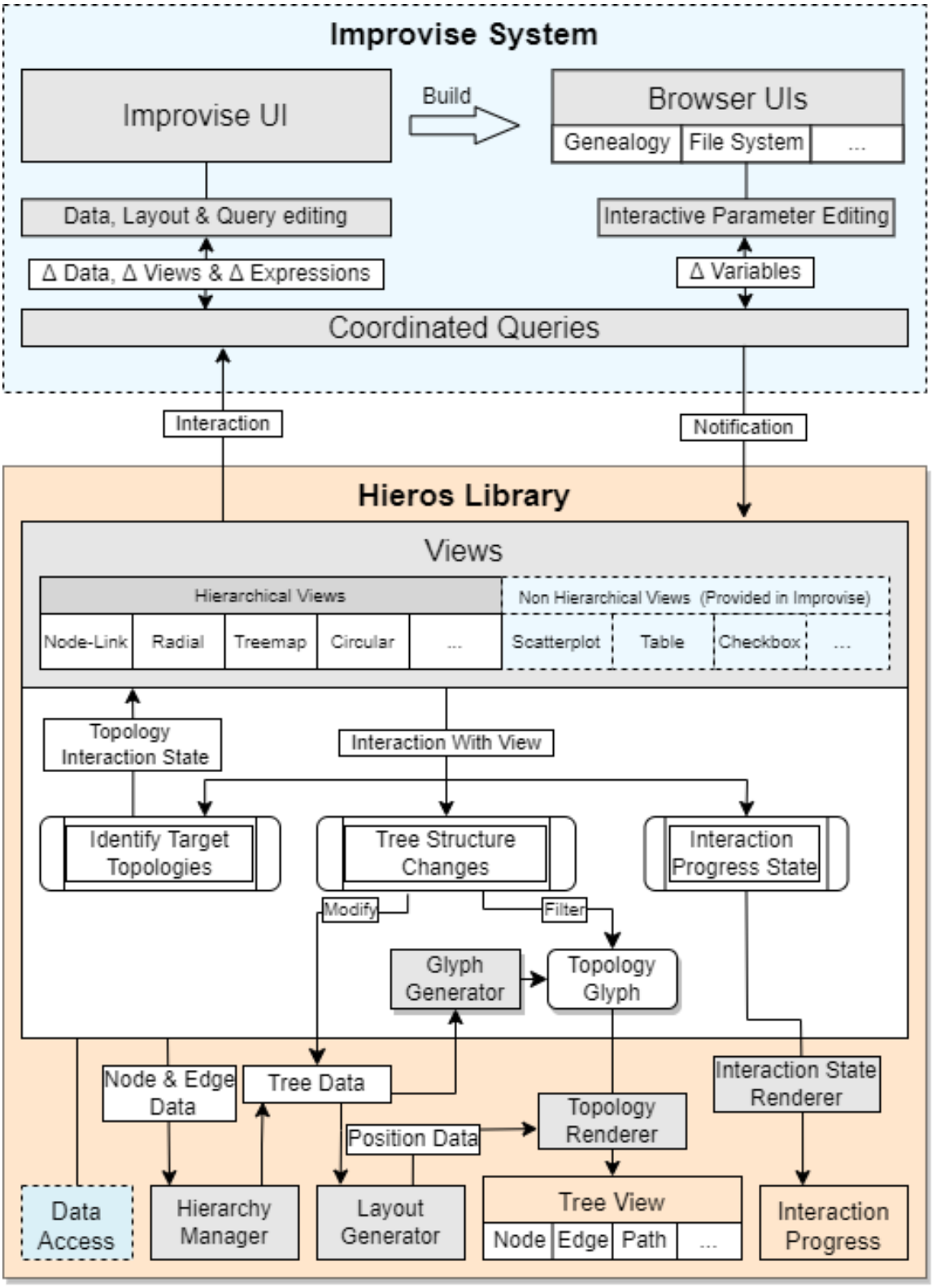


Figure 5.3: Architecture of Hieros

computing the average value of node attributes within a topology), and computing structural attributes (e.g., determining the length of a path or the number of nodes in a level).

5.3.2 Layout Generator

The *Layout Generator* module is part of the *Visual Mapping* stage of the visualization pipeline and is responsible for generating the positional information used to display topologies in views on the screen. It includes a collection of layout algorithms for various tree visualization techniques. The tree data structure generated by the *Hierarchy Module*, along with any user-specified *Layout Parameters*, serves as input to the layout algorithms of the *Layout Generator* module. The layout algorithm computes the screen coordinates to determine the positions of base and composite topologies, i.e., the spatial location at which each topology should be rendered in views. The layout algorithm used in a given view can be selected from a set of available alternatives. For example, the *Squarified Layout* of a rectangular treemap can be replaced with a *Slice-and-Dice Layout*, offering flexibility to choose among different layout variations to support diverse analytical tasks.

The *Layout Generator* also supports interactive *Layout Operations* that modify the parameters supplied to the layout algorithm. Examples of layout operations include rotating a radial tree, adjusting margins, rotating labels, and applying zoom and pan transformations.

5.3.3 Glyph Generator

The *Glyph Generator* module is also part of the *Visual Mapping* stage of the visualization pipeline. It is responsible for creating the *Visual Structures*, also referred to as *Glyphs*, for the various topologies displayed, using information about these topologies from the *Tree Data* and their corresponding visual encoding specifications (*Glyph Specification*). The *Glyph Specification* is used to map the visual channels of a glyph from a specific topology and its structural and attribute information. Visual transformations such as filters can be integrated into the glyph specification to conditionally apply visual encoding based on attribute

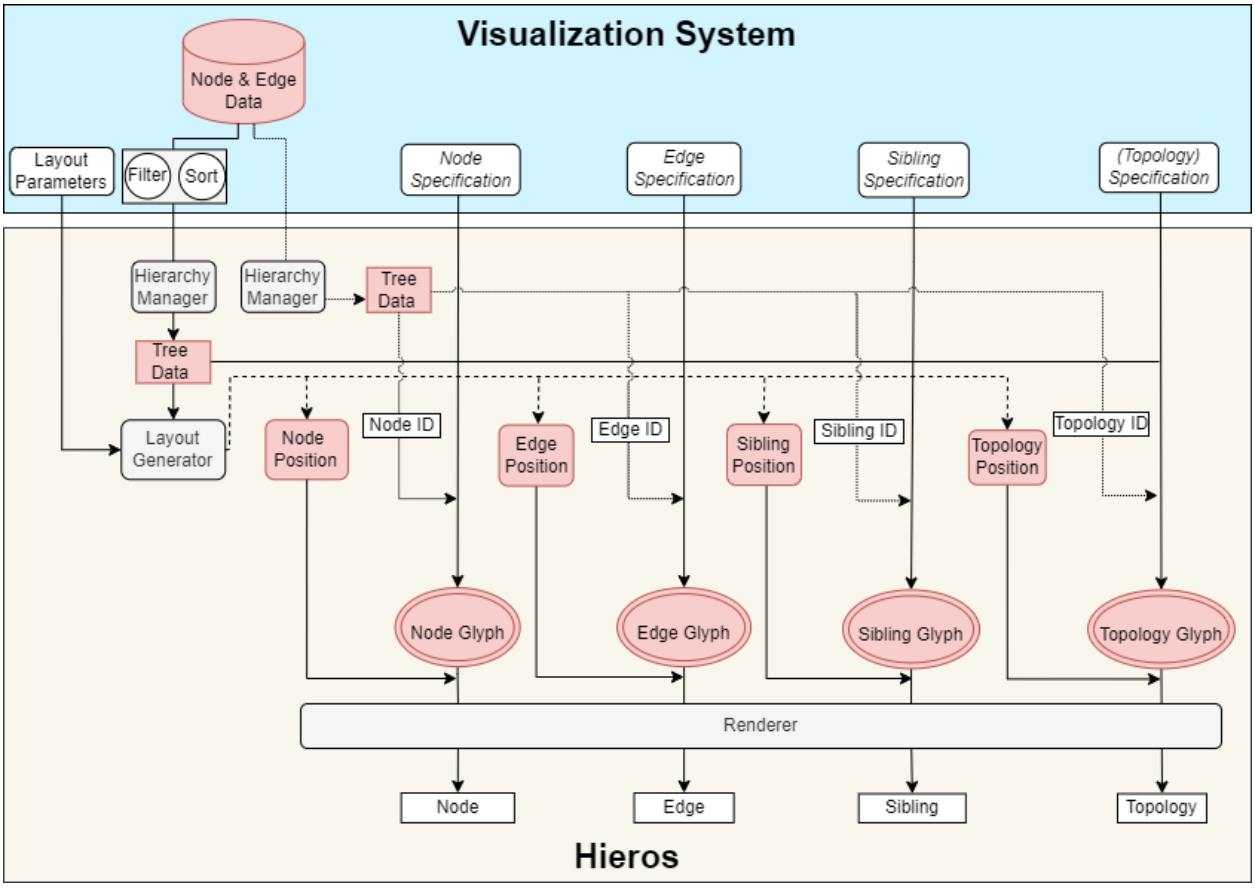


Figure 5.4: Internal structure of the *Glyph Generator* module in Hieros.

or structural properties of the topologies. To facilitate this mapping, the *Glyph Generator* maintains a schema that defines the possible set of attributes and their values for each topology-specific glyph. For instance, to map the color of a siblings glyph from the number of nodes it contains, the schema for the siblings topology includes a corresponding integer column representing the node count, which the glyph specification then uses as the source for the color encoding. The *Glyph Generator* generates a table with this schema and populates the relevant columns using information obtained from the *Hierarchy Manager*. Figure 5.4 shows how the glyphs for node, edge, and siblings topologies are generated using their specifications and the structural and attribute information about them from the tree data.

A current limitation of the implementation is a lack of designer-level support for specifying operators on structural and attribute data for use in custom visualizations. These

operations are currently implemented at the Hieros developer level and are made available for designers to use. Extending the pipeline to allow designers to define such operators directly would significantly improve the immediacy and flexibility of specifying the visual encodings of topologies based on arbitrarily derivations of their structures and attributes.

5.3.4 Topology Renderer

The *Topology Renderer* module is part of the *Visual Transformation* stage of the visualization pipeline. It is responsible for rendering the *Visual Structures* of topologies into views on the screen to generate the overall visualization. It uses the *Topology Position* information to determine the exact coordinates at which each topology should be rendered.

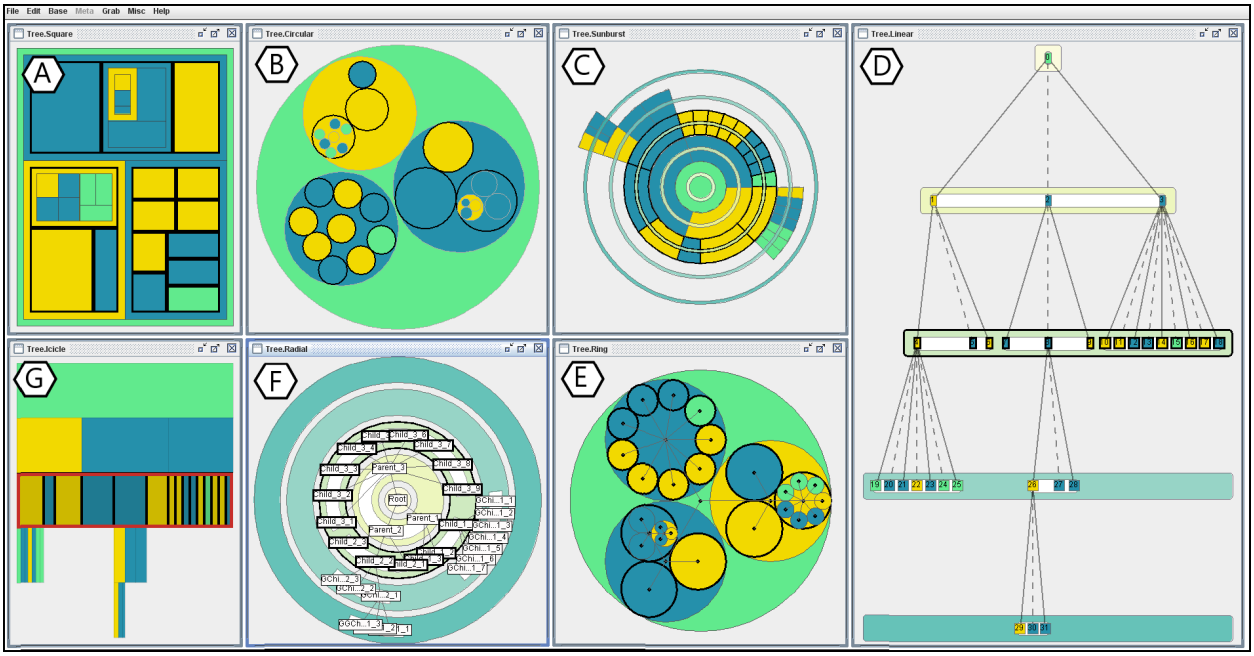


Figure 5.5: Augmented versions of common tree visualizations provided by the Hieros library. Clockwise from top left (A) Squarified Treemap, (B) Circular Treemap, (C) Sunburst view, (D) Linear node-link view, (E) Rings Layout, (F) Radial node-link, and (G) Icicle Plot.

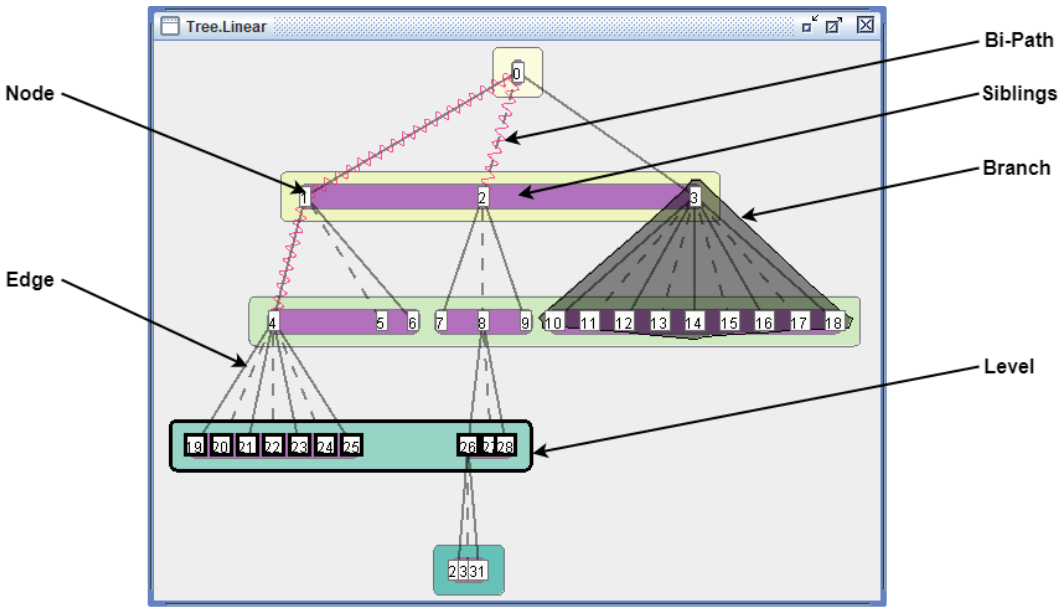


Figure 5.6: The Linear Node-Link View explicitly represents node, edge, sibling, level, bi-path, and branch topologies.

5.3.5 View Designs

The Hieros library provides a representative set of modified versions of seven commonly used tree visualization designs for representing hierarchical data. These visualizations are capable of representing various topologies to differing extents, as shown in Figure 5.5. The visualizations are described in detail next, grouped by the categorizations from Chapter 3.

Connection-Based Visualizations use explicit lines to represent parent-child relationship between nodes.

The Linear Node-Link View is a connection-based tree representation, shown in Figure 5.6. Nodes are drawn horizontally in their levels from top to bottom, starting with the root node at the top. The current implementation of the view is designed to explicitly represent nodes, edges, siblings, levels, paths, bi-paths, and branches topologies.

In Figure 5.6, nodes are visually encoded as rectangles with node identifier numbers shown in them. Edges are drawn as connecting lines between their parent and child nodes. Siblings are each represented by a rounded rectangle enclosing all the child nodes under a common parent node. Levels are each represented by a horizontal rounded rectangle enclosing all

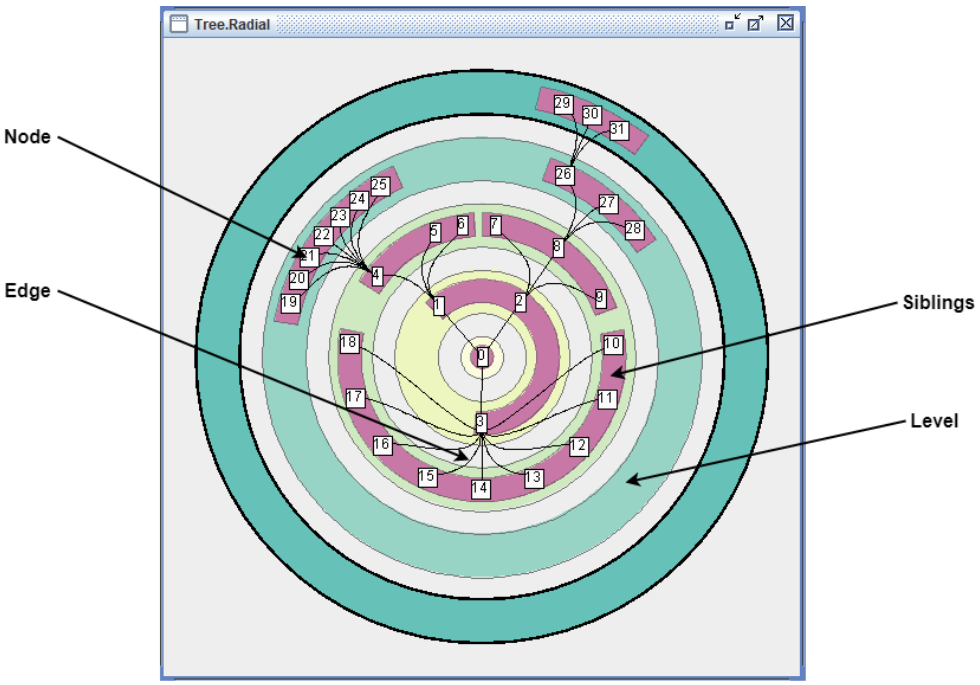


Figure 5.7: The Radial Node-Link View explicitly represents node, edge, sibling, and level topologies.

nodes and siblings that are equally distant from the root node. Paths and bi-paths are each represented by a squiggly line overlaid on its constituent edges. Figure 5.6 shows a bi-path between the first node at the third level from the top and second node at the second level from the top. Branches are each represented by a convex hull enclosing all the nodes in a branch. Figure 5.6 shows a branch with the third node in the second level as the root.

The topologies are drawn in the following order: levels, siblings, edges, paths, bi-paths, branch, and nodes. This ordering is chosen to minimize visual occlusion and ensure that each topology remains clearly visible. This order may vary depending on the specific tree visualization design. The shapes of nodes and edges can be replaced by various alternative glyph objects such as circles and arrows. Views can be specified such that selecting a sibling or a level highlights the corresponding topology as well as highlighting its constituent nodes. In Figure 5.6, the fourth level from the root node is shown as selected by highlighting both the level topology and its constituent node topologies.

The Radial Node-Link View is a connection-based tree visualization shown in Figure 5.7.

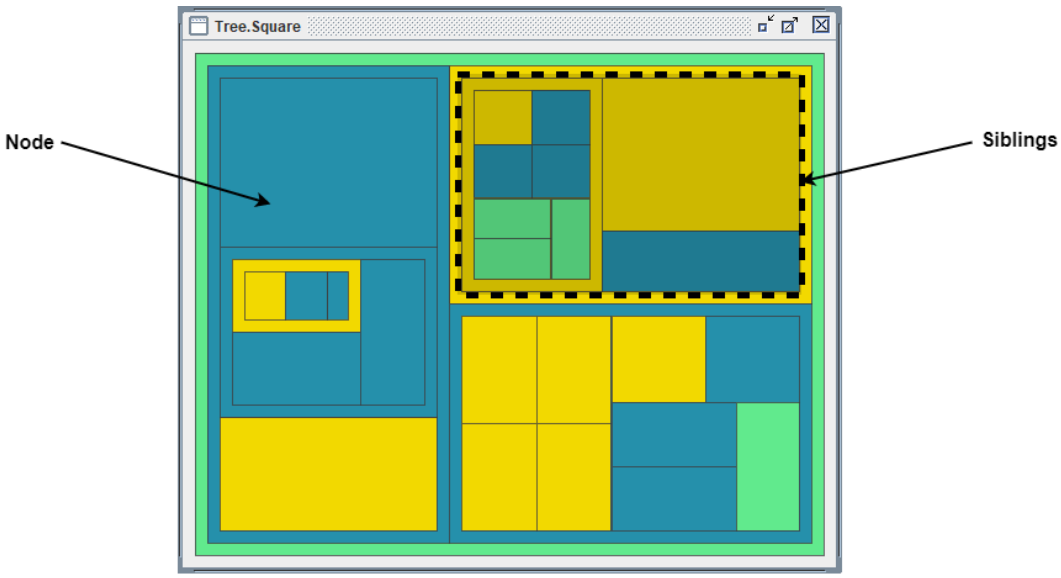


Figure 5.8: The Squarified Treemap explicitly represents node and siblings topologies.

Nodes are drawn radially outward in their levels starting with the root node at the center. The current implementation of the view is designed to explicitly represent nodes, edges, siblings, and levels. Edges are drawn as cubic curves connecting parent nodes to their child nodes. Siblings are represented by arc segments enclosing the child nodes under the same parent. Levels are represented by concentric rings. The shapes of edges can be replaced by straight lines, and those of nodes can be replaced by circles or any polygons. Alternative glyph choices for the topologies can be specified from the Improvise user interface.

The visual encodings of paths and branches can be implemented as waves and convex hulls in the radial node-link view, respectively much like in the Linear Node-Link View. Selecting a sibling or a level will highlight the corresponding topology as specified in its glyph design. Figure 5.7 shows the outermost level selected and highlighted with a solid black outline.

Containment-Based Visualizations use explicit groupings to represent parent-child relationships by enclosing child nodes within their parent node.

The Squarified Treemap is a containment-based tree visualization, as shown in Figure 5.8. Nodes are represented using rectangular glyph objects. Child nodes are enclosed within their

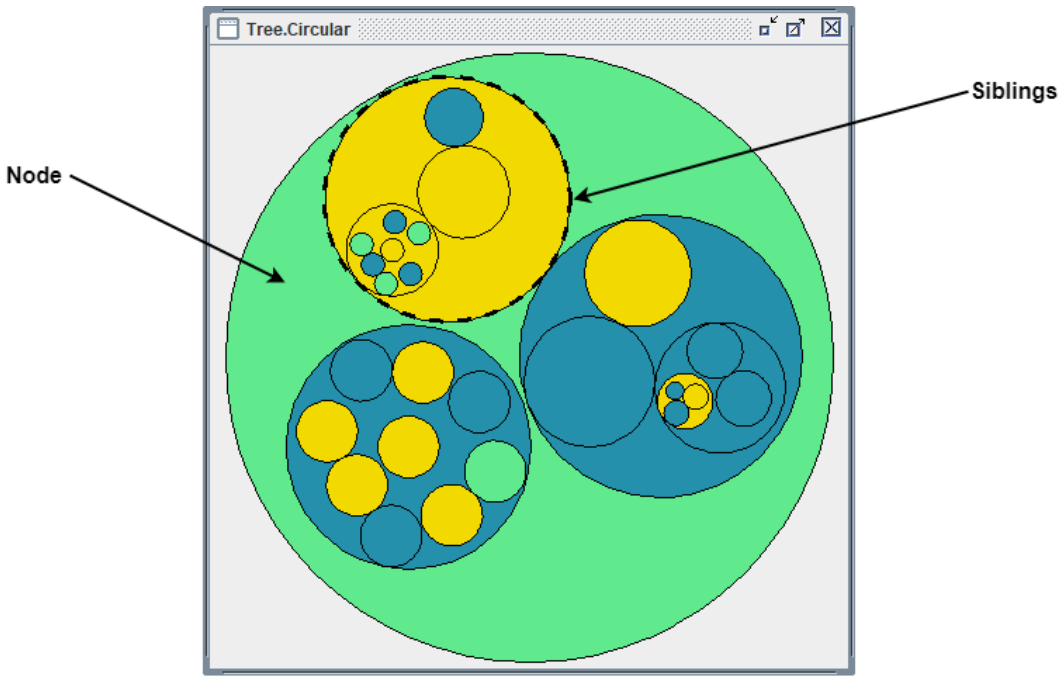


Figure 5.9: The Circular Treemap explicitly represents nodes and siblings topologies.

parent node by nesting rectangles. Siblings are represented explicitly by using a dashed outline drawn on the outer edge of the parent rectangle. Levels have no apparent representation in this technique but can be implicitly represented by highlighting nodes as proxies for their levels. Similarly, the view does not explicitly represent paths. Like siblings, branches can be explicitly shown using an outline on the outer edge of the root node of the branch. However the view can be visually ambiguous if it uses the border of nodes to represent both branch and siblings topologies. The lack of empty space surrounding node topologies and the tight packing of their rectangles makes it difficult to represent composite topologies on top of the nodes themselves. A small buffer area around each node can help, but leaves less space for nesting and hence limits the number of levels in the tree than can be drawn at all. Consequently it is challenging to interact with topologies other than nodes and edges (the latter through each edge's child node).

The Circular Treemap is a containment-based tree visualization, shown in Figure 5.9. Nodes are represented using circles. The child nodes of a parent node are enclosed within their parent circle. Siblings are represented by using a outline drawn on the outer edge of the

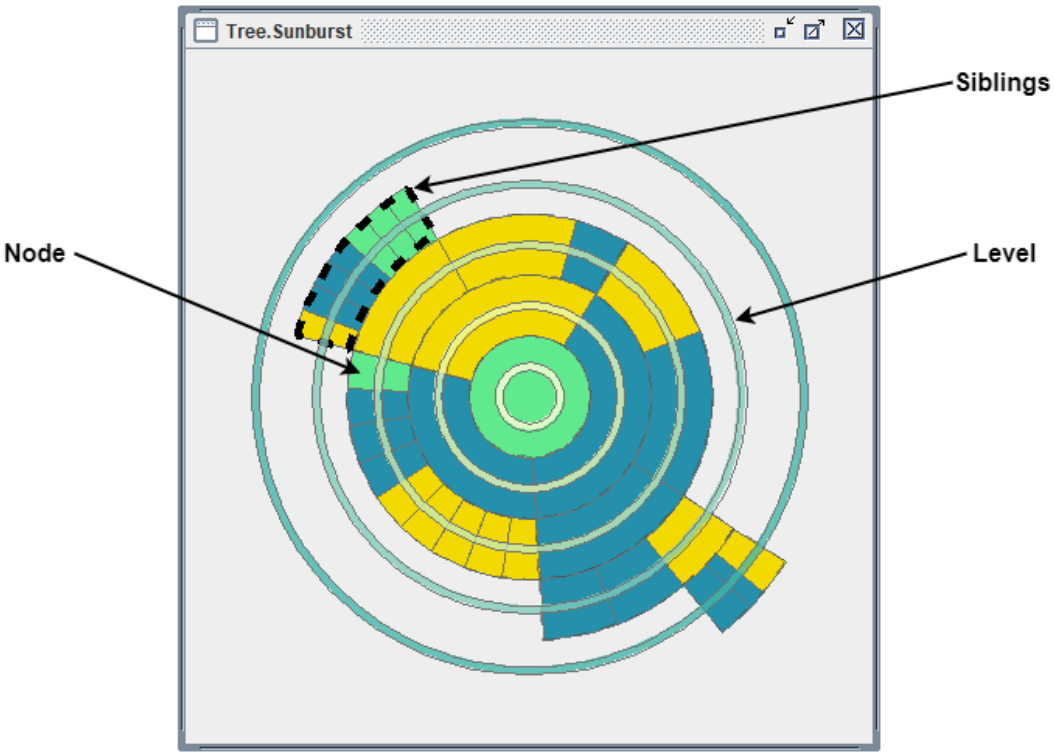


Figure 5.10: The Sunburst Chart explicitly represents node, siblings, and level topologies.

parent node of the siblings. Levels have no apparent explicit representation in this technique, but can be implicitly represented by highlighting the nodes which serve as proxies for their levels. Similarly, the view does not explicitly represent paths but can do so by implicitly highlighting the nodes along the path. Like siblings, branches can also be explicitly shown using a dashed outline on the inner edge of the root node of the branch.

The Circular Treemap suffer similar drawbacks as the Squarified Treemap; despite the ample free space within a parent circle not occupied by its children, it is not clear how to convey composite topologies other than representing them implicitly by drawing in their constituent nodes.

Alignment-Based Visualizations align one of the edges of the parent node.

The Sunburst Chart is an alignment-based tree visualization, shown in Figure 5.10. The root node is represented by a circular glyph, and all descendant child nodes are represented by arcs of various angular extent. The angle is proportional to some quantitative attribute of each parent's child node as shown in the figure, possibly including a simple count of the child

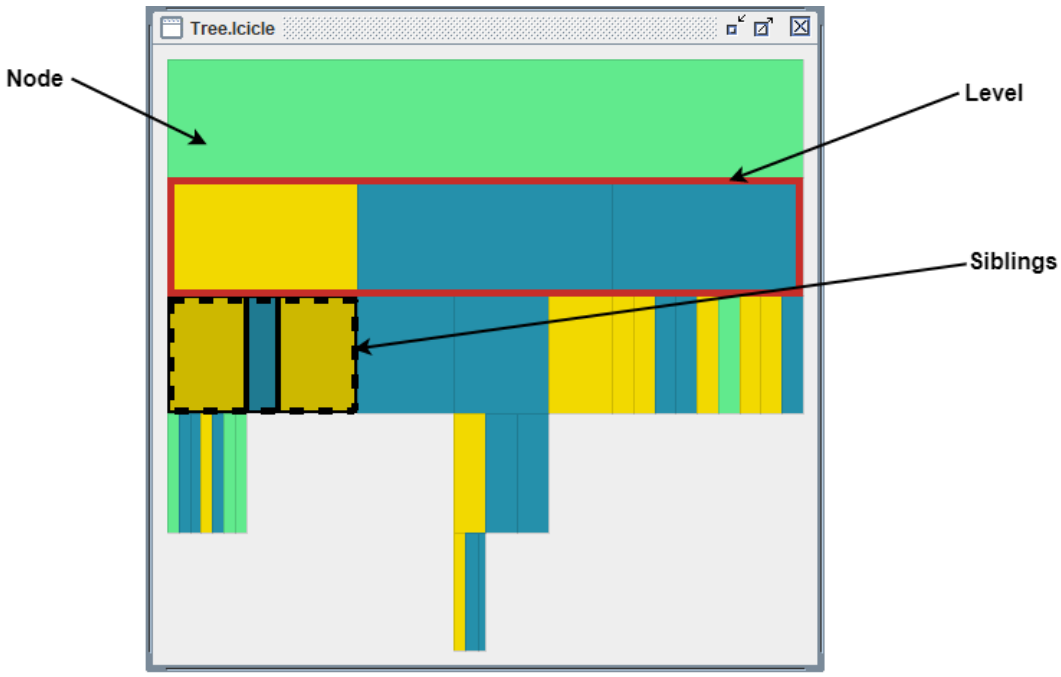


Figure 5.11: The Icicle Plot explicitly represents node, sibling, and level topologies.

nodes. The Root node is located at the center with child nodes progressively located along the arcs of their parents outward to each leaf node. Siblings can be explicitly represented using a dashed outline drawn along the outer edges of the nodes areas. Levels are explicitly represented using a thin 360° ring glyph that is drawn over the radial center of the areas of for all the nodes in each level. The view is not currently implemented with glyphs for paths, but one could do so by drawing a thin bar connecting the centers of starting and ending nodes. Branches could be represented explicitly by drawing a cone that extends outward from a given node and encompasses its descendants radially outward.

The Sunburst Chart suffers from challenges of overplotting when representing siblings, branches, and paths as their representation require drawing glyph on top of nodes also making it challenging to interact with them. Levels however are sufficiently distinct as both visual elements and interaction targets thanks to their low and uniform coverage of all nodes in concentric rings.

The Icicle Plot is an alignment-based tree visualization, as shown in Figure 5.11. All nodes of the icicle plot are represented using rectangular glyphs. Nodes in successive levels

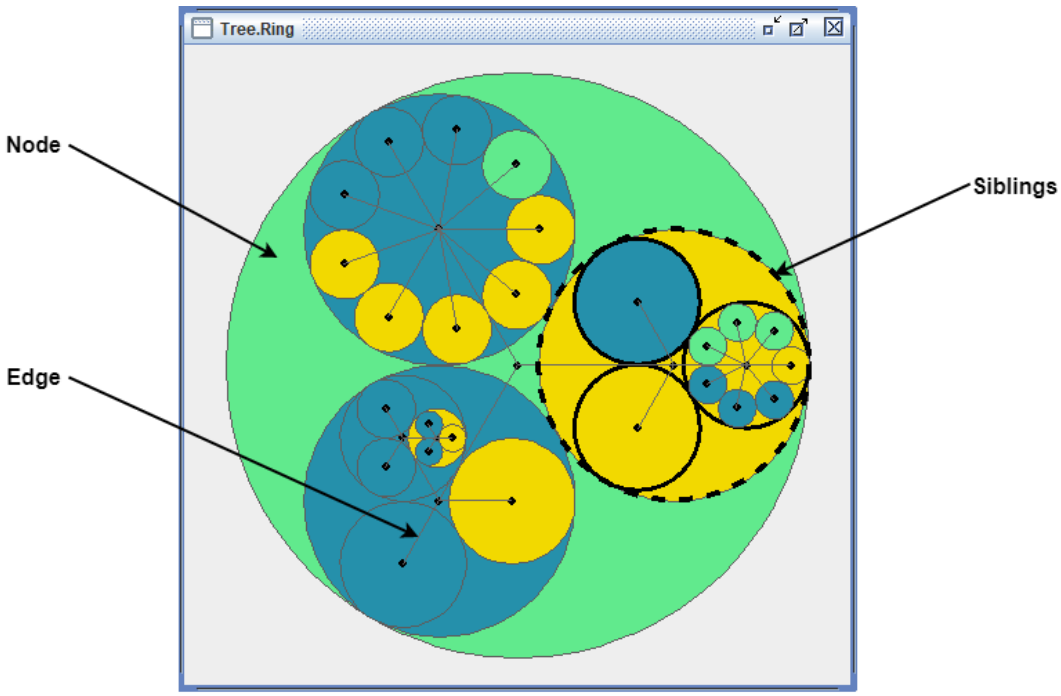


Figure 5.12: The Rings View explicitly represents node, edge, and siblings topologies.

are drawn from top to bottom, starting with the root node. Alternatively, nodes can be laid out from left to right instead of from top to bottom. The current implementation of the Icicle Plot does not explicitly show edges, but this could be done by allocating space above each child node to visually encode its incoming edge. Siblings are represented using a dashed outline drawn on the inner edge of the constituent nodes of the siblings topology, as shown in the Figure 5.11. Like the sunburst chart, this view is not implemented with explicit glyphs for paths, but one could do so with bars in much the same way, or implicitly represent paths by highlighting nodes along the paths. The view is also not currently implemented to explicitly represent branches, but this could be done with a rectangular envelop similar to how ones could be used in the Sunburst Chart. Also like Sunburst Charts, Icicle Plots face overplotting challenges in drawing levels, siblings, branches and paths that make these topologies difficult to interact with other than through their constituent nodes.

Hybrid Visualizations use a combination of connection, containment, and/or alignment to represent parent-child relationships.

	Connection-Based		Containment-Based		Alignment-Based		Hybrid
	Linear Node-Link	Radial Node-Link	Squarified Treemap	Circular Treemap	Icicle Plot	Sunburst Chart	Modified Rings
Node	Explicit	Explicit	Explicit	Explicit	Explicit	Explicit	Explicit
Edge	Explicit	Explicit	Implicit	Implicit	Implicit	Implicit	Explicit
Siblings	Explicit	Explicit	Explicit	Explicit	Explicit	Explicit	Explicit
Level	Explicit	Explicit	Implicit	Implicit	Explicit	Explicit	Implicit
Branch	Explicit	Explicit	Explicit	Explicit	Explicit	Explicit	Explicit
Path	Explicit	Explicit	Implicit	Implicit	Implicit	Implicit	Explicit
Bi-Path	Explicit	Explicit	Implicit	Implicit	Implicit	Implicit	Explicit

Figure 5.13: General capabilities of the seven example view types for representing different topologies.

The Rings Visualization is a hybrid layout that uses a combination of connection, containment and alignment. We use a modified version of the Rings layout algorithm [128] that we modified to provide more opportunities to explicitly represent composite topologies in a hierarchy. The modified layout algorithm as shown in Figure 5.12 differs from the original in multiple ways. All nodes are represented by circles. Instead of laying out children on multiple concentric rings inside a parent, we lay out child nodes as a single concentric ring just inside the parent's circle. Our algorithm also uses explicit shape to represent the center of each node's circles, which offers additional visual encoding channels for the nodes. Similar to The Circular Treemap, Rings uses circles to represent nodes; unlike Circular Treemap, the area of node circles in Rings is not proportional to a chosen magnitude, but is rather a subtle coincidence of nested siblings counts. Edges are represented explicitly using lines to connect the small center circles of parent and child node. Siblings are represented explicitly by drawing an outline along the inner perimeter of the parent node of the siblings. Branches could be explicitly represented similarly. Levels and paths are not explicitly represented in the current implementation, but could be using wavy lines as in the Linear Node-Link View.

Figure 5.13 summarizes the implemented composite topologies in the aforementioned seven tree visualization techniques, along with their potential representational support. Cells labeled "Explicit" indicate techniques that can readily represent topologies explicitly. Cells

	Connection-Based		Containment-Based		Alignment-Based		Hybrid
	Linear Node-Link	Radial Node-Link	Squarified Treemap	Circular Treemap	Icicle Plot	Sunburst Chart	Modified Rings
Node	Direct	Direct	Direct	Direct	Direct	Direct	Direct
Edge	Direct	Direct	Proxy	Proxy	Proxy	Proxy	Direct
Siblings	Direct	Direct	Indirect	Indirect	Indirect	Indirect	Indirect
Level	Direct	Direct	Proxy	Proxy	Indirect	Direct	Proxy
Branch	Indirect	Indirect	Indirect	Indirect	Indirect	Indirect	Indirect
Path	Indirect	Indirect	Proxy	Proxy	Proxy	Proxy	Indirect
Bi-Path	Indirect	Indirect	Proxy	Proxy	Proxy	Proxy	Indirect

Figure 5.14: Capabilities of the view types for interacting with topologies.

labeled "Implicit" indicate techniques that likely require falling back to implicit representations through nodes and/or edges.

Overall connection-based techniques are the most suitable for explicit representation and interaction with all kinds of topologies due to the advantages of the layout and spacing between nodes. However, all techniques have trade-offs. further study is needed to understand the strengths and weakness of various tree visualizations for depicting topologies.

5.3.6 Interaction Operations

Detection of Interaction Type and Target

Direct interaction with explicitly represented composite topologies occurs via mouse events. Indirect interaction with composite topologies are implicitly represented by nodes happens via keyboard events. Interaction with paths and branches happens this way in the currently implemented views. Predefined modifier keys serve to disambiguate interactions. For interaction with the other topologies, a view selects the topmost rendered topology as the interaction target. For example, if a sibling topology is drawn over a level topology and a click occurs over both, the sibling, being the topmost topology, will be selected as the interaction target.

Creation

Some topologies, such as paths and branches, are designed to be created lazily through interaction. These topologies are not generated from the node and edge data used to create the initial view of the tree. Only the specifications of how these topologies should appear are defined as glyphs using the Improvise expression language. Interactions that combine keyboard and mouse actions can trigger the creation of these topologies, followed by their selection. For instance, a path is created and selected when a starting node and ending node are selected in succession while pressing the keys *Ctrl + Z*. Similarly, a branch is created and selected by clicking on a node while pressing the keys *Ctrl + B*.

Alternatively, the path and branch topologies could be precomputed and rendered only when the user interacts with the corresponding nodes. (Whether to precompute or lazily generate topology objects is a practical issue that affects the time and space complexity of pipeline cases, particularly for topologies that are super linear in the number of nodes, such as the quadratic number of paths.)

Selection

Topologies can be explicitly selected by directly interacting with their explicit representations explicitly, or by indirectly interaction through their corresponding node and/or edge representations. Topologies can also be implicitly selected by selecting their constituent nodes whose highlighting then serves as a visual proxy for the topology. Selected items are highlighted according to each topology's glyph specification. Figure 5.14 summarizes the ways that topologies can be selected in the currently implemented views. Topologies that are implicitly represented and manipulated via nodes as proxies, can suffer visual ambiguity when a node serves as a proxy for multiple topologies simultaneously. For instance, a node could serve as a selection target as itself, as the parent of a siblings, as the root of a branch, and as a member of a level. In such cases, interactions must be designed to distinguish selection of multiple independent topologies though their shared node proxies. The glyphs for

the nodes and topologies must be correspondingly designed to provide different highlighting for the combinations of ways they might be selected, like in compound brushing [33].

Navigation

Navigation in tree visualizations broadly occurs in two ways to bring different portions of a tree into perspective: screen-space navigation, in which users can zoom, pan, or rotate; and structure-based navigation, which focuses on bringing portions of the tree into view based on topological landmarks. For instance, in structure-based navigation, users can navigate between levels in the linear and radial node-link views by moving up or down the hierarchy to select a range of one or more levels to display. Structure-based navigation over branches is also possible by moving into and out of branches down and up the tree. While these navigation operations can occur without explicit topology representation, explicitly representing topologies allows for direct interaction with the topologies themselves and visual encoding to emphasize the navigated portion(s) of the tree.

Structure Manipulation

Structure manipulation can be performed by changing the data that represents a tree's structure through interaction with its topologies. Structural manipulation is reflected by changes to the visual representation of the tree and the topologies in it. In connection and alignment-based views, for instance, scrolling the mouse wheel while the mouse pointer is positioned over a siblings topology could cycle through the siblings to reorder them, or arrange them based on their attribute information. Users could order the nodes in a level, branch, or entire tree by sequentially or recursively manipulating sets of siblings. As unitary interaction targets, explicit topologies can support a wide variety of structure manipulation designs. In general, structure manipulation could edit the entire structure of a tree by adding and removing nodes individually or in topological chinks. Such pervasive editing capability is left as future work.

Attribute Manipulation

Attribute manipulation can be performed by interacting with a topology to change the features and attributes represented by its visual encoding channels. For instance, a siblings topology in the Linear Node-Link View might possess a categorical data attribute, such as the collective character of the siblings as a group, and visually encode it as fill color. The ability to explicitly encode attribute information in composite topologies themselves increases design flexibility by providing a larger number of more specific targets to represent and manipulate hierarchical data than with node and edge representations alone. Exploring the design space of visually encoding topology attributes in explicit topological representations is a wide-open direction for future work.

Visually Representing Interaction Progress

Interactions with multiple steps, such as selecting a path by specifying source and destination nodes, require users to track their progress. We visualize intermediate incomplete interactions using additional glyphs: a text glyph to indicate the action and a circle glyph to mark the target location. For example, clicking a source node highlights it with a small circle and displays text identifying it as the source node. Figure 5.15 illustrates this example of interaction progress, showing "path start" when the source node (here the root node) is selected and "path end" when the destination node (here the third child node of the root node in left to right order) is selected, with two red circles marking the mouse click areas during path selection. In this case, the interaction would then be complete and the circles and text would then disappear when the modifier keys for path selection are released.

Similar visual representation of interaction progress can be used for interactions involving multiple steps such as selecting a siblings topology and then rotating the siblings to change the order of the siblings.

Coordinating Topologies across Tree Views

Hieros uses the Live Properties architecture of the Improvise [138] system to coordinate multiple tree views. Live Properties relies on having unique identifiers for coordinated data items, i.e., topologies. Applying data transformations such as filtering to node and edge data prior to the generation of tree data in the *Data Transformation Stage* can result in distinct tree structures with node, edge, and composite topologies whose identities do not align, precluding their coordination particular through selection. To address this issue, Hieros first

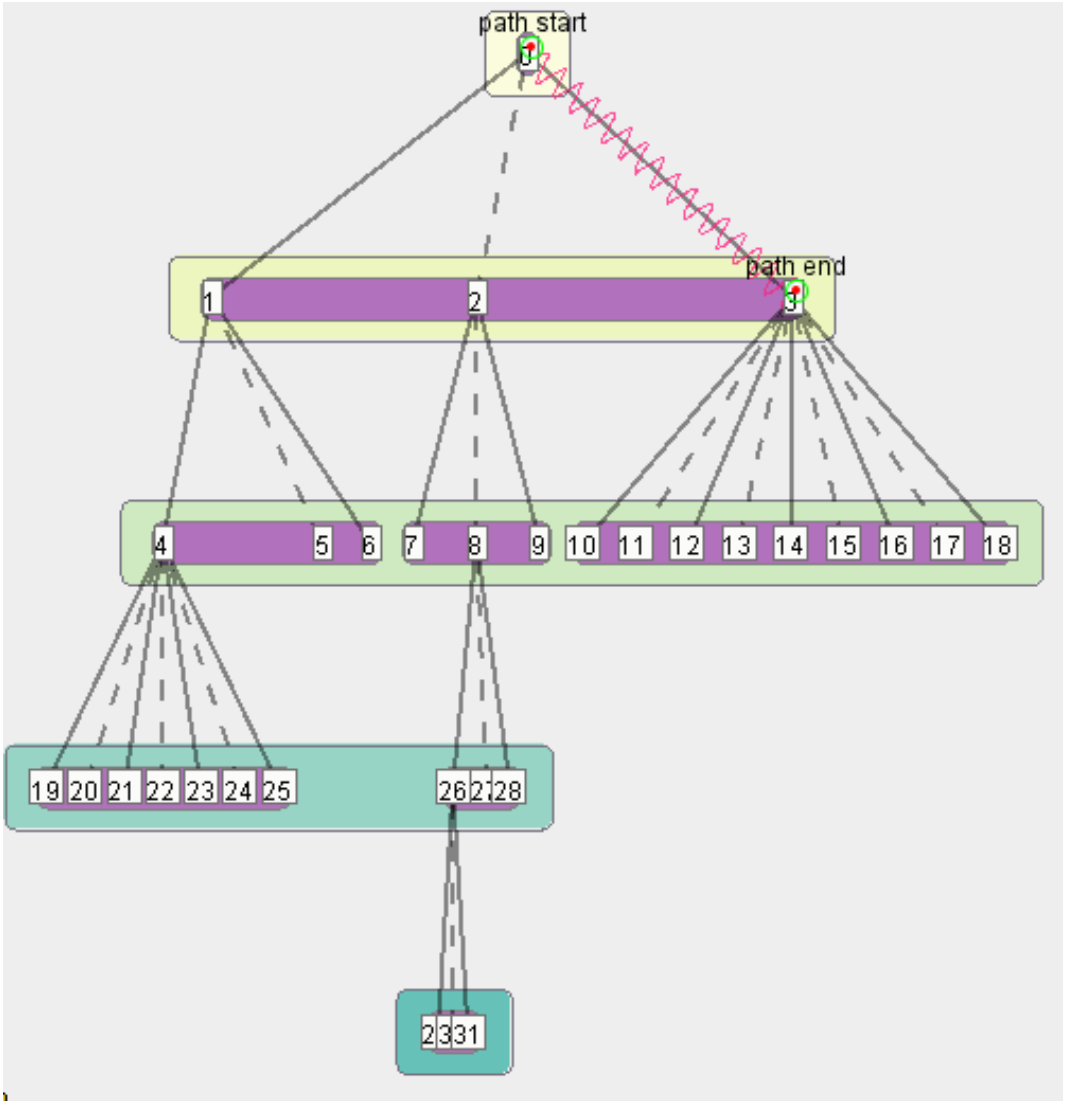


Figure 5.15: Linear Node-Link View showing intermediate interaction progress while selecting the path topology connecting the root of the tree and its third child.

constructs a reference tree data structure (containing all potential composite topologies) from the entire node and edge data of the source tree before any prior transformations are applied, and then maps the unique identifiers of the topologies from the reference tree to the rendered topologies. This allows views to correctly identify corresponding topologies across coordinated views for subsequent coordination. Figure 5.4 illustrates how unique identifiers *(ID)* are generated from the tree data structure for each of the topologies before applying any filter, sort or other data altering transformations.

Hierarchical views that use the same data and topology identifiers in this way can be coordinated through straightforward extensions of selection and navigation coordination patterns [140] to tree visualizations.

- **Shared selection of explicit topologies:** Views highlight the same topologies that are selected in any of the views. For instance, selection of nodes across views.
- **Shared selection via implicit topologies:** Views highlight the same topologies selected and highlighted through node and/or edge proxies in any of the views.
- **Shared selection via mix of explicit and implicit topologies:** Views coordinate topologies by selecting composite topology in some views and the constituent base topology in other views. For instance, selection of level topology in one view will highlight the constituent nodes of the level in another view. This pattern is useful in the common case of only some views being able to explicitly represent the topologies.
- **Shared navigation**: Views show the same part of the tree upon interaction with one of the coordinated tree views, such as the same range of values. This coordination pattern is achieved by sharing the parameter(s) involved in the screen space and/or structure based navigation operations and applying the same transformation to the Visual Mappings and View Transformations of the coordinated views.
- Other coordination patterns, such as **selection dependent data** and **selection-dependent encoding** as discussed by Weaver [140], can be applied through the Data

Transformation and Visual Mapping stages of the pipeline just like in non-tree Improvise visualizations.

5.4 Design Considerations

Fixed Order of Rendering of Topologies. For a given tree representation, the order in which topologies are rendered in a view is determined by taking the design of their representations and interactions into consideration. First, they should be ordered to minimize unwanted overplotting/occlusion. The way in which topologies are grouped together to form more complex topologies influences the drawing order of topologies. For instance, groups of nodes form siblings, and groups of siblings form levels. In the case of Linear Node-Link Views, drawing the levels first, followed by the siblings, and finally the nodes helps to prevent occlusion of the less complex topologies by their more complex ones. This approach also allows the surrounding space of simple topologies to be used for encoding various attributes of the more complex topologies. Occlusion can still happen through poor choices of visual encodings (such as overly large marks) for topologies above others in the drawing order. Second, they should be ordered to make them visible and reachable as interaction targets. A defined drawing order simplifies the design of efficient algorithms for checking topological containment, which is essential for operations involving them. Effective user learning and performance of selection operations depends on the knowledge of how interaction events (such as mouse clicks) will be successively hit tested against the topological layers in a view.

Closed Set of Glyph Choices for Topologies. For a given topology, each tree visualization design supports a limited set of glyph designs. The space of potential glyph designs is unlimited in principle. In practice, the layout algorithm of a view substantially limits the glyph designs that can be utilized without causing occlusions or generally making inefficient use of available space.

5.5 Implementation Considerations

Modular Layout Algorithms. Hierarchical data is diverse in its structural complexity. We implemented modular layout algorithms to allow designers to plug them into views flexibly according to their design goals. Users can switch algorithms interactively. For instance, designers can let users switch between squarified, slice and dice, and pivot layouts in the Treemap View using a radio button in the user interface. Different types and amounts of hierarchical data (such as biological taxonomies versus organizational charts) benefit from applying different layout algorithms without a need to implement new view modules. By modularizing layout, Hieros developers and designers can implement custom layouts to handle different space allocation needs, such as to accommodate the special layout concerns of text marks/labels or layout dynamics such as force-directed layout.

Rendering of Paths and Branches on Interaction. For a hierarchical tree with n nodes, the tree will have $\mathcal{O}(n^2)$ paths each with up to $\mathcal{O}(n)$ nodes and $\mathcal{O}(n)$ branches each with up to $\mathcal{O}(n^2)$ nodes, making for potentially cubic time and space complexity in both cases. Instead of pre-computing all paths and branches during the first render, we compute them lazily in response to relevant interactions with the constituent nodes. Interactions with a tree often involves filtering out nodes, which can cause frequent recalculations of paths and branches. Moreover, drawing all all possible paths or branches in a tree is not generally practical due to occlusion. How calculating and drawing only small numbers of paths and/or branches on demand trade off utility for usability is an important topic for future work.

Flexible Controls for Interaction. Through Live Properties, Hieros uses interactive parameters to drive alternative ways of performing operations in the visualization pipeline. User interaction in Hieros visualizations can be performed both by directly interacting with tree visualizations and their topological representations or by interacting with other views/controls. This is helpful for supporting operations on topologies that cannot be explicitly represented in tree visualizations. These parameters can control screen space navigational aspects of the layout, such as panning, zooming. They can also determine filtering of the tree data struc-

ture such as to set an effective root or limit the number of levels for structural navigation. These parameters can be coordinated with other views through their Live Properties, such as to support *shared navigation* coordination patterns between multiple tree views.

5.5.1 Deployment Requirements

Hieros is provided as an integrated library in Improvise for building tree visualizations using Improvise. As such, the deployment requirements of the Improvise carry over to the Hieros. Some of the essential Improvise external libraries required for Hieros include:

- **Java Swing** (http://java.sun.com/products/jfc/tsc/index.html) for user interface components,
- **Apache Xerces** (http://xerces.apache.org) for serializing visualization documents as XML files, and
- **Java Bindings for OpenGL** (JOGL, https://jogl.dev.java.net/) for 3-D rendering.

Improvise is packaged as two Java jar files—the main application (improvise.jar) and its supporting libraries (oblivion.jar along with jar files for its several dependencies). These files are deployed in two ways:

- As a UNIX shell script for execution from a command line.
- As a double-clickable, self-executing, cross-platform jar application.

Improvise is free software, distributed under the GNU Affero General Public License (AGPL).

5.6 Assessment of Hieros's Design Capabilities

Hierarchical visualization tools [67, 81, 103] are applications that support a data pipeline for operations such as filtering, sorting, and searching on tree visualization. Much of the tree

visualization research community has focused on evaluating the performance of these tools for particular tasks. Hieros is motivated by the potential to generate tree visualization data processing pipelines and thereby pursue of a substantially expanded tree visualization design space, and developing systems, libraries and tools for that space. In this section, we focus on comparing the relative strengths and weaknesses of existing systems that rely primarily on nodes and edges with those of the Hieros library and its data pipeline capable of handling topologies beyond nodes and edges.

5.6.1 Assessment of Data Transformations

A basic feature of a tree visualization system is to support data transformation operations such as filtering and sorting of items in the visualization. Existing tools like Treemap 4.0 [37] and SequoiaView [67] offer various kinds of filters for hierarchical data. In Treemap 4.0, filters based on node attributes (e.g., size, age) visually gray out nodes, while structural filters controlled by dynamic query sliders and widgets filter topologies from the tree. In contrast, SequoiaView applies structural filtering exclusively, effectively deleting items from the hierarchy before rendering them.

Hieros provides support for data transformations both on individual nodes and on on composite topologies. Filter and sort transformations can be performed on node data to prune sections of the hierarchy or sort the order in which nodes are seen. Filter Data transformations can also be applied on the tree data structure to filter out composite topologies. For instance, one can filter the number of levels to show in the tree by filtering out the nodes below that level in the hierarchy. Hieros also supports data transformations on the visual encodings of topologies to support visual filtering of items rather than filtering them from the data itself. In the current version of Hieros, views maintain data tables about the structural and attribute information of nodes. Filters can be applied while mapping the data in those tabled into the visual channels of topology glyphs.

5.6.2 Assessment of Layout Creation

The tree visualization design space is large and diverse [114]. Over 340 tree visualizations with different approaches to layout has been identified [113] with numerous applications to specific use cases. The need for a more generic layout approach inspired the creation of a generative layout approach for rooted tree drawings by Schulz, et al. [115]. This approach involves a six step layout process for drawing trees with an operator at each step to control the generation of the layout including to position the nodes and edges in the tree. It categorizes the generation of tree layouts into two approaches: top-down, starting from the root node or bottom-up, starting from the leaves. It emphasized the need for a minimal set of layout parameters and operations that are agnostic to the shape of node glyphs.

Hieros views use a combination of top-down and bottom-up approaches to generate tree layouts. Its layout algorithms are parameterized to take a minimal set of values that control the overall layout. For instance, the Radial Node-Link view uses a top-down layout approach that starts with the root node at the center and then extends each level outwards radially to the leaf nodes. The parameters needed for the layout are the scale and rotation angle of the view. The Linear Node-Link View on the other hand, uses a bottom-up approach which calculates the layout starting with the lowest level of leaves and proceeds upward to the root node. The parameters needed for this layout are the border margins and the scale of the view. These parameters help in making minor adjustments to the layout to account for different amounts of data and user preferences.

Hieros views make use of the window size to make sure the layout is drawn within the view's screen space. Each view process the tree data it displays to determine the items to be layed out, then calculates the positions of the tree's topologies in multiple passes either bottom-up or top-down. The initial pass first calculates the position of nodes and edges, followed by composite topologies in order of relative "containment", e.g., siblings as collections of nodes, followed by levels which are collections of siblings. This ensures that the topologies are positioned in keeping with the positions of the constituent nodes and edges

as well as any other topologies that they are dependent upon.

Each view's layout attempts to fit the view's available screen space according to the size and shape of glyphs generated for the displayed topologies. This approach is a design choice rather than a fundamental requirement of views in Hieros. It comes with both advantages and challenges. The advantage of using fixed glyph size is that it can guarantees a minimal size of glyphs for viewing and thus also to support interactive manipulation of the glyphs. The disadvantage is that the available screen space for the view and its layout may not be enough to draw all the glyph objects with the specified space and size without overplotting. This disadvantage is prevalent in visualizations and not specific to Hieros. An alternate approach is to layout the view in an unrestricted space relative to the specified shape and size of the glyphs, and then scale everything to fit the available screen space of the view. This approach lets the users see the entire tree in overview and pan and zoom to inspect desired portions. The zoom level determines the absolute size of the glyphs from their specified relative sizes. How tree layout algorithms can account for the geometric characteristics of composite topology visual encodings is an important direction for future research.

5.6.3 Assessment of Topology Representation

Most existing tree representations visualize hierarchies using only node and edge topologies. These nodes and edges are represented either implicitly or explicitly in different tree visualization designs. Some tree visualizations [81, 92, 67] represent composite topologies such as branches, paths, siblings, or levels by highlighting the constituent nodes and edges. For example, Rings [128] colors nodes to represent levels. In cases where a tree has a single node in a given level, for instance, highlighting the node is ambiguous because it may be intended to represent node or level information. Even so, it may not be possible to effectively use nodes to show information about multiple topologies simultaneously.

Composite topologies are distinct in structure and can have associated data attributes of their own. Kleiberg, et al. [66] uses 3D botanical visualizations to visualize file systems.

It uses a geometric model of a botanical tree to visualize a hierarchy highlighting structure via analogous topologies such as branches, and leaves. The visualization represents siblings as a sphere. Contact Trees [106] use botanical trees in a similar manner to visualize social networks.

Explicit representation of topologies facilitates straightforward interactive operations on these topologies, including to provide more relevant feedback during interaction. Tasks such as comparing topologies across a hierarchy become more straightforward by inspecting an entire visually encoded object rather than analyzing the composite structure through its individual nodes. For instance, the count of siblings can be encoded as a color of an explicit sibling topology instead of counting the sibling nodes themselves.

Other aggregate information about the constituent nodes of a sibling topology can be similarly encoded. Some visualizations like Treemap 4.0 [37] use nodes to visually encode aggregate information about hidden nodes in the branches beneath them using the root node of the branch. Mapped tree [111] (Node-Link variants) representations, particularly those that are axis parallel [113] have ample buffer space around nodes to explicitly represent topologies using area to visually encode attribute and structural information. The explicit edge representations in Node-Link views (see Figures 5.6, 5.7, and 5.12) are similarly suitable to visually encode paths.

5.6.4 Assessment of Interactive Capabilities

Interaction in hierarchical visualizations allows users to navigate, explore, and analyze tree structure. Navigation techniques fall into two broad categories. Global interactions such as pan and zoom [29] and focus+context [130] allow users to change the overall perspective and magnify regions of interest. In contrast, local interactions focus on specific parts of a tree, using methods like Semantic Zooming [95] and Geometric Distortion in Hyperbolic Browsers [69] to expand or collapse branches while maintaining context.

Selection techniques rely on visual cues like color, size, or animation to draw attention to

specific nodes, paths, or levels, as seen in Hierarchical Edge Bundling [61] and the Grammar of Graphics [146]. Filtering and querying further help reduce complexity by displaying only the parts of the hierarchy that meet certain criteria. Layout and reconfiguration interactions such as switching between different layouts, reordering siblings [75], or rotating a tree (van Ham & van Wijk, 2003 [133]) allow users to adjust visual structure. Features like tool-tips and linked detail views [81] provide details on demand. Structural manipulation techniques support direct modifications to the tree itself by adding, deleting, or moving nodes. Finally, comparison and juxtaposition techniques, for instance in TreeJuxtaposer [81], enable side-by-side analysis of different portions of a hierarchy.

Despite such exceptions, most existing interactions focus on individual nodes or overall tree structure, leaving a gap in supporting direct manipulation of intermediate topologies such as levels, siblings, branches, and paths. Techniques like structure-based brushing [48] can highlight clusters of similar items, but they do not fully address the need to treat these topological constructs as distinct entities. Hieros fills this gap by extending interaction support beyond nodes and edges to include such topologies. It allows users to interact directly with structures such as siblings and levels, enabling operations like structure-based selection and navigation. For example, users can select a topological element, automatically highlighting its entire structure, and navigate by progressively displaying or emphasizing elements like levels, making it easier to traverse the tree. Although these interaction techniques are currently implemented in Hieros example views as proofs of concept, this research opens the door for modifying existing methods such as focus+context to incorporate topological features. Interactions like collapsing or expanding branches can be extended to other topologies, e.g., collapsing siblings or nodes within a level, or reducing a path to a single edge. Navigation can be refined to allow movement between entire sibling sets or levels. By explicitly representing topologies, Hieros enhances traditional interaction methods and paves the way for substantially expanded research on techniques for exploration of hierarchical data.

5.7 Challenges and Future Work

Hieros adds capabilities to existing tree visualization techniques by adding support for representing composite topologies explicitly, performing interaction operations on those representations and extending data transformation operations on the composite topologies themselves. This work exposes some of the challenges and opportunities for future research on tree visualization design and systems. Most tree visualizations layouts and designs use a fixed layout to represent nodes and edges. Adding composite topologies to these layouts makes information denser. Fixed layouts constrain the choice of visual channels available to represent additional topologies. In the example Hieros view designs, the node-link views only utilize area and texture channel to encode information, and area is not ideal for representing quantitative data [77]. Further research is needed to explore the tree visualization designs and the flexibility of each of these topologies to represent various kinds of information in them.

The visualization pipeline of Hieros supports a variety of data transformation operations that involve the calculation of arbitrary information about the topologies. However the current implementation does this in an ad hoc manner inside the individual view implementations. In future work, the visualization pipeline should be extended to support surfacing these operations and make them accessible at a designer level outside view implementations, i.e., in the three stages of the visualization pipeline prior to view rendering.

5.8 Summary

We present the Hieros library for building interactive tree visualizations with composite topologies as first-class objects. Hieros introduces a hierarchical data pipeline that converts raw node and edge data into structured tree representations, while supporting operations at different stages of the data pipeline. Hieros uses the Improvise visualization system for declarative specification of tree designs, including the visual encodings of nodes, edges, and

composite topologies. Hieros also address performance concerns through lazy generation of composite topology objects and support for fast rendering of views during interaction. Our contributions are as follows:

- a **tree visualization system** that enables the interactive exploration of tree topologies, offering users a rich set of operations on these components;
- a **data pipeline** for hierarchical data that not only constructs the tree data structure but also exposes key stages of the visualization pipeline to support extensible creation and application of data transformation operators;
- **a sample set of interactive operations** that work at various stages of the tree visualization pipeline from creation of topologies, to selection of topologies, to navigation over topologies, to manipulation of topologies;
- **seven variants of known tree visualization techniques**—linear node-link view, radial node-link view, squarified treemap, circular treemap, sunburst chart, icicle plot, and rings—that are capable of representing **common composite topologies** to cater to different types of hierarchical data and analytical tasks;
- **guidance for designing glyphs** for these topologies, ensuring that visual encodings are both meaningful and compatible with interactive operations and visualization designs; and
- **an assessment** of the various capabilities of Hieros in terms of its data pipeline, layout algorithms, representation of topologies, support for interaction with topologies, and support for coordination using topologies.

By integrating these components, Hieros assists designers in creating analytically rich tree visualizations. The library's modular architecture and interactive capabilities facilitate efficient data exploration and support coordinated views, thereby opening new avenues for analyzing complex hierarchical information across a wide range of domains.

Chapter 6

Evaluation of Topology-Augmented Tree Visualizations

6.1 Overview

Hierarchical visualizations typically focus on explicitly representing nodes and edges, while composite topologies are visually represented implicitly (if at all) through the visual encoding of nodes and edges [81]. In a hierarchy, topologies can convey information about their structure or the attributes they store. For example, a file represented as a node in a file hierarchy may include structural information, such as its depth relative to the root, or attribute information, like file type and file size. Similarly, composite topologies can possess both structural and attribute information. For example, the children of a parent in a genealogy hierarchy comprise a sibling topology which can include structural information such as the generation of the children or attribute information such as a property of the siblings as a group. However, information about composite topologies is usually treated as data associated with a node that serves as an anchor for the composite structure. For instance, Pandey, et al. [92] do not represent siblings topologies explicitly; instead, the

number of nodes in a siblings topology is described as the *Degree* property of the parent node, rather than as a property of the sibling topology itself, effectively treating the sibling topology as an attribute of the node topology.

We conducted a participant-based evaluation to assess the utility and usability of the topology-augmented tree visualizations developed using the Hieros library. The visualizations show node, edge, siblings, level, path, and branch topologies. Operations on topologies involve creation, selection, navigation, and modification of topologies. Qualitative results indicate that participants found visualization of the composite topologies themselves to be helpful for understanding those structures and their attribute information, as well as useful for performing tasks on them. They expressed confidence in interacting with the structures and found the visualizations appealing. Qualitative results confirm that participants were able to perform tasks on structures accurately and quickly, with an average accuracy of 90% across all tasks and an average mean completion time of well under two minutes for the most complicated composite structure manipulation task tested. These findings suggest that augmenting tree visualizations with explicit representations of composite topologies increases their interpretability, is found useful by participants for data exploration and analysis, supports more straightforward interactions with the topologies, and results in an overall more satisfying user experience.

6.2 Method

6.2.1 Participants

A total of 17 undergraduate and graduate students over the age of 18 were recruited from the School of Computer Science, the Data Science and Analytics Institute, and the School of Electrical and Computer Engineering at the University of Oklahoma (OU). Participants were recruited through advertisements posted on bulletin boards and via emails distributed by administrators and faculty in Computer Science. Initially, three participants (P1 to P3)

were assigned to a pilot study group, and fourteen participants (P4 to P17) were assigned to the main study. The pilot study was conducted to refine observation protocols and data collection methods and fix any issues with the apparatus setup. For the rest of this chapter, "participants" refer to the participants of the main study (P4 to P17).

All participants were familiar with hierarchical data. Sixteen reported being knowledgeable about file hierarchy data, 14 indicated familiarity with organizational hierarchies, nine were familiar with genealogical hierarchies, and two were familiar with geospatial hierarchies. Additionally, all participants had experience working with interactive data visualizations and performing tasks such as selection, navigation, and highlighting items. While all participants indicated familiarity with parent-child relationships, many were uncertain about other types of relationships. Eight participants reported familiarity with composite topologies, including ancestor-descendant relationships (paths), siblings, leaves, and cousins.

Among the 14 participants, six had completed a full-semester visualization course, three attended a short course or workshop, two majored in the field, and three were self-taught or had no formal training. In terms of experience using visualizations, ten participants had less than one year, while four participants had one to five years. Regarding the types of visualizations they had worked with previously, eleven participants indicated they had used interactive visualizations that responded to mouse and keyboard input, whereas three participants had worked with static visualizations.

All participants received training on identifying topologies in linear and radial node-link tree views and performing interactions on them in the visualizations used for the experiment.

6.2.2 Experimental Setup

The experiment was conducted in a laboratory at the University of Oklahoma using a 16-inch LCD laptop with an external mouse and keyboard. The stimuli consisted of three visualization applications running in the Improvise [138] visualization environment on the entire screen. Participants received written instructions explaining the visual encoding of

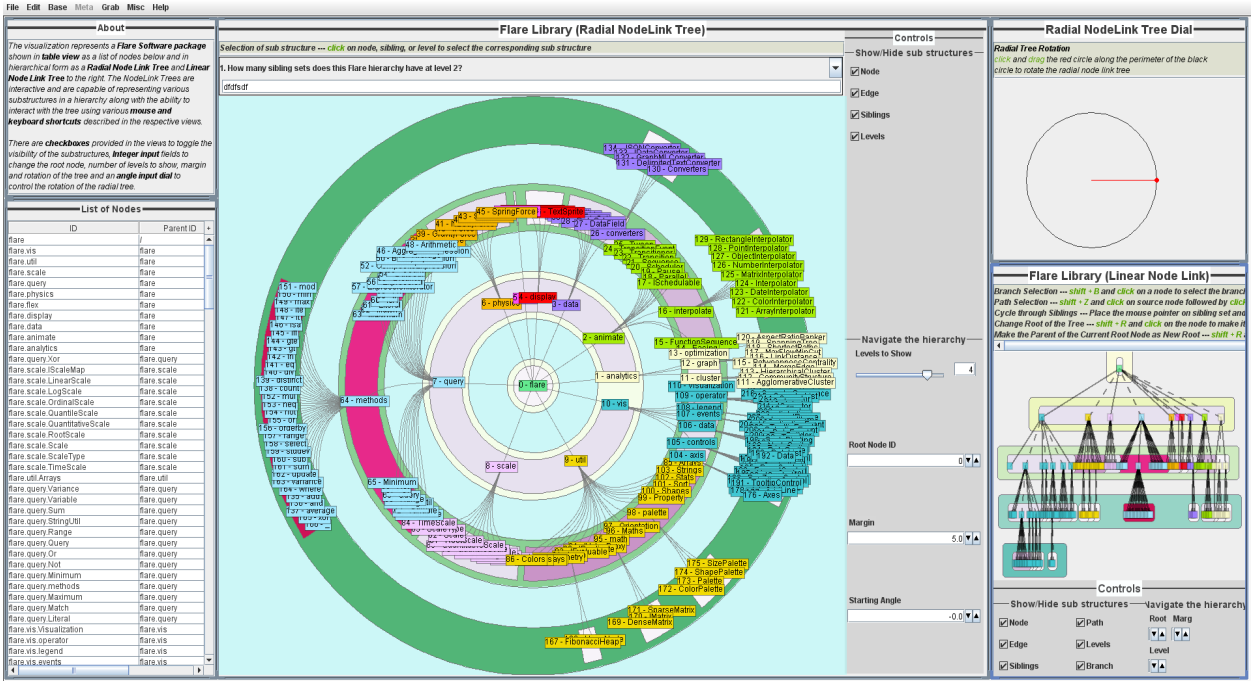


Figure 6.1: The File Hierarchy Visualization as an example of a node-link view capable of showing various topologies independently. The visualization includes a table view on the left, a radial node-link view in the center and linear node-link view on the right. A circular dial (top right) is provided to rotate the tree. Additional controls for radial node-link view (center right), and linear node-link view (bottom right) are provided to perform widget-based interactions on the respective tree views.

the various composite topologies and the methods for interacting with them. Data from the experimental procedure was recorded through screen recordings of the laptop.

6.2.3 Usage Scenarios

For the experiment, we selected two commonly understood hierarchical data domains: (1) file hierarchies and, (2) organizational hierarchies. We conducted the experiment in two phases. The first phase involved two usage scenarios in which participants performed directed tasks. The second phase involved a usage scenario in which participants performed open-ended exploration of the data to answer a broader question. Below are the details of the usage scenarios and the example visualizations used.

Usage Scenario 1: File Hierarchy

This usage scenario involves participants performing directed tasks on topologies within a file hierarchy visualization. The hierarchical data used for the visualization comes from a software package called Flare from Prefuse [58] (Flare is a flash version of prefuse). The Flare package comprises multiple sub-packages, many containing their own sub-packages, which altogether form a hierarchical organization of the software library. Each package is represented as a node in the hierarchy. The file hierarchy data contains a total of 252 nodes. Each package has three attributes: The name of its parent package, its size, and its category.

The File Hierarchy Visualization is a coordinated multiple view visualization (Figure 6.1) with three main views as follows.

- A table view shows the list of nodes in the hierarchy along with their attributes in the columns of the table.

- A radial tree is the main view where the participants performs the tasks. Is consists of the file hierarchy data represented as a radial tree. The tree is capable of representing nodes, edges, and two composite topologies: siblings and levels. Each kind of topology has a checkbox to toggle its visibility. There are controls to filter the levels of the tree, set the viewed root of the tree, rotate the tree, and adjust the outer margin (padding) of the tree. A radial dial is also provided to rotate the tree. The color of each node represents the categorical value representing the name of the ancestor of a node at level one (second level from the root node). The color of each siblings is proportional to the count of nodes in the siblings relative to other siblings. The color of each level is proportional to the count of nodes in the level relative to the other levels.

- A linear node link view is capable of representing nodes, edges, and five composite topologies: siblings, branches, levels, paths, and bi-paths. Each kind of topology has a checkbox to toggle their visibility. The tree also has controls to adjust the margin (outer padding), set the viewed root node, and adjust the number of levels to show

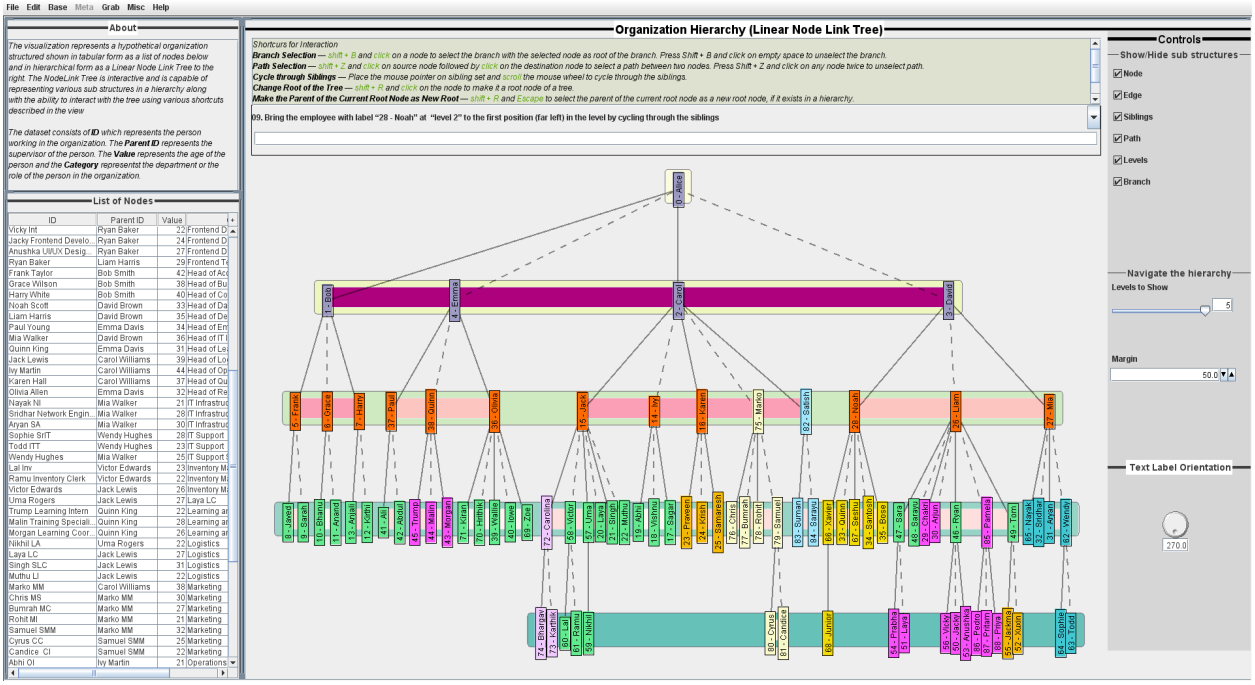


Figure 6.2: The Organizational Hierarchy Visualization, as an example of a node-link visualization capable of showing various topologies independently, showing six topologies explicitly represented in combination.

below the root. The color of each siblings is proportional to the count of nodes in the siblings relative to other siblings. The color of each level is proportional to the distance of the level from the root node relative to the other levels.

The visualization also contains a drop-down menu listing all 8 tasks that need to be performed on the visualization. Each participant selects a given task from the drop-down menu prior to performing the task in the visualization's views.

Usage Scenario 2: Organizational Hierarchy

This usage scenario involves participants performing simulated directed tasks on topologies within an organizational hierarchy visualization. The hierarchical data comprises 32 employees organized by their supervisoremployee relationships. Each employee is represented as a node in the hierarchy. Each employee node includes three attributes: the name of the supervisor, the age of the employee, and the department of the employee.

The Organizational Hierarchy Visualization is a coordinated multiple view visualization (Figure 6.2) with two main views as follows.

- A table view shows the list of employees in the hierarchy along with their attributes in the columns of the table.
- A linear node link tree representing the organizational hierarchy is capable of representing nodes, edges, and five composite topologies: siblings, paths, bi-paths, branches and levels. Each kind of topology has a checkbox to toggle their visibility. The tree also has controls to adjust the margin (outer padding), adjust the number of levels to show below the root, and rotate the text labels of the nodes. The color of each siblings is proportional to the average age of the nodes in the siblings relative to other siblings. The color of each level is proportional to the distance of the level from the root relative to the other levels. Paths are shown by selecting a source node and a destination node with a dedicated key pressed. Similarly branches are shown by selecting the root node of the branch while pressing a dedicated key.

The visualization also contains a drop-down menu listing all 13 tasks that need to be performed on the visualization. Each participant selects a given task from the drop down menu prior to performing the task in the visualization's views.

Usage Scenario 3: Coordinated File Hierarchy Visualization

This usage scenario involves participants conducting open-ended exploration task on topologies using a coordinated multiple view visualization. For this usage scenario, we used the same file hierarchy data as the Usage Scenario 1 to represent the software library.

The Coordinated File Hierarchy Visualization used in this usage scenario is a modified version of visualization in Figure 6.1. The Coordinated File Hierarchy Visualization is shown in Figure 6.3. The visualization is a of coordinated multiple view visualization with two main views, as follows.

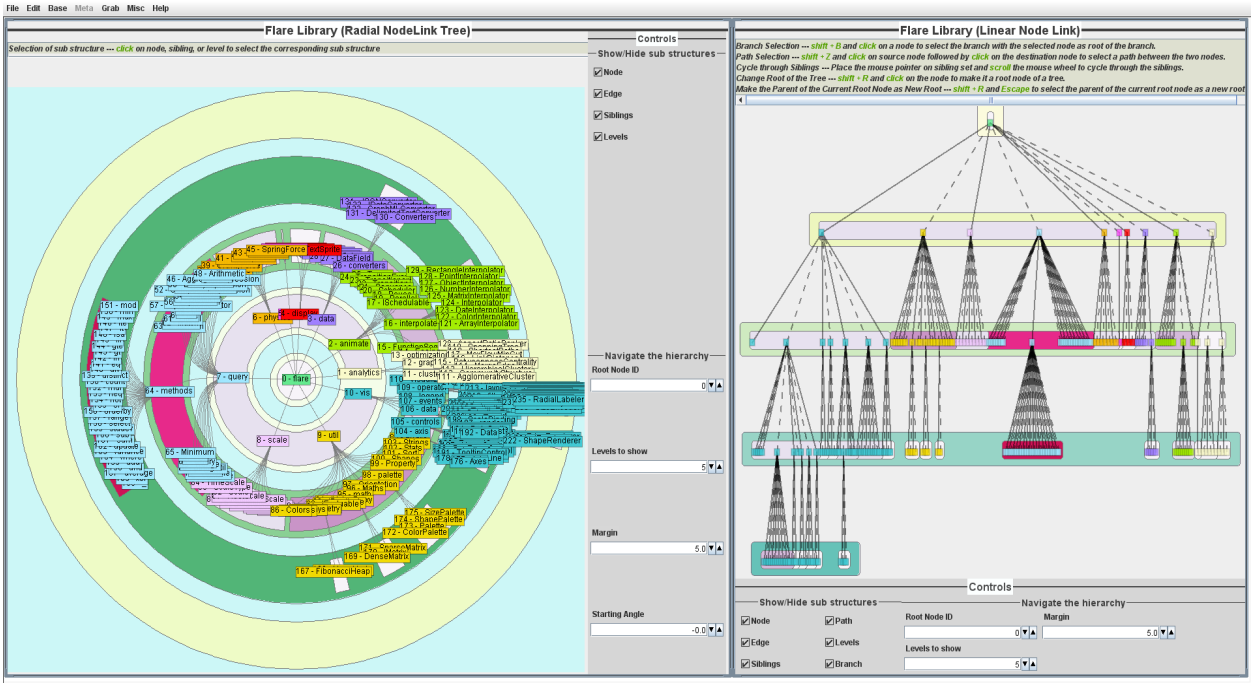


Figure 6.3: The Coordinated File Hierarchy Visualization as an example of node-link visualization capable of showing various topologies independently, showing six topologies explicitly represented in combination with topology coordination between the views.

- A radial node-link view is the main view where participants perform tasks. Is consists of the file hierarchy data represented as a radial tree. The tree is capable of representing nodes, edges, and two composite topologies: siblings and levels. Each kind of topology has a checkbox to toggle its visibility. There are controls to filter the levels of the tree, set the viewed root of the tree, rotate the tree, and adjust the outer margin (padding) of the tree. A radial dial is also provided to rotate the tree. The color of each node represents the categorical value representing the name of the ancestor of a node at level one (second level from the root node). The color of each siblings is proportional to the count of nodes in the siblings relative to other siblings. The color of each level is proportional to the count of nodes in the level relative to the other levels.

- A linear node link tree representing the organizational hierarchy is capable of representing nodes, edges, and five composite topologies: siblings, paths, bi-paths, branches and levels. Each kind of topology has a checkbox to toggle their visibility. The tree

also has controls to adjust the margin (outer padding), adjust the number of levels to show below the root, and rotate the text labels of the nodes. The color of each siblings topology is proportional to the average age of the nodes in the siblings relative to other siblings. The color of each level is proportional to the distance of the level from the root relative to the other levels. Paths are shown by selecting a source node and a destination node with a dedicated key pressed. Similarly branches are shown by selecting the root node of the branch while pressing a dedicated key.

Interactions performed on same topologies in one of the views are reflected in the corresponding topology in the other view. Nodes are coordinated across the views such that node selections performed in either view will highlight the nodes in both. Siblings and levels are coordinated across the views such that selecting a sibling or a level topology in either view will highlight the corresponding item in both views. Views are also coordinated through their widget controls such that setting the viewed root of the tree, or changing the number of levels to show below the root in either view adjusts those settings for both views.

6.2.4 Tasks and Procedure

Usage Scenario 1 & 2: Directed Tasks

The first phase of the experiment asked participants to perform tasks on the visualizations. The study is a one group design with the tasks on topologies as independent variables, and speed and accuracy of task performance as dependent variables. The tasks are split into two groups by domain: participants performed tasks in the File Hierarchy Visualization followed by tasks on the Organizational Hierarchy Visualization. The former involved tasks on siblings, levels, and the entire tree. The latter involved tasks on nodes, edges, siblings, levels, paths, bi-paths and branches. Participants were asked to select each task from the drop down menu in each of the visualizations to prepare the visualization for performing the tasks. We defined 21 tasks involving various topology queries across the two domains.

Topology	Type	T#	Description
Node	A	T3	Select a sibling group where the parent node is labeled "9-util".
Sibling	S	T1	Count the sibling groups at level 2 in the Flare hierarchy.
Sibling	S	T9	Reorder siblings to move “28 - Noah” to the first (leftmost) position in level 2.
Sibling	S	T12	Select and highlight the edge between “1 - Bob” (level 1) and “6 - Grace” (level 2), then move it to the far-right side.
Sibling	S	T13	Select and move “36 - Olivia” to the leftmost position in level 2.
Sibling	A	T2	Identify the parent node of the sibling group that contains the most files.
Sibling	A	T6	Identify the parent module of the largest sibling group at level 2.
Sibling	A	T10	Identify the supervisor of the largest sibling group at level 3.
Path	S	T16	Highlight the path between “8 - Javed” (level 3) and the root node “0 - Alice”.
Bi-Path	S	T14	Select and highlight the path between “5 - Frank” (level 2) and “62 - Wendy” (level 3).
Bi-Path	S	T15	Determine the length of the selected path between “5 - Frank” and “62 - Wendy”.
Bi-Path	S	T17	Identify the common supervisor (ancestor) of “13 - Anjali” and “16 - Karen”.
Level	S	T4	Determine the total number of levels in the Flare hierarchy.
Level	S	T7	Limit the visible depth of the radial tree to a maximum of three levels from the root.
Level	S	T11	Select and highlight the lowest level in the hierarchy (farthest from the root).
Level	A	T5	Find the level with the most packages, considering level 0 as the starting point.
Branch	A	T19	Count the number of levels the branch rooted at “15 - Jack” spans.
Branch	S	T18	Select and highlight the branch where “15 - Jack” is the parent node.
Tree	S	T8	Rotate the tree to place the node labeled “72-variable” at the topmost position.
Tree	S	T21	Set the parent (supervisor) of “26 - Liam” as the new root.
Tree	A	T20	Set “26 - Liam” as the root of the tree.

Table 6.1: Evaluation tasks grouped by topology and information type, (A: attribute information; S: structural information).

The tasks are listed in Table 6.1. The user interface for selecting tasks looks similar to the Hierarchical Visualization Testing Environment (HVTE) [11], the difference being that each participant needs to restart the Hieros visualization environment for each session instead of selecting the participant from the side navigation bar. The tasks in Table 6.1 can be further

Task	Usage Scenario	Targeted Topology	Target Action	Type
T1	File Hierarchy	Sibling	Find	Structure
T2	File Hierarchy	Sibling	Find	Attribute
T3	File Hierarchy	Node	Select	Attribute
T4	File Hierarchy	Level	Find	Structure
T5	File Hierarchy	Level	Find	Attribute
T6	File Hierarchy	Sibling	Find	Attribute
T7	File Hierarchy	Level	Navigate	Structure
T8	File Hierarchy	Tree	Find + Navigate	Structure
T9	Organization Hierarchy	Sibling	Find + Navigate	Structure
T10	Organization Hierarchy	Sibling	Find	Attribute
T11	Organization Hierarchy	Level	Select	Structure
T12	Organization Hierarchy	Sibling	Find+Navigate	Structure
T13	Organization Hierarchy	Sibling	Find+Navigate	Structure
T14	Organization Hierarchy	Bi-path	Select	Structure
T15	Organization Hierarchy	Bi-path	Find	Structure
T16	Organization Hierarchy	Path	Select	Structure
T17	Organization Hierarchy	Bi-path	Find	Structure
T18	Organization Hierarchy	Branch	Select	Attribute
T19	Organization Hierarchy	Branch	Find	Structure
T20	Organization Hierarchy	Tree	Find + Navigate	Attribute
T21	Organization Hierarchy	Tree	Navigate	Structure

Table 6.2: Usage scenario, targeted topology, target action, and information type of each task.

classified into tasks by topology and the actions performed, as shown in Table 6.2.

Usage Scenario 2: Open-ended Exploration Task

Participants next performed open-ended exploration of the Coordinated File Hierarchy Visualization (Figure 6.3) using its two coordinated views. Participants were asked to explore levels 2 and 3 in the Flare Package hierarchy and compare various various aspects of siblings, levels, and nodes using the two views. They were given 10 minutes to perform the task and were asked to answer the following questions after completing the task:

1. What insights did you discover while exploring the levels? Which features of the visualization helped in your analysis? Please write a brief summary in a few lines.

2. What challenges did you encounter while exploring the data, if any, in using both

Scale	Learnability	Utility	User Preference & Ease of Use
2	Very Confident	Extremely Useful	Strongly Agree
1	Somewhat Confident	Very Useful	Agree
0	Neutral	Somewhat Useful	Neutral
-1	Somewhat Unconfident	Not Very Useful	Disagree
-2	Very Unconfident	Not Useful at All	Strongly Disagree

Table 6.3: Likert-scale used for responses across different evaluation criteria

visualizations? Please write a brief summary in a few lines.

Post-Experiment Questionnaire

After completing both directed and open-ended tasks, participants were asked to fill out a questionnaire consisting of 15 five-level Likert-scale questions, with allowed responses shown in Table 6.3. These questions assess the **utility**, **user preference**, and **learnability** of the visualizations for performing the tasks. (Note: In the questionnaire, we refer to topologies as substructures.)

Learnable (Scale: *Very Confident* to *Very Unconfident*)

1. I can identify substructures in a hierarchical visualization technique.
2. I can interact with substructures of interest to perform selection and modify the layout with little effort.
3. I can choose an appropriate hierarchical visualization that effectively shows substructures for my tasks.
4. I can quickly obtain an overall understanding of hierarchical data by looking at substructure representations.

Useful (Scale: *Extremely Useful* to *Not Useful at All*)

5. Viewing multiple visualizations of the same hierarchical data simultaneously.

6. Linking hierarchical visualizations via selection to highlight substructures across views.

7. Seeing alternative visual representations of substructures for a given visualization type.

8. Using different interaction methods to change the root of the hierarchy.

9. On-demand display of substructures like paths and branches via shortcuts to reduce interface complexity.

Easy to Use (Scale: *Strongly Agree* to *Strongly Disagree*)

11. I found visualizations with substructures intuitive and easy to understand.

12. It is easy to recognize and interpret hierarchical relationships using substructures.

13. Cognitively, it is much easier to understand the hierarchy if substructures are represented.

Prefer to See Substructures (Scale: *Strongly Agree* to *Strongly Disagree*)

10. I prefer hierarchical visualizations that include substructures.

14. Displaying substructures (e.g., siblings and levels) does not reduce the aesthetic appeal of the visualization.

6.2.5 Usability & Utility Measures

We assessed the usability and utility of representing composite topologies in tree visualizations through a combination of quantitative and qualitative measures.

We evaluated **usability** by measuring *learnability*, *accuracy*, *ease of use*, and *user preference* through the time and accuracy of participant performance on directed tasks, along with corresponding qualitative feedback.

1. **Learnability:** We estimated learnability by measuring the task completion times while performing the tasks and from the assessment of the survey questions on *learnability*.

2. **Accuracy:** We evaluated the accuracy with which the participants were able to accomplish the tasks from the task correctness data.

3. **Ease of Use:** We analyzed the qualitative feedback obtained through the post-tasks survey from the *ease of use* category of responses to estimate perceived ease of use of the visualizations.

4. **User preference:** We analyzed the qualitative feedback obtained through the post-tasks survey from the *user preference* category of responses to estimate the perceived user preference of working with topology-augmented tree visualizations.

We evaluated **utility** by assessing the *suitability* and *effectiveness* of representing composite topologies in visualization through the correctness of participant performance on directed tasks, along with corresponding qualitative feedback.

1. **Suitability:** We estimated suitability by measuring whether the participants were able to complete the tasks correctly as well as assessing their feedback from the *useful* category of responses in the survey.

2. **Effectiveness:** Effectiveness was assessed based on the participant feedback obtained from the responses to the open-ended questions in Usage Scenario 3.

Table 6.4 shows the time taken (in seconds) by each participant for each task. Table 6.5 presents the correctness of each task per participant.

6.3 Quantitative Results

We analyzed task completion speed and correctness across tasks in both usage scenarios mentioned in Section 6.2.3. Additionally, we examined individual participant performance

T/P	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21
P4	98	156	20	12	31		12	32	125	89	13	121	33	76	19	63	75	32	15	45	23
P5	50	60	25	28	17	27	9	23	18	66	18	35	50	46	13	13	26	26	14	16	38
P6	61	32	40	11	14	26	6	32	35	55	22	71	33	75	17	25	61	21	19	26	19
P7	27	83	24	17	11	23	31	18	42	80	23	78	46	75	15	29	79	54	9	53	37
P8	82	104	59	18	24	41	58	40	152	64	36	84	28	34	17	29	51	59	9	71	13
P9	55	152	30	25	41	38	10	23	35	65	14	30	20	59	20	26	58	19	22	22	38
P10	69	42	36	15	26	54	14	46	55	112	40	37	17	87	22	45	70	34	35	27	22
P11	49	43	23	34	13	42	35	40	42	80	44	22	16	57	15	20	56	36	17	21	22
P12	66	109	50	24	19	40	42	58	187	176	11	61	47	95	57	53	79	42	15	43	
P13	182	65	52	9	34	88	7	39		126	14	109	34	39	25	48	54	68	15	24	59
P14	75	69	50	18	31	45	35	63	169	212	42	305	57	140	29	62	83	99	34	29	25
P15	94	131	56	13	35	45	47	53	137	99	22	62	21	73	26	60	84	76	46	41	81
P16	79	152	34	16	15	48	27	76	116	60	12	65	19	65		54	112	70	10	25	35
P17	50	150	60	25	20	37	13	29	30	42	16	32	29	49	16	32	33	22	9	23	48

Table 6.4: Task completion time in seconds by participant and task. Empty cells indicate non-responses to a task.

T/P	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21
P4	✗	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
P5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
P6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
P7	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
P8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
P9	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓
P10	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
P11	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
P12	✗	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
P13	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✗	✓	✓	✓	✓	✓	✓	✗	✓	✓
P14	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓
P15	✗	✓	✓	✗	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
P16	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
P17	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Sum																					
✓	9	11	13	13	13	13	14	14	13	9	14	12	14	14	12	13	14	14	9	14	13
✗	5	3	1	1	1	0	0	0	0	5	0	2	0	0	1	1	0	0	5	0	0

Table 6.5: Correctness of tasks for participants. Correct answers are indicated by a green checkmark. Incorrect answers are indicated by a red cross. Non-responses are indicated by an empty red cells. Totals of correct and incorrect answers for all participants are shown in the bottom two rows.

to identify variations among participants. Rather than focusing on absolute values, we analyzed general trends in task performance per topology, as each task within a topology varies due to the usage scenario as well as the kind of data about the topology that the task is based upon.

6.3.1 Speed of Task Performance

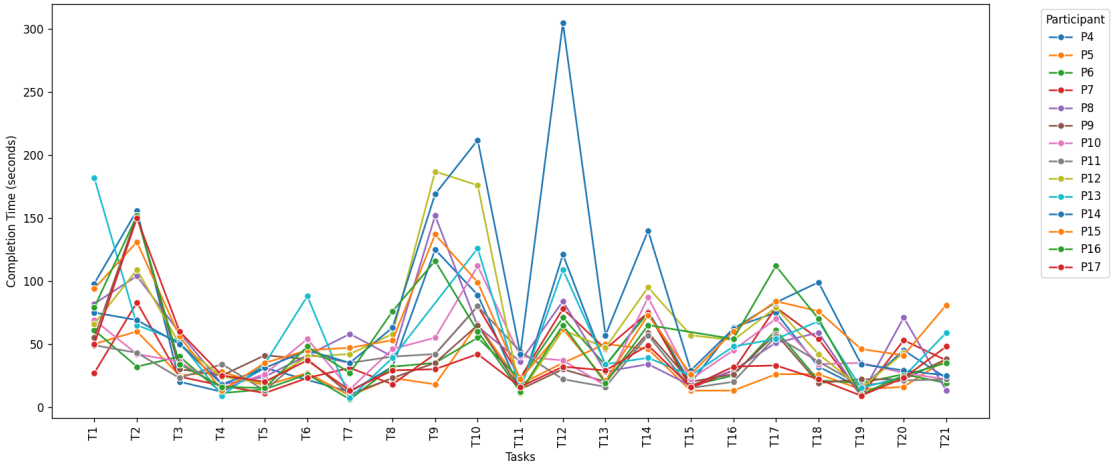


Figure 6.4: Completion time for participants across tasks.

Figure 6.4 displays a line chart of completion times for tasks T1–T21 across participants P4–P17. Tasks T1–T8 are from the first usage scenario and Tasks T9–T21 are from the second usage scenario. Our analysis of the speed of task performance also includes the completion times for incorrectly performed tasks. We observe that the trends in time taken by participants for each of the tasks follow a similar pattern with respect to the preceding and succeeding tasks, suggesting that the task completion times might not be random, and instead are influenced by the topology involved. We also see outliers in the task completion times, suggesting that some participants took comparatively longer time in completing some tasks.

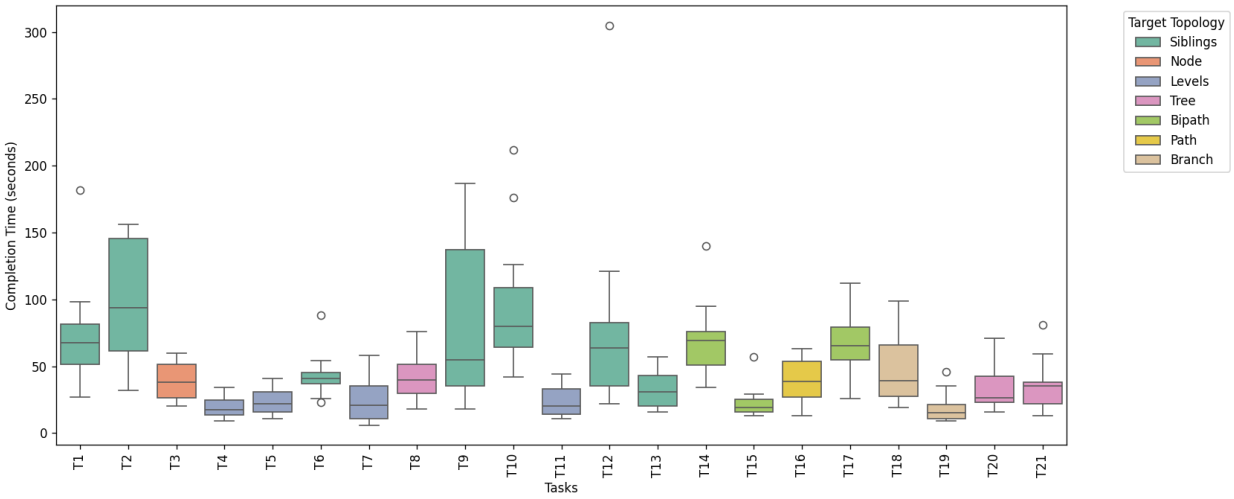


Figure 6.5: Distribution of completion times for each task.

Task	Count	Mean	Std Dev	Min	25%	50%	75%	Max
T1	14	74.07	36.52	27.00	51.25	67.50	81.25	182.00
T2	14	96.29	45.76	32.00	61.25	93.50	145.25	156.00
T3	14	39.93	14.37	20.00	26.25	38.00	51.50	60.00
T4	14	18.93	7.25	9.00	13.50	17.50	24.75	34.00
T5	14	23.64	9.49	11.00	15.50	22.00	31.00	41.00
T6	13	42.62	16.32	23.00	37.00	41.00	45.00	88.00
T7	14	24.71	16.87	6.00	10.50	20.50	35.00	58.00
T8	14	40.86	16.77	18.00	29.75	39.50	51.25	76.00
T9	13	87.92	60.70	18.00	35.00	55.00	137.00	187.00
T10	14	94.71	48.31	42.00	64.25	80.00	108.75	212.00
T11	14	23.36	11.97	11.00	14.00	20.00	32.75	44.00
T12	14	79.43	71.36	22.00	35.50	63.50	82.50	305.00
T13	14	32.14	13.35	16.00	20.25	31.00	43.00	57.00
T14	14	69.29	27.02	34.00	51.00	69.00	75.75	140.00
T15	13	22.38	11.46	13.00	16.00	19.00	25.00	57.00
T16	14	39.93	16.92	13.00	26.75	38.50	53.75	63.00
T17	14	65.79	22.21	26.00	54.50	65.50	79.00	112.00
T18	14	47.00	24.45	19.00	27.50	39.00	65.75	99.00
T19	14	19.21	11.36	9.00	11.00	15.00	21.25	46.00
T20	14	33.29	15.32	16.00	23.25	26.50	42.50	71.00
T21	13	35.38	18.72	13.00	22.00	35.00	38.00	81.00

Table 6.6: Descriptive statistics for task completion times across all participants

Task Completion Time

Figure 6.5 shows the distribution of task completion times (T1–T21) using boxplots, where each box captures how participants performed as a group on each task. The vertical axis represents completion time in seconds, while the horizontal axis lists the tasks. Each task is color-coded by its Target Topology for easy comparison. The height of each box represents the interquartile range (IQR), showing how spread out the middle 50% of times are. The bold horizontal line inside the box marks the median, indicating the typical completion time. Whiskers extend to capture most values within 1.5 times of the IQR, while outliers (dots) highlight cases where completion times were significantly different from the rest. Detailed results are provided in Table 6.6.

The tasks in Figure 6.5 are ordered in the sequence in which they appear for each individual topology. For topologies involving multiple tasks, we see a mix of increasing and decreasing trends, suggesting that there are no obvious learning effects. For instance, we see that task completion times for T1 and T2 involving siblings topologies are increasing, whereas task completion time for T12 to T13 involving siblings topology are decreasing. We see a similar mix of increasing trend for tasks T4 to T5 and decreasing trend for tasks T7 to T11 involving level topologies. We observe similar trends in tasks involving bi-Path topologies. These observations surface no obvious pattern in task completion times for tasks based on topology.

Some tasks on sibling topologies (T2, T9) exhibit relatively high median completion times and wide interquartile ranges, indicating that these tasks required more time and showed greater variability among participants. These tasks are based on inspecting the color channel of siblings topologies as well as manipulating the order of siblings at multiple levels, and are the beginning tasks in each usage scenario. Tasks involving inspecting the structure of a topology, such as counting the number of levels and counting the length of a path (T4, T15, and T19) have lower completion times with smaller interquartile ranges, suggesting that these tasks are more direct and that most participants thus completed them more quickly

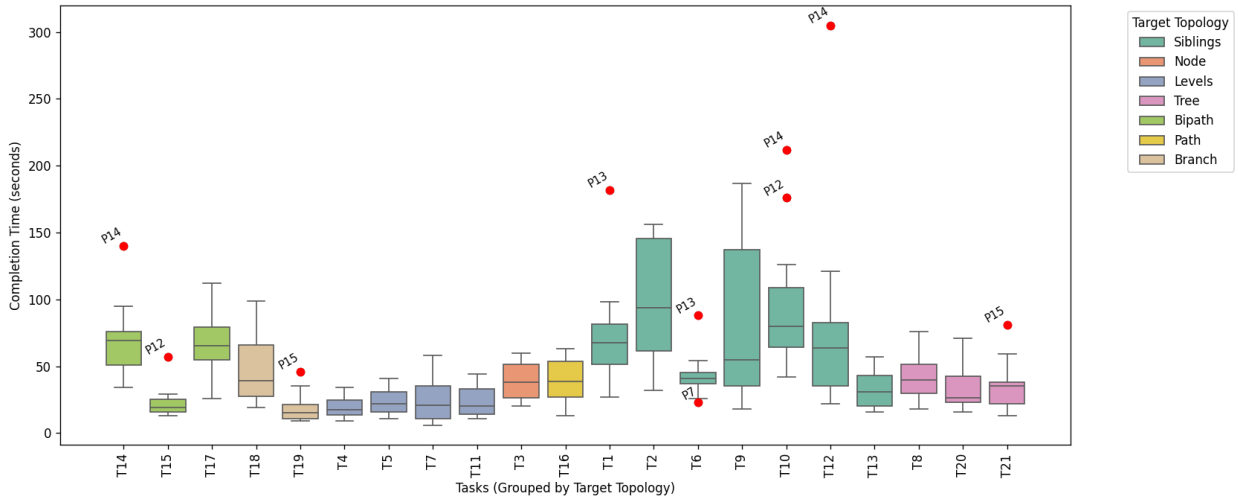


Figure 6.6: Distribution of completion times per task, colored by topology, with participant outliers labeled.

and with little variation. These tasks are based on inspecting the explicit representation of the topology for siblings and levels.

The higher standard deviation for tasks involving comparing of sibling colors (T2 and T10) and manipulating sibling structure (T9, T12) indicates that participants struggled for these specific kinds of tasks more than the others. Conversely, the tasks with lower standard deviation, as seen in tasks on levels (T4, T5, and T19), and on paths (T15) indicate that completion times for tasks on these topologies is faster to perform and consistent across participants. We also observe that the tasks with higher standard deviations are those tasks that involve multiple steps. These composite tasks with multiple steps also have skewed distributions, with some participants taking too long and other completing very quickly showing the variability of the tasks.

Figure 6.6 shows box plots of task completion time distributions grouped by topology. The mean completion times indicate a trend of decreasing completion time for some topologies. For example, a decreasing trend is observed from T14 to T15 and from T18 to T19. There is also a similar relationship between T14 and T15, and between T18 and T19. This relationship is not surprising as participant tend to perform a task that related to a previously

performed task perform much faster than otherwise. Similarly, completion times decrease from T2 to T6, and also decline from T12 to T13. Topology dependency is observed on tasks involving the color of siblings (T2 to T6) on manipulating siblings (T12 to T13). Increasing trends are also observed on tasks involving some topologies particularly paths (T15 to T17).

Outlier Analysis

Figure 6.6 shows outliers for each task. Outlier analysis indicates that among the seven identified outliers in task completion time, participant P12 appeared twice, P14 three times, P15 twice, P13 twice, and P7 once. This suggests that a few participants took significantly longer than others. Often affected tasks, T1 involves counting siblings that are closely and circularly spread, while T6 and T10 require distinguishing color differences. T12, T14, T15, T19, and T21 involve searching for a specific node in the tree before performing the task. For T1, participants often had to count siblings multiple times due to difficulty in keeping track of the starting node in a circular layout. In tasks involving color differentiation, the use of a linear color scale made it challenging to detect subtle variations in quantitative value. In tasks requiring node searching, participants displayed high variability in search strategies, often missing the target node on their first attempt. These challenges appear to stem from tree visualization designs of specific view types and are not specific to explicit representation of composite topologies. Further participant feedback is needed to assess learnability.

Correlation Analysis

Figure 6.7 presents the correlation matrix for task completion times after removing outliers. A correlation value close to +1 (red) indicates that participants who took longer on one task also tended to take longer on the other, while values near -1 (blue) suggest an inverse relationship, in which more time spent on one task corresponds to less time spent on another. Correlation values near 0 indicate no clear linear relationship between tasks. While some strong positive correlations appear, such as between T1 and T9, T12, and T16, further

inspection reveals many false positives, suggesting that these tasks are largely unrelated. The same is true with negative correlations. Overall, the correlation analysis did not yield meaningful insights.

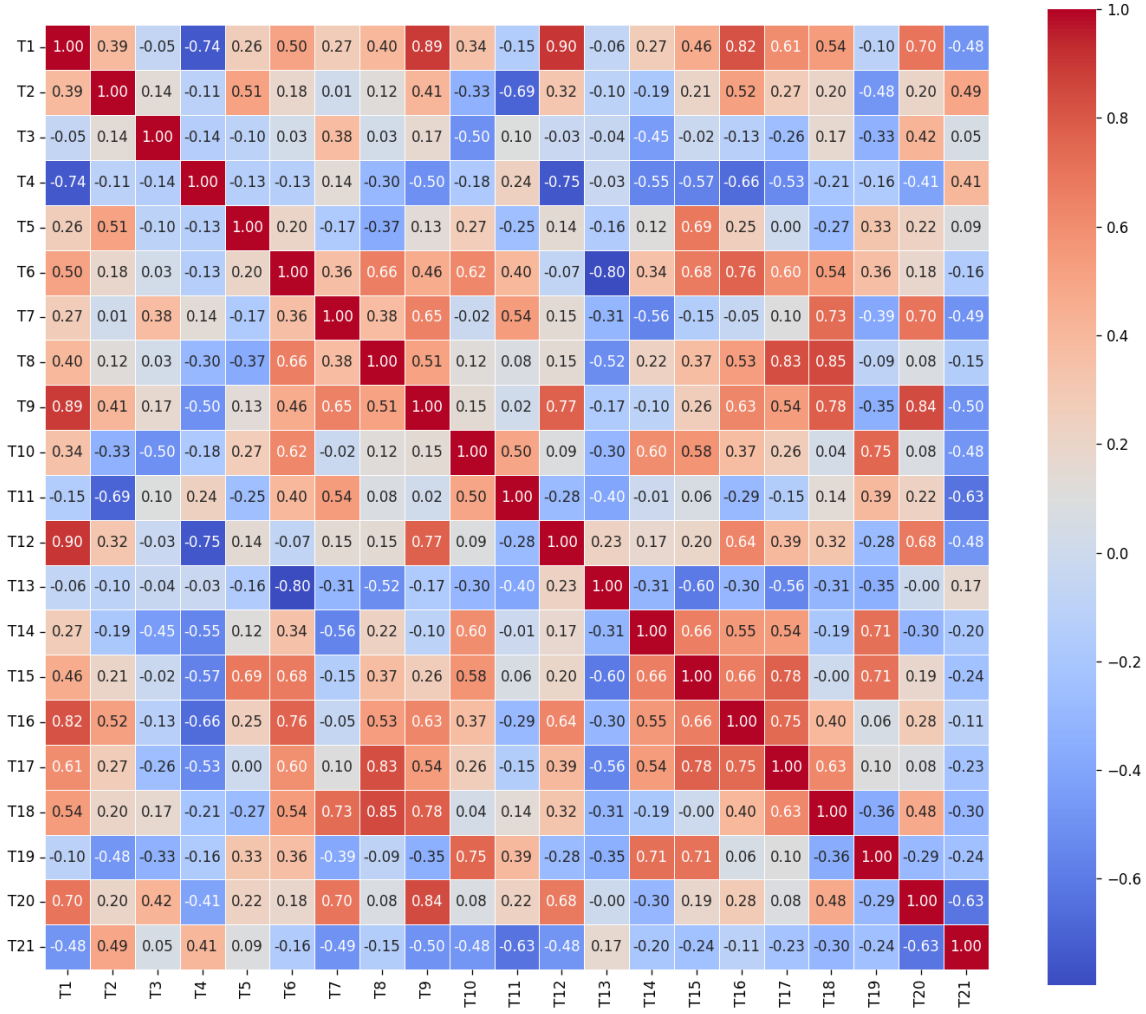


Figure 6.7: Correlation matrix of task completion times, with outliers removed.

Topology - Type	Tasks	Accuracy	Error Rate
Node-attribute	T3	$\frac{13}{14} = 0.929$	$\frac{1}{14} = 0.071$
Siblings-structure	T1,T9,T12,T13	$\frac{48}{56} = 0.857$	$\frac{8}{56} = 0.143$
Siblings-attribute	T2,T6,T10	$\frac{33}{42} = 0.786$	$\frac{9}{42} = 0.214$
Path-structure	T16	$\frac{13}{14} = 0.929$	$\frac{1}{14} = 0.071$
Bi-path-structure	T14,T15,T17	$\frac{40}{42} = 0.952$	$\frac{2}{42} = 0.048$
Levels-structure	T4,T7,T11	$\frac{41}{42} = 0.976$	$\frac{1}{42} = 0.024$
Levels-attribute	T5	$\frac{13}{14} = 0.929$	$\frac{1}{14} = 0.071$
Branch-structure	T19	$\frac{9}{14} = 0.643$	$\frac{5}{14} = 0.357$
Branch-attribute	T18	$\frac{14}{14} = 1.000$	$\frac{0}{14} = 0.000$
Tree-structure	T8, T21	$\frac{27}{28} = 0.964$	$\frac{1}{28} = 0.036$
Tree-attribute	T20	$\frac{14}{14} = 1.000$	$\frac{0}{14} = 0.000$

Table 6.7: Accuracy and error rate per topology and information type.

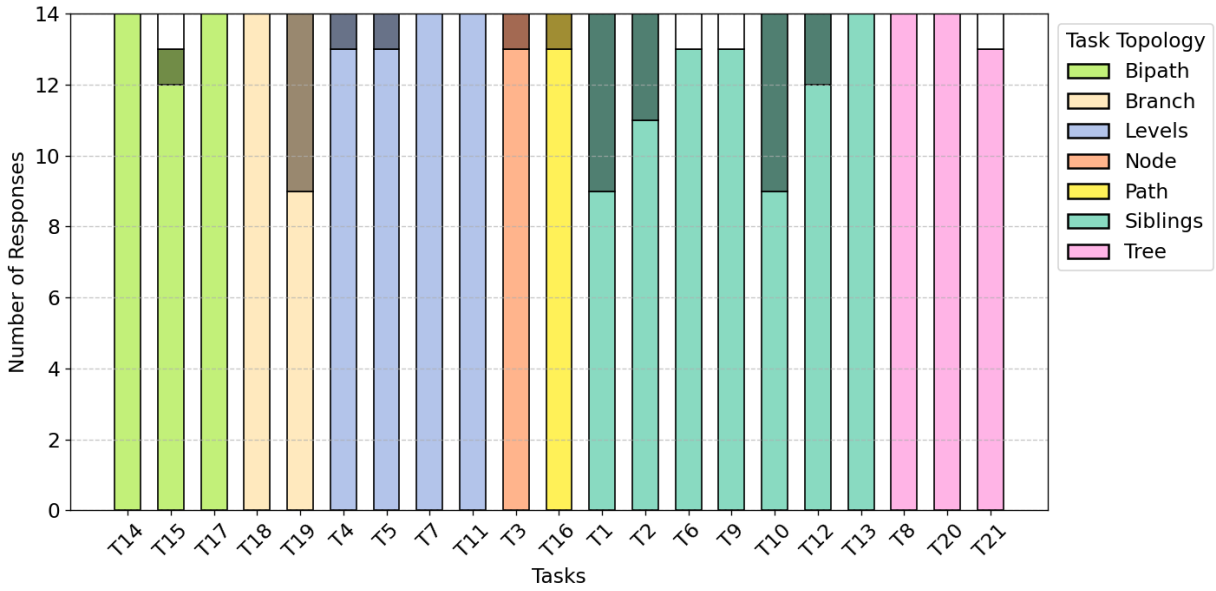


Figure 6.8: Responses per task, with correct (light), incorrect (dark) and non-responses (white) colored by topology.

6.3.2 Task Correctness

Correctness provides an estimate of how accurately participants perform different types of tasks. Some tasks for a topology involve structure information, and others involve attribute information. To assess correctness, we sub-categorize the topologies on these two information types, as shown in Table 6.2 with the accuracy and error rate for each topology-data type pair.

Figure 6.8 shows the count of correct responses, the count of incorrect responses, and the count of non-responses for each task. Correctness scores are binary: a score of 1 is assigned for each correct response, while both incorrect and non-responses receive a score of 0. Table 6.7 presents the accuracy and error rate of participants' responses, categorized by task topology and data type. The accuracy for each category is computed using a formula to calculate the proportion of correctly answered tasks within each topology. A higher accuracy value indicates that participants performed well in those task categories.

$$\text{Accuracy} = \frac{\text{Correct Responses}}{\text{Total Responses (Correct + Incorrect + Non-Responses)}} \tag{6.1}$$

Similarly, the error rate is calculated as:

$$\text{Error Rate} = \frac{\text{Incorrect Responses + Non-Responses}}{\text{Total Responses}} \tag{6.2}$$

This value reflects the proportion of responses that were either incorrect or left unanswered. A higher error rate suggests that participants found tasks in that category more challenging.

In Table 6.7, we observe that the *Branch-attribute* category has the highest accuracy of 1.000, indicating that all responses were correct. Conversely, the *Branch-structure* category exhibits a relatively low accuracy of 0.643 and the highest error rate of 0.357, suggesting increased difficulty in these tasks. The task for that category, task 19 involves calculating the number of levels in a branch. Qualitative feedback indicates that participants either mistakenly counted the gaps between the levels rather than the levels themselves, or filtered

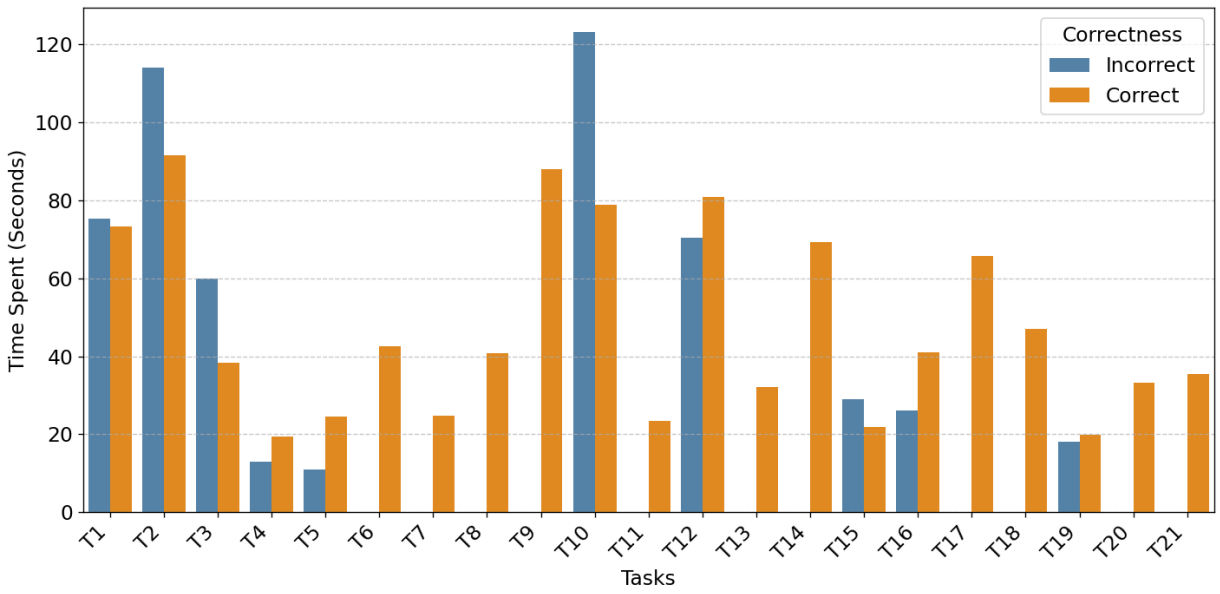


Figure 6.9: Grouped bar chart of average task time and correctness for each task.

the tree and inadvertently hid the last level, leading them to miscalculate the levels.

For the *Siblings-attribute* category, the accuracy is relatively low at 0.786, while *Siblings-structure* exhibits the third-lowest accuracy at 0.857. These attribute-related tasks relied on encoding numerical attributes as color values. Specifically, sibling colors were assigned based on either the number of siblings (T2) or the average age of siblings (T10). Participants struggled to differentiate subtle variations in color, leading to errors. Regarding *Siblings-structure*, most errors occurred in T1, which required counting the number of siblings within a level in a radial node-link tree. Participant feedback indicates that errors primarily stemmed from overlooking siblings whose visual representation was too narrow in terms of arc length, making them difficult to identify. For all other tasks, participants achieved an average accuracy of over 92%.

6.3.3 Task Completion Time versus Task Correctness

Figure 6.9 presents the average time taken by participants for both correct and incorrect task attempts. The orange bars in the grouped bar chart represent the average time spent

(in seconds) on correctly answered tasks across participants, while the adjacent blue bars represent the time spent on incorrectly answered tasks. Missing blue bars for certain tasks indicate that all participants answered those tasks correctly. Below are some key observations regarding the relationship between task correctness and the average time spent for individual tasks.

Longest Time-Consuming Tasks Figure 6.9 highlights that tasks T2 and T10 were the most time-consuming when answered incorrectly. This suggests that participants not only made more errors on these tasks, as indicated in Table 6.7, but also required significantly more time to complete them. These findings indicate that using color to represent quantitative information may not be ideal, which aligns with Mackinlays [77] ranking of visual channels, where color is considered less effective for encoding quantitative data. The results also suggest that the tasks were sufficiently challenging to cause participants to spend considerably more time on them compared to tasks answered correctly.

Shortest Time-Consuming Tasks Figure 6.9 highlights that tasks T4 and T19 were among the least time-consuming when answered incorrectly. Both tasks are associated with the structure attribute of the topology level, requiring participants to evaluate how many levels a particular branch or the entire tree spans. Although participants completed these tasks quickly, a closer look at Figure 6.8 reveals an important distinction. Task T4, which involved counting the levels of the entire tree, was performed with high accuracy (13 out of 14 participants, or 92.8%). In contrast, Task T19, which required calculating the levels of a specific branch, was completed with a lower accuracy of 64.1%.

Analysis of screen recordings and participant feedback indicates that while participants found it easy to count the levels, users often filtered the tree during exploration and unintentionally hid the lower levels, resulting in inaccurate interpretation of the tree.

Tasks Without Errors The figure highlights that participants were able to successfully complete 11 out of 21 tasks (T6, T7, T8, T9, T11, T13, T14, T17, T18, T20, T21) without any errors. Among these, the majority of tasks, i.e., 9 (T6, T7, T8, T9, T11, T13, T14, T17, T21), are structure-related tasks, while T18 and T20 are attribute-related tasks on topologies. These results suggest that participants made no errors for the answered questions for more than 50% of the tasks overall, and within the correctly answered tasks, participants were mostly able to answer structure-related tasks on composite structures more accurately and consistently compared to attribute-related tasks on composite topologies. The task completion times for these correctly answered tasks generally fall between the longest and shortest duration observed across all tasks.

Average Response Completion Time vs Accuracy Table 6.5 shows the correct and incorrect answers by the participants, while Table 6.6 shows the mean completion time of the above tasks, both correctly and incorrectly. Our results showed that different tasks with the same accuracy often had different mean completion times. For instance, tasks T1, T10, and T19 all have an accuracy of 64.2%, with 9 participants answering correctly out of 14 for each task, while having mean completion times of 74.07, 94.71, and 19.21 seconds, respectively. Analysis of the tasks showed that T1, T10, and T19 are not tasks of the same type, as shown in Table 6.7, and are varied in nature. T1 focuses on counting the siblings, T10 focuses on identifying the attribute of the sibling, and T19 focuses on identifying the levels in a branch structure. The steps involved in completing the tasks are different, resulting in varied mean completion times, although they have the same accuracy.

The tasks that were answered correctly with an accuracy of 100% also exhibited different mean completion times. Table 6.7 shows that even when tasks are of same type, and are completed with same 100% accuracy, their mean completion times can vary significantly. For example, in sibling-structure tasks, T9 and T13 both achieved 100% accuracy, yet their mean completion times were 87 seconds and 32 seconds, respectively. Similarly, in Bi-path

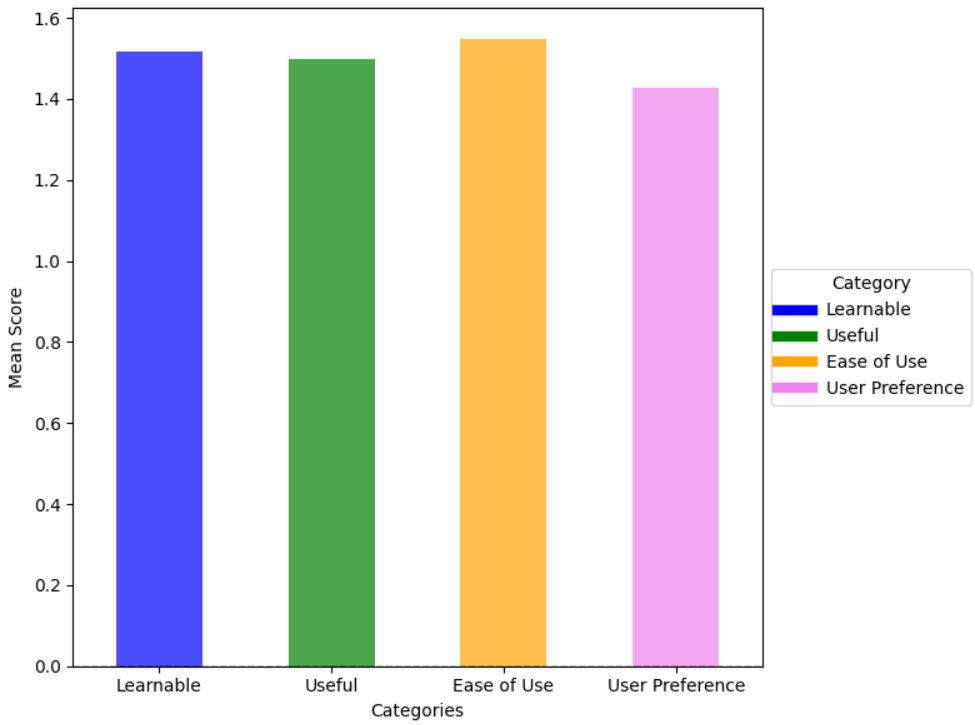


Figure 6.10: Mean score per category.

structure tasks, T15 and T17 had mean completion times of 22 seconds and 65 seconds, despite identical accuracy. This indicates that task completion times can differ notably even for tasks of the same nature and accuracy.

Overall, the above results suggest that the average completion time of tasks and accuracy may not be directly related to each other. Tasks on tree vary in the number of steps involved and may be influenced by many factors such as the scale of data in the domain as was seen in two usage scenarios, and the task and specific topology in particular.

6.4 Qualitative Data Analysis

We administered a post-survey, asking participants to rate the experience of working with composite topologies on three aspects, as shown in Table 6.3.

Average Likert ratings were high all around on a 5-point scale with ranging from -2 to +2: easy to learn (mean = 1.51, s.d.= 0.39), useful (mean = 1.50, s.d.= 0.48), easy to use

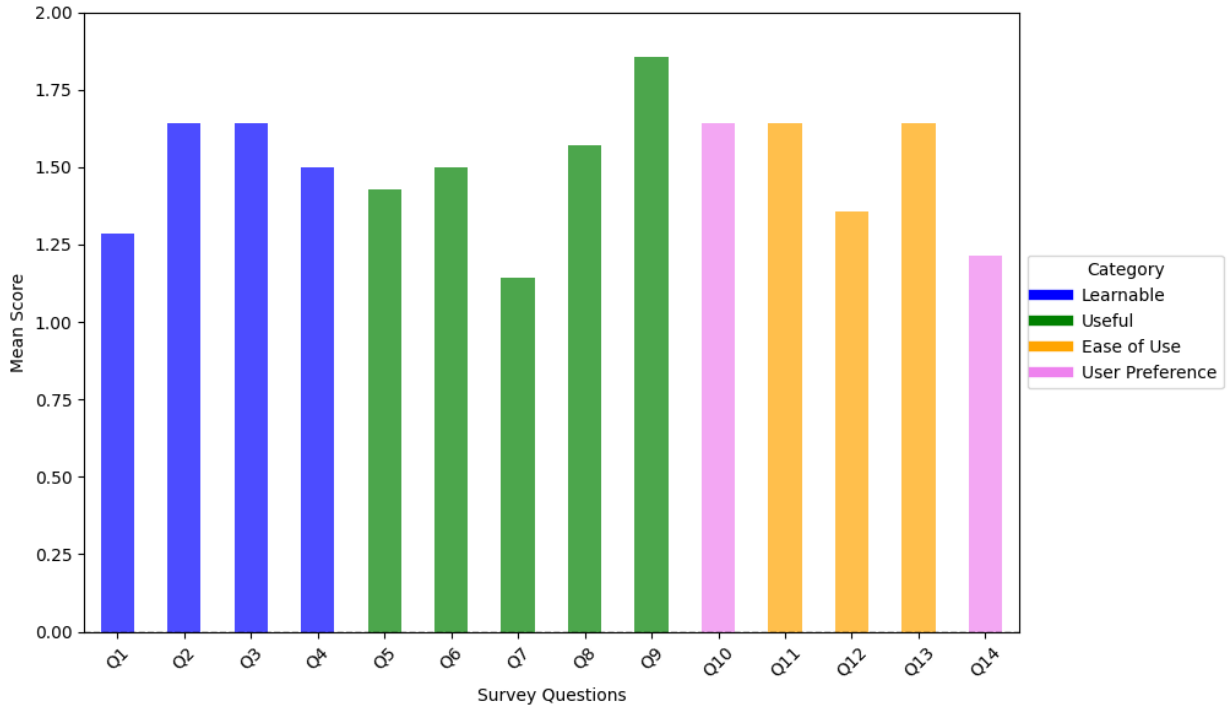


Figure 6.11: Mean score per question.

(1.54, s.d.= 0.40) and, user preference (mean = 1.42, 0.29). The feedback is encouraging and suggests that users found the topology-augmented tree visualization to be easy to learn and easy to use. They also found the visualizations useful in their data analysis and expressed a preference for explicit representation of composite topologies.

Participants generally found the tree visualizations useful for understanding hierarchical data. P16 noted, "Controls for filtering individual topologies to show only topologies of interest is helpful in the data analysis. Also filtering the number of levels to show is also helpful in understanding the data." Similarly, P14 remarked, "I used the controls to hide the nodes with which I was able to access the sibling sets better," and P3 stated, "The controls help with de-cluttering by hiding or showing the different levels, making the information easier to read and understand."

When differences in the sizes of sibling sets were high, participants appreciated the use of color to help distinguish them. P11 commented, "Color of the siblings helped me the most in discovering the difference in the number of nodes in the sibling." Many participants were

able to effectively identify composite topologies; P6 commented, "By using the color coded siblings its easy to understand that L3 has less nodes."

Some participants did find the visualizations challenging, particularly when using a less effective color encoding channel [77] for displaying quantitative data. P17 mentioned, "When the quantitative value for the siblings is low, it is hard to distinguish differences between the siblings." In addition, participants observed that overlapping visual channels sometimes interfered with data interpretation; for instance, P16 said, "When multiple sibling sets of similar color are very close to each other, they appeared as a single sibling set." In the case of the radial node-link tree, the spatial spread of nodes also affected perception. P15 noted, "Spread of nodes makes level 2 appear bigger compared to level 3 even though it is not." Learning the interface required some time. P14 remarked, "It took a while to understand the controls and how to navigate the tool," and P4 added, "It takes time for me to learn the interface and keep it in my mind." Additionally, participants provided valuable feedback for improvement. P11 suggested that filtering sometimes obscures the context, making it difficult to determine the current level, emphasizing a need for better contextual cues. Overall, in the open-ended questions in Usage Scenario 3, participants indicated a mix of positive and negative sentiment in their responses to questions about the effectiveness of design choices for representing certain kinds of topological information.

6.5 Assessment of Results

Evaluation of the tree visualizations developed using Hieros provides valuable insights about the usability and utility of representing topologies as first-class objects in tree visualizations. In this section, we break down our findings on quantitative performance and qualitative feedback, and identify challenges and areas for improvement. We also do an assessment of the user evaluation approach and suggest future directions for empirical studies.

6.5.1 Quantitative Performance

Task Accuracy and Completion Times

Participants achieved reasonably high accuracy and lower completion times on tasks while assessing structure of the composite topologies, and lower accuracy and longer completion times on tasks involving interpretation of quantitative information about topologies, especially with tasks that require distinguishing subtle quantitative differences. The challenge of explicitly representing composite topologies in tree visualizations is due to the limited available channels for encoding information, particularly quantities. The layout of some tree visualization such as the node-link visualizations, limits the use of the more effective channels [77] of length and area for encoding quantitative attributes. Designers may need to resort to less effective channels for quantities such as such as color, texture, or density. This reflects a general challenge in visualization, not one specific to representing quantitative information about composite topologies. Participants also achieved longer completion times on tasks on paths and siblings that involves finding a node as a first step in completing the task. Our observations suggest that searching for a node in a tree by name is equivalent to searching every node in a tree which likely contributes a correspondingly longer time to complete such tasks. Manipulation tasks on siblings topologies involving multiple steps also resulted in longer completion times, suggesting difficulty in task performance. Despite moderate variability in task completion times, participants achieved an average accuracy of 90% across all tasks and an average mean completion time of well under two minutes for the most complicated composite structure manipulation task tested.

6.5.2 Qualitative Feedback

Ease of Use and Learnability

Participants in general provided positive feedback regarding usability. Comments such as "the controls for filtering individual topologies were helpful" and "I can see the relationships

of each node and its siblings clearly" highlight the ability of participants to explicitly see the composite topologies and control their appearance, indicating that they enhance clarity of information. Participants also found the visualizations *easy to use* with a mean of 1.54, between strongly agree and agree. Participants also found the visualizations *learnable* with a mean of 1.51, between very confident and somewhat confident in identifying composite topologies, interacting with them, and quickly getting an overview of the hierarchy by looking at the composite topologies that make it up.

Suitability

Participants appreciated the ability to interact with hierarchical topologies directly. The option to toggle the visibility of different topologies, such as branches and levels, allowed them to focus on the information most relevant to the task at hand. This direct interaction with topologies rather than relying solely on node level manipulation was seen as particularly valuable for tasks involving comparative analysis of topologies as well as structural analysis of topologies. The open-ended exploration phase further confirmed that participants could derive meaningful insights by leveraging these explicit representations. The kinds of tasks that we tested are representative of the varieties of tasks one could perform on topologies, and are not exhaustive. For instance, some domains require representing arbitrary clusters of nodes in a hierarchy, which is not currently supported by the Hieros library. However, being able to represent common topologies provides the necessary path and direction towards supporting new and specialized topologies in different domains. The qualitative feedback on the utility had a mean value of 1.50 with standard deviation of 0.48, indicating that participants characterized utility between Extremely Useful and Very Useful.

User Preference

Responses to the survey questions about appeal and user preference indicates that suggested that participants prefer to see composite topologies in the tree visualizations during data

analysis and they do not feel that explicitly representing composite topologies reduces the appeal of the visualizations. The qualitative feedback on user preference and appeal suggested a mean value of 1.42 with standard deviation of 0.29, indicating that participant's sentiment of these aspects of usability is between strongly agree and agree.

6.5.3 Overall Assessment of Usability and Utility

Feedback on usability from the participants indicates that participants found the topology-augmented tree visualizations easy to use, and they preferred working with tree visualizations that explicitly represent topologies. The task performance metrics for accuracy and task completion time showed that participants were able to complete tasks on composite topologies with an average accuracy of 90% across all tasks and an average mean completion time of 96.29 seconds for the most complicated composite structure manipulation task tested.

Feedback on utility indicates that participants found the explicit representation of composite topologies and supporting interactive operations on them useful for data analysis and exploration tasks. Responses to the survey questions indicate that participants had both positive and negative sentiments about the effectiveness of performing tasks on some composite topologies, suggesting a need to improve the visual representations of these topologies and interactions to perform tasks on them.

6.6 Challenges and Areas for Improvement

6.6.1 Interface Complexity and Cognitive Load

While explicit representation of topologies might make it easier to perform tasks on composite topologies, the added complexity of interacting with multiple topologies might initially overwhelm users. Participants, however, did not provide any such comments. More research is needed to understand challenges involved in transitioning from simple node and edge based

interactions to complex composite structural interactions.

6.6.2 Assessment of Scalability

Some participants preferred to use interactions like "zooming" and "panning" across the visualization in screen space while performing the tasks on topologies, as opposed to filtering the data and navigating the tree using topologies. There may be many reasons for this. One reason might be familiarity with such features. Another reason might be the need to remember new and added interactions to navigate the hierarchy. Overall, effectively using filtering of composite topologies may need prior training. Approaches to learning topology interactions are another area for future research.

6.6.3 Next Steps for User Evaluation

The evaluation indicated cross-talk influences for two tasks; there is a strong possibility of learning effects when performing tasks on similar topologies. As users become more familiar with topologies through task performance, refinement of the evaluation methodology, such as by implementing counterbalancing techniques or providing extended training until task performance speeds converge, could help mitigate these effects.

6.7 Summary

This chapter describes three usage scenarios for evaluating the utility and usability of example visualizations created with the Hieros library. The evaluation demonstrated that representing composite topologies explicitly enhances the exploration and analysis of hierarchical data by supporting various kinds of tasks directly on those representations. We also found evidence for the utility and usability of the topology-augmented hierarchical visualizations. Nevertheless, there are opportunities for improvement, particularly in usability when performing complex operations involving multiple steps. This motivates exploration of

alternative tree visualization designs to augment for ongoing development and study. Future research involving direct comparison of quantitative metrics could offer evidence on comparative usability and utility to guide the selection of tree visualization techniques and how they explicitly show topologies to support specific data analysis needs and goals.

Chapter 7

Conclusion and Future Work

7.1 Overview

In this dissertation, we address the challenge of explicitly representing *composite topologies* in tree visualization, motivated by the observation that current tree visualizations often overlook such explicit representation and support for operations on them. Our goal was to explore how topology-augmented tree visualizations can support operations on tree visualizations for the purposes of data analysis and exploration. This concluding chapter summarizes our main findings, relates them back to the research questions posed in Chapter 1, highlights key contributions, discusses limitations, and suggests future directions for extending the approach.

7.2 Contributions

This dissertation presents the following thesis: *Tree visualization techniques can be augmented to explicitly represent composite topological structures and enable direct interaction with those structures. Explicit representation and interaction with composite topologies visually emphasizes their intrinsic structure and relationships, thereby supporting faster and more effective exploration and analysis of hierarchical data than by visualizing and interact-*

ing with those structures solely through their constituent node and edge representations. The research described in this dissertation makes the following contributions:

- a JavaScript coordination library for building coordinated multiple view using D3.js [24] along with two fully implemented coordinated multiple view examples;
- a conceptual model of *representational suitability* to assess the suitability of various tree representation techniques in their ability to represent various aspects of topologies;
- an implementation of a tree visualization library, Hieros, for the Improvise visualization system, capable of representing various topologies in hierarchical visualizations as well as supporting interactions on topologies;
- a software architecture and hierarchical data pipeline to support representing topologies in tree visualizations as well as supporting a variety of operations at various stages of the visualization pipeline;
- seven variants of popular tree visualization techniques with layout algorithms capable of representing composite topologies and interactions on those topologies; and
- an evaluation of the utility and usability of example visualizations created with the Hieros visualization library and its architecture for creating topology-augmented tree visualizations for tree analysis tasks involving topologies.

Topology-augmented tree visualization opens new avenues for systematically studying hierarchical data visualization. The capabilities of the data pipeline and the example tree visualization techniques are beneficial for studying a much expanded design space of tree visualizations as well as developing new methodologies and evaluation methods for assessing tree visualizations. This research could ultimately benefit designers in developing new kinds of effective tree visualization techniques to support the data analysis tasks and processes of end users.

7.3 Publications

The following publications describe the research results of this dissertation:

- Chekuri, O. S. and Weaver, C. (2025). C4D3: A View-Level Abstraction and Coordination Library for Building Coordinated Multiple Views with D3. In Proceedings of the 20th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - IVAPP [**FullPaper - Published**] [31]

- Chekuri, O. S. and Weaver, C. An Investigation into the Representational Suitability of Tree Visualizations. IEEE Visualization Conference Poster, 2023. [**Extended Abstract and Poster - Published**] [32]

- Chekuri, O. S. and Weaver, C. A Model of the Representational Suitability of Tree Visualizations. [**Full Paper In Revision**]

- Chekuri, O. S. and Weaver, C. Towards Explicitly Representing Topologies in Tree Visualizations. [**Short Paper - Submitted to IEEE Vis 2025**]

- Chekuri, O. S. and Weaver, C. Hieros: A Software Architecture for Topology-Augmented Hierarchy Visualization. [**Full Paper - In Preparation**]

7.4 Revisiting the Research Questions

Suitability of Tree Representations for Representing Information about Topologies. We developed a conceptual model of representational suitability for assessing the suitability and ability of tree visualizations for representing various kinds of information about topologies. We also showed that different kinds of tree visualizations demonstrate varying levels of support for representing various kinds of information across topologies. (See Chapter 3.)

Supporting Coordination in Client-Side Visualization Libraries. We demonstrated that the C4D3 library provides a coordination abstraction and a view abstraction that together enable *declarative* specification of coordination between multiple views to support a variety of coordination types and patterns [140]. These capabilities facilitate the development of complex, multi-view visualization applications. (See Chapter 4.)

Topological Operations in Tree Visualizations. We have shown various kinds of information that composite topologies can represent in tree visualizations to support perceptual operations in the representational suitability model (Chapter 3), and discussed examples of operations one can perform on tree topologies at various stages of the visualization pipeline. (See Chapter 5.)

Software Architecture for Topology Support. We developed a data pipeline for hierarchical data and a software architecture that treats topologies as first-class objects with support for interactive operations on topologies at various stages of the information visualization pipeline. These operations include generating the necessary topology objects such as paths and branches, setting a node to view as the root of the tree, interacting with the rendered topologies such as by selecting them, modifying the topologies (such as to change the order of nodes in a siblings topology), and changing the layout. We also extended the coordination architecture of Improvise [138] to support coordination of topologies across tree visualizations. (See Chapter 5.)

Designing Visualizations and Interactions for Topologies. Seven prototype visualizations represent topologies explicitly. We selected a representative sample of tree visualizations and studied how the topology-augmented versions of those visualizations appear and behave. We designed how widget-based and direct manipulation interactions work with the composite topologies to perform various topological operations. We then designed how the state of interaction appears for various operations, including how intermediate interaction

states appear for complex operations involving multiple steps. (See Chapter 5.)

Benefits and Drawbacks of a Topology-Augmented Tree Visualizations. User evaluation results showed that extending existing tree visualization techniques to support explicit representation of composite topologies confers benefits for seeing and working with the structural and attribute information of the elements in exploration and analysis. Ongoing research challenges include studying the effectiveness of the explicit representation of topologies for representing different kinds of information and the effectiveness of interactions for acting upon that information through their topological representations. The evaluation results also suggest directions for improving the user evaluation methodology for assessing topology-augmented tree visualizations. (See Chapter 6.)

7.5 Future Work

The research described in this dissertation also opens up wide range of new research questions and directions worth pursuing.

Interoperability of C4D3 with Other Visualization Libraries. C4D3's view abstraction currently supports the D3.js [24] library and ecosystem for creating coordinated multiple views. We plan to extend C4D3's support to other libraries such as Vega-Lite [108] and Plotly.js [3] to broaden adoption. Additionally, we aim to develop a bridging version of C4D3 that works with front-end JavaScript libraries like React.js [1] and Vue.js [2].

Supporting the Visualization Data Processing Pipeline for C4D3. We aim to provide access to the full visualization data processing pipeline, starting with dynamic queries [7] and progressing through re-implementation of the full Coordinated Queries architecture in Improvise [138].

Designer-Centric Evaluation of the Representative Suitability Model. Through scenario-based assessment, we were able to demonstrate the utility of the representational suitability model, for choosing appropriate tree visualizations. We would like to gain deeper insight into how designers use the model in practice by conducting a designer-centric user evaluation. We also plan to extend the model and assess how it works for topology-augmented tree visualizations.

Interaction Model for Supporting Topological Operations. Moving forward, we plan to develop an interaction suitability model for tree visualizations, similar to the representational suitability model to assess the suitability of different interactions on explicit topology representations for editing information directly within tree visualizations. The insights gained could inform the design of user studies focused on advanced tree visualization analysis tasks and processes.

Coordination Patterns in Tree Visualizations. We plan to further explore support for existing coordination patterns [140], in addition to the ones developed in Chapter 5, that make use of composite topologies in tree visualizations, investigate novel patterns, and study their impact on tree visualization design and analysis.

Supporting a Wide Variety of Tasks and Operations on Trees. We have demonstrated the ability of the Hieros library to support a variety of tasks at different stages of the visualization pipeline. Moving forward, our goal is to expose the hierarchical data pipeline at the designer level, allowing for declarative specification of operations.

Beyond Static Tree Visualizations for Hieros. We have implemented the Hieros data pipeline and visualization techniques for statically sourced tree data. We plan to study ways to modify the software architecture and data pipeline to handle dynamic data that involves evolving hierarchies. We plan to assess the capabilities and limitations of existing

tree visualization techniques and interaction methods in dealing with dynamic data and improve upon them to support such dynamics in data analysis processes.

7.6 Conclusions

The primary aim of this dissertation was to develop a systematic understanding of the design space of explicit composite topologies in tree visualizations, beyond the usual nodes and edges, including options to represent and interact with topologies in support of exploration and analysis of hierarchical data. Through a combination of conceptual models, implemented visualization libraries, and a user evaluation, this work examined possibilities for composite topologies, developed support for topology-augmented visualization design, gathered evidence for the utility and usability of such topology-augmented visualizations in supporting a variety of operations directly on topologies, assessed shortcomings, and described future directions for ongoing research and development. We anticipate that this work will open up numerous directions in hierarchical visualization research, including the development of taxonomies, user evaluation strategies, new visualization techniques, and ultimately improved support for data analysis with tree visualizations.

Bibliography

[1] React: The library for web and native user interfaces. `https://react.dev`. Last accessed: June 18, 2025.

[2] Vue: The progressive JavaScript framework. `https://vuejs.org`. Last accessed: June 18, 2025.

[3] Plotly Javascript open source graphing library. `https://plotly.com/javascript`. Last accessed: June 18, 2025.

[4] Andrea Adamoli and Matthias Hauswirth. Trevis: A context tree visualization & analysis framework and its use for classifying performance failure reports. In *Proceedings of the 5th International Symposium on Software Visualization*, SOFTVIS '10, pages 73–82, New York, NY, USA, October 2010. Association for Computing Machinery.

[5] Dimitris K. Agrafiotis, Deepak Bandyopadhyay, and Michael Farnum. Radial Clustergrams: Visualizing the Aggregate Properties of Hierarchical Clusters. *Journal of Chemical Information and Modeling*, 47(1):69–75, January 2007.

[6] Christopher Ahlberg. Spotfire: An information exploration environment. *SIGMOD Rec.*, 25(4):25–29, December 1996.

[7] Christopher Ahlberg, Christopher Williamson, and Ben Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, pages 619–626, New York, NY, USA, June 1992. Association for Computing Machinery.

[8] Christopher Ahlberg and Erik Wistrand. IVEE: An information visualization & exploration environment. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*, pages 66–73, 142–143, Atlanta, GA, October 1995. IEEE.

[9] Alexander Aiken, Jolly Chen, Michael Stonebraker, and Allison Woodruff. Tioga-2: A Direct Manipulation Database Visualization Environment. In *Proceedings of the Twelfth International Conference on Data Engineering*, ICDE '96, pages 208–217, USA, February 1996. IEEE Computer Society.

[10] Ragaad AlTarawneh and Shah Rukh Humayoun. Visualizing Software Structures through Enhanced Interactive Sunburst Layout. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '16, pages 288–289, New York, NY, USA, June 2016. Association for Computing Machinery.

[11] Keith Andrews and Janka Kasanicka. A Comparative Study of Four Hierarchy Browsers using the Hierarchical Visualisation Testing Environment (HVTE). In *Proceedings of the 11th International Conference Information Visualization*, IV '07, pages 81–86, USA, July 2007. IEEE Computer Society.

[12] Keith Andrews, Wolfgang Kienreich, Vedran Sabol, Jutta Becker, Georg Droschl, Frank Kappe, Michael Granitzer, Peter Auer, and Klaus Tochtermann. The InfoSky visual explorer: Exploiting hierarchical structure and document similarities. *Information Visualization*, 1(3/4):166–181, December 2002.

[13] M. Balzer and O. Deussen. Hierarchy Based 3D Visualization of Large Software Structures. In *IEEE Visualization 2004*, pages 4p–4p, October 2004.

[14] M. Balzer and O. Deussen. Voronoi treemaps. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, pages 49–56. IEEE, 2005.

[15] Todd Barlow and Padraic Neville. A Comparison of 2-D Visualizations of Hierarchies. In *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, INFOVIS '01, page 131, USA, October 2001. IEEE Computer Society.

[16] Luc Beaudoin, Marc-Antoine Parent, and Louis C. Vroomen. Cheops: A compact explorer for complex hierarchies. In *Proceedings of the 7th Conference on Visualization '96*, VIS '96, pages 87–ff., Washington, DC, USA, October 1996. IEEE Computer Society Press.

[17] Michel Beaudouin-Lafon. Designing interaction, not interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, pages 15–22, New York, NY, USA, May 2004. Association for Computing Machinery.

[18] Richard A. Becker and William S. Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, May 1987.

[19] Benjamin B. Bederson, Ben Shneiderman, and Martin Wattenberg. Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. *ACM Trans. Graph.*, 21(4):833–854, October 2002.

[20] Jacques Bertin. *Semiology of Graphics*. University of Wisconsin Press, 1983.

[21] Robert P. Biuk-Aghai, Patrick Cheong-Iao Pang, and Bin Pang. Map-like visualisations vs. treemaps: An experimental comparison. In *Proceedings of the 10th International Symposium on Visual Information Communication and Interaction*, VINCI '17, pages 113–120. Association for Computing Machinery, 2017. event-place: Bangkok, Thailand.

[22] Marko Bohanec. DEXiTree: A program for pretty drawing of trees. *Proc. Information Society IS*, 2007:8–11, 2007. Ljubljana.

[23] Michael Bostock and Jeffrey Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1121–1128, 2009.

[24] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, December 2011.

[25] Nadia Boukhelifa and Peter J. Rodgers. A Model and Software System for Coordinated and Multiple Views in Exploratory Visualization. *Information Visualization*, 2(4):258–269, December 2003.

[26] Matthew Brehmer and Tamara Munzner. A Multi-Level Typology of Abstract Visualization Tasks. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2376–2385, December 2013.

[27] Mark Bruls, Kees Huizing, and Jarke J Van Wijk. Squarified treemaps. In *Data Visualization 2000: Proceedings of the Joint EUROGRAPHICS and IEEE TCVG Symposium on Visualization in Amsterdam, The Netherlands, May 29–30, 2000*, pages 33–42. Springer, 2000.

[28] Andreas Buja, Dianne Cook, and Deborah F. Swayne. Interactive High-Dimensional Data Visualization. *Journal of Computational and Graphical Statistics*, 5(1):78–99, 1996.

[29] Stuart K Card, Jock Mackinlay, and Ben Shneiderman. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[30] Sheelagh Carpendale. *Evaluating Information Visualizations*. Springer Berlin Heidelberg, 2008. Publication Title: Information Visualization: Human-Centered Issues and Perspectives.

[31] Omkar Chekuri and Chris Weaver. C4D3: A View-Level Abstraction and Coordination Library for Building Coordinated Multiple Views with D3. In *Proceedings of the 20th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 1: IVAPP*, pages 955–966. INSTICC, SciTePress, 2025.

[32] Omkar S Chekuri and Chris Weaver. An Investigation into the Representational Suitability of Tree Visualizations. IEEE Conference on Visualization and Visual Analytics (VIS), Extended Abstract and Poster, October 2002.

[33] Hong Chen. Compound brushing. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*, pages 181–188, Seattle, WA, October 2003. IEEE Computer Society.

[34] Ran Chen, Xinhuan Shu, Jiahui Chen, Di Weng, Junxiu Tang, Siwei Fu, and Yingcai Wu. Nebula: A coordinating grammar of graphics. *IEEE Transactions on Visualization and Computer Graphics*, 28(12):4127–4140, 2022.

[35] Ed H. Chi, James Pitkow, Jock Mackinlay, Peter Pirolli, Rich Gossweiler, and Stuart K. Card. Visualizing the evolution of Web ecologies. In *Proceedings of the SIGCHI*

Conference on Human Factors in Computing Systems, CHI '98, pages 400–407, USA, January 1998. ACM Press/Addison-Wesley Publishing Co.

[36] Ed Huai-hsin Chi. A taxonomy of visualization techniques using the data state reference model. In *IEEE Symposium on Information Visualization 2000. INFOVIS 2000. Proceedings*, pages 69–75. IEEE, 2000.

[37] Gouthami Chintalapani, Catherine Plaisant, and Ben Shneiderman. Extending the Utility of Treemaps with Flexible Hierarchy. In *Proceedings of the Information Visualisation, Eighth International Conference*, IV '04, pages 335–344, USA, July 2004. IEEE Computer Society.

[38] William S. Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984. Publisher: Taylor & Francis.

[39] Christopher Collins, Gerald Penn, and Sheelagh Carpendale. Bubble Sets: Revealing Set Relations with Isocontours over Existing Visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1009–1016, November 2009.

[40] B. Craft and P. Cairns. Beyond guidelines: what can we learn from the visual information seeking mantra? In *Ninth International Conference on Information Visualisation (IV'05)*, pages 110–118, 2005.

[41] Maurício Rossi de Oliveira and Celmar Guimarães da Silva. Adapting heuristic evaluation to information visualization-a method for defining a heuristic set by heuristic grouping. In *International Conference on Information Visualization Theory and Applications*, volume 4, pages 225–232. SCITEPRESS, 2017.

[42] Jean-Daniel Fekete. The InfoVis Toolkit. In *Proceedings of the IEEE Symposium on Information Visualization*, INFOVIS '04, pages 167–174, USA, October 2004. IEEE Computer Society.

[43] Jean-Daniel Fekete, Pierre-Luc Hémery, Thomas Baudel, and Jo Wood. Obvious: A meta-toolkit to encapsulate information visualization toolkits — One toolkit to bind them all. In *2011 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 91–100, October 2011.

[44] Jean-Daniel Fekete and Catherine Plaisant. Infovis 2003 contest - general tasks applicable to most trees. `http://www.cs.umd.edu/hcil/iv03contest/generaltasks.html`, 2003. IEEE Symposium on Information Visualization; Last accessed: June 18, 2025.

[45] Carolin Fiedler, Willy Scheibel, Daniel Limberger, Matthias Trapp, and Jürgen Döllner. Survey on user studies on the effectiveness of treemaps. In *Proceedings of the 13th International Symposium on Visual Information Communication and Interaction*, VINCI '20, pages 1–10, New York, NY, USA, December 2020. Association for Computing Machinery.

[46] Camilla Forsell. Evaluation in information visualization: Heuristic evaluation. In *2012 16th International Conference on Information Visualisation*, pages 136–142, 2012.

[47] Camilla Forsell and Jimmy Johansson. An heuristic set for evaluation in information visualization. In *Proceedings of the International Conference on Advanced Visual Interfaces*, AVI '10, page 199206, New York, NY, USA, 2010. Association for Computing Machinery.

[48] Ying-Huey Fua, Matthew O Ward, and Elke A Rundensteiner. Navigating hierarchies with structure-based brushes. In *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis' 99)*, pages 58–64. IEEE, 1999.

[49] Tsai-Ling Fung, Jia-Kai Chou, and Kwan-Liu Ma. A design study of personal bibliographic data visualization. In *2016 IEEE Pacific Visualization Symposium (PacificVis)*, pages 244–248. IEEE, 2016.

[50] Anna Georgelis. Multiperspective visualization of genealogy data. Master's thesis, Linköping University, Media and Information Technology, 2018.

[51] Mohammad Ghoniem, Jean-Daniel Fekete, and Philippe Castagliola. On the readability of graphs using node-link and matrix-based representations: A controlled experiment and statistical analysis. *Information Visualization*, 4(2):114–135, 2005.

[52] Martin Graham and Jessie Kennedy. Visual exploration of alternative taxonomies through concepts. *Ecological Informatics*, 2(3):248–261, 2007.

[53] Martin Graham and Jessie Kennedy. A survey of multiple tree visualisation. *Information Visualization*, 9(4):235–252, 2010-12.

[54] Lars Grammel, Chris Bennett, Melanie Tory, and Margaret-Anne Storey. A Survey of Visualization Construction User Interfaces. In Mario Hlawitschka and Tino Weinkauf, editors, *EuroVis - Short Papers*. The Eurographics Association, 2013.

[55] John Alexis Guerra-Gómez, Audra Buck-Coleman, Catherine Plaisant, and Ben Shneiderman. Treeversity: Interactive visualizations for comparing two trees with structure and node value changes. In *Proceedings of the Conference of the Design Research Society (DRS)*, volume 2, pages 640–653, 2012.

[56] Sebastian Hahn and Jürgen Döllner. Hybrid-treemap layouting. In *Proceedings of the Eurographics/IEEE VGTC Conference on Visualization: Short Papers*, EuroVis '17, pages 79–83, Goslar, DEU, June 2017. Eurographics Association.

[57] Xi He and Ying Zhu. An indented level-based tree drawing algorithm for text visualization. In *2020 24th International Conference Information Visualisation (IV)*, pages 258–263, 2020.

[58] Jeffrey Heer, Stuart K Card, and James A Landay. Prefuse: a toolkit for interactive information visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 421–430, 2005.

[59] Nathalie Henry, Jean-Daniel Fekete, and Michael J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1302–1309, 2007.

[60] Ralph D. Hill, Tom Brinck, Steven L. Rohall, John F. Patterson, and Wayne Wilner. The *Rendezvous* architecture and language for constructing multiuser applications. *ACM Transactions on Computer-Human Interaction*, 1(2):81–125, June 1994.

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

[62] Peter Hüsken and Jürgen Ziegler. Degree-of-interest visualization for ontology exploration. In Cécilia Baranauskas, Philippe Palanque, Julio Abascal, and Simone Diniz Junqueira Barbosa, editors, *Human-Computer Interaction INTERACT 2007*, Lecture Notes in Computer Science, pages 116–119. Springer, 2007.

[63] T. J. Jankun-Kelly, Ma Kwan-Liu, and M. Gertz. A model and framework for visualization exploration. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):357–369, March-April 2007.

[64] Brian Scott Johnson. *Treemaps: Visualizing Hierarchical and Categorical Data*. PhD thesis, University of Maryland, 1993.

[65] Mark S. Keller, Trevor Manz, and Nils Gehlenborg. Use-coordination: Model, grammar, and library for implementation of coordinated multiple views. In *2024 IEEE Visualization and Visual Analytics (VIS)*, pages 166–170, 2024.

[66] E. Kleiberg, H. van de Wetering, and J.J. van Wijk. Botanical visualization of huge hierarchies. In *IEEE Symposium on Information Visualization, 2001. INFOVIS 2001.*, pages 87–94, 2001.

[67] Alfred Kobsa. User experiments with tree visualization systems. In *IEEE Symposium on Information Visualization*, pages 9–16. IEEE, 2004.

[68] Philipp Koytek, Charles Perin, Jo Vermeulen, Elisabeth André, and Sheelagh Carpendale. Mybrush: Brushing and linking with personal agency. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):605–615, 2018.

[69] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '95*, pages 401–408, Denver, Colorado, United States, 1995. ACM Press.

[70] Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, and Nathalie Henry. Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization*, BELIV '06, pages 1–5, New York, NY, USA, May 2006. Association for Computing Machinery.

[71] Guozheng Li, Min Tian, Qinmei Xu, Michael J McGuffin, and Xiaoru Yuan. GoTree: A grammar of tree visualizations. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020.

[72] Manuel Lima. *The book of trees: Visualizing branches of knowledge*, volume 5. Princeton Architectural Press New York, 2014.

[73] M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki, and K. Wenger. DEVise: Integrated querying and visual exploration of large datasets. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data - SIGMOD '97*, pages 301–312, Tucson, Arizona, United States, 1997. ACM Press.

[74] Lim Kian Long, Lim Chien Hui, Gim Yeong Fook, and Wan Mohd Nazmee Wan Zainon. A study on the effectiveness of tree-maps as tree visualization techniques. *Procedia Computer Science*, 124:108–115, 2017.

[75] Rainer Lutz, Daniel Rausch, Fabian Beck, and Stephan Diehl. Get your directories right: From hierarchy visualization to hierarchy manipulation. In *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 25–32. IEEE, 2014.

[76] Sehi LYi, Qianwen Wang, Fritz Lekschas, and Nils Gehlenborg. Gosling: A grammar-based toolkit for scalable and interactive genomics data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):140–150, 2022.

[77] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110141, April 1986.

[78] Honghui Mei, Yuxin Ma, Yating Wei, and Wei Chen. The design space of construction tools for information visualization: A survey. *Journal of Visual Languages & Computing*, 44:120–132, 2018.

[79] Kazuo Misue. Area-adaptive drawing of rooted trees. In *2024 IEEE 17th Pacific Visualization Conference (PacificVis)*, pages 152–161, 2024.

[80] Tamara Munzner. *Visualization analysis and design*. CRC Press, 2014.

[81] Tamara Munzner, François Guimbretière, Serdar Tasiran, Li Zhang, and Yunhong Zhou. Treejuxtaposer: scalable tree comparison using focus+context with guaranteed visibility. *ACM Transactions on Graphics*, 22(3):453462, July 2003.

[82] B.A. Myers, D.A. Giuse, R.B. Dannenberg, B.V. Zanden, D.S. Kosbie, E. Pervin, A. Mickish, and P. Marchal. Garnet: Comprehensive support for graphical, highly interactive user interfaces. *Computer*, 23(11):71–85, November 1990.

[83] Nicholas Hugo Müller, Benny Liebold, Daniel Pietschmann, Peter Ohler, and Paul Rosenthal. Hierarchy visualization designs and their impact on perception and problem solving strategies. In *Proceedings of the International Conference on Advances in Computer-Human Interactions*, pages 93–101, 2017.

[84] Petra Neumann, Stefan Schlechtweg, and Sheelagh Carpendale. ArcTrees: Visualizing relations in hierarchical data. In *Proceedings of the Seventh Joint Eurographics / IEEE VGTC Conference on Visualization*, EUROVIS'05, pages 53–60. Eurographics Association, 2005. ISSN: 1727-5296 event-place: Leeds, United Kingdom.

[85] Jakob Nielsen. Finding usability problems through heuristic evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, page 373380, New York, NY, USA, 1992. Association for Computing Machinery.

[86] Jakob Nielsen. *Usability engineering*. Morgan Kaufmann, 1994.

[87] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, page 249256, New York, NY, USA, 1990. Association for Computing Machinery.

[88] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, pages 249–256. Association for Computing Machinery, 1990. event-place: Seattle, Washington, USA.

[89] Chris North and Ben Shneiderman. A taxonomy of multiple window coordinations. Technical Report CS-TR-3854, University of Maryland Department of Computer Science, 1997.

[90] Chris North and Ben Shneiderman. Snap-together visualization: A user interface for coordinating visualizations via relational schemata. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '00, pages 128–135, New York, NY, USA, May 2000. Association for Computing Machinery.

[91] John Paisley, Chong Wang, David M. Blei, and Michael I. Jordan. Nested hierarchical dirichlet processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):256–270, 2015.

[92] Aditeya Pandey, Uzma Haque Syeda, Chaitya Shah, John A. Guerra-Gomez, and Michelle A. Borkin. A state-of-the-art survey of tasks for tree design and evaluation with a curated task dataset. *IEEE Transactions on Visualization and Computer Graphics*, 28(10):3563–3584, 2022.

[93] Hal Pashler and Steven Yantis. *Stevens' handbook of experimental psychology, sensation and perception*, volume 1. John Wiley and sons, 3 edition, 2002-11.

[94] Georgios A. Pavlopoulos, Seán I. O'Donoghue, Venkata P. Satagopam, Theodoros G. Soldatos, Evangelos Pafilis, and Reinhard Schneider. Arena3D: Visualization of biological networks in 3D. *BMC Systems Biology*, 2(1):104, November 2008.

[95] Ken Perlin and David Fox. Pad: an alternative approach to the computer interface. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, page 5764, New York, NY, USA, 1993. Association for Computing Machinery.

[96] C. Plaisant, J. Grosjean, and B.B. Bederson. SpaceTree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.*, pages 57–64, Boston, MA, USA, 2002. IEEE Computer Society.

[97] Catherine Plaisant. The challenge of information visualization evaluation. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, page 109116, New York, NY, USA, 2004. Association for Computing Machinery.

[98] Ramana Rao and Stuart K. Card. The table lens: merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, page 318322, New York, NY, USA, 1994. Association for Computing Machinery.

[99] E.M. Reingold and J.S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, SE-7(2):223–228, 1981.

[100] Jonathan C. Roberts. State of the art: Coordinated & multiple views in exploratory visualization. In *Proceedings of the Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV)*, pages 61–71, Zürich, Switzerland, 2007. IEEE Computer Society.

[101] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone Trees: animated 3D visualizations of hierarchical information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, page 189194, New York, NY, USA, 1991. Association for Computing Machinery.

[102] J. Rosindell and L. J. Harmon. OneZoom: A Fractal Explorer for the Tree of Life. *PLOS Biology*, 10(10):e1001406, October 2012.

[103] Ursula Rost and Erich Bornberg-Bauer. TreeWiz: Interactive exploration of huge trees. *Bioinformatics (Oxford, England)*, 18(1):109–114, January 2002.

[104] S.F. Roth, P. Lucas, J.A. Senn, C.C. Gomberg, M.B. Burks, P.J. Stroffolino, A.J. Kolojechick, and C. Dunmire. Visage: A user interface environment for exploring information. In *Proceedings IEEE Symposium on Information Visualization '96*, pages 3–12,, San Francisco, CA, USA, 1996. IEEE Computer Society Press.

[105] Bahador Saket, Hannah Kim, Eli T. Brown, and Alex Endert. Visualization by demonstration: An interaction paradigm for visual data exploration. *IEEE Trans. Visual. Comput. Graphics*, 23(1):331–340, 2017-01.

[106] Arnaud Sallaberry, Yang-chih Fu, Hwai-Chung Ho, and Kwan-Liu Ma. Contact Trees: Network Visualization beyond Nodes and Edges. *PLoS ONE*, 11(1):e0146368, January 2016.

[107] Manojit Sarkar and Marc H. Brown. Graphical fisheye views of graphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, page 8391, New York, NY, USA, 1992. Association for Computing Machinery.

[108] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, January 2017.

[109] Arvind Satyanarayan, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):659–668, January 2016.

[110] Arvind Satyanarayan, Kanit Wongsuphasawat, and Jeffrey Heer. Declarative interaction design for data visualization. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, pages 669–678, Honolulu Hawaii USA, October 2014. ACM.

[111] Willy Scheibel, Matthias Trapp, Daniel Limberger, and Jürgen Döllner. A taxonomy of treemap visualization techniques:. In *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, pages 273–280. SCITEPRESS - Science and Technology Publications, 2020.

[112] Jean Scholtz. Developing guidelines for assessing visual analytics environments. *Information Visualization*, 10(3):212–231, 2011.

[113] Hans-Jorg Schulz. Treevis.net: A Tree Visualization Reference. *IEEE Computer Graphics and Applications*, 31(6):11–15, 2011.

[114] Hans-Jorg Schulz, Steffen Hadlak, and Heidrun Schumann. The design space of implicit hierarchy visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 17(4):393–411, 2011.

[115] Hans-Jörg Schulz, Zabedul Akbar, and Frank Maurer. A generative layout approach for rooted tree drawings. In *2013 IEEE Pacific Visualization Symposium (PacificVis)*, pages 225–232, 2013.

[116] Nihar Sheth, Katy Börner, Jason Baumgartner, Ketan Mane, and Eric Wernert. Treemap, Radial Tree and 3D Tree Visualizations. In *Poster Compendium, IEEE Information Visualization Conference*, pages 128–129, 2003.

[117] B. Shneiderman and M. Wattenberg. Ordered treemap layouts. In *IEEE Symposium on Information Visualization, 2001. INFOVIS 2001.*, pages 73–78. IEEE, 2001.

[118] Ben Shneiderman. Tree Visualization with Tree-Maps: 2-d Space-Filling Approach. *ACM Trans. Graph.*, 11(1):92–99, 1992.

[119] Ben Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, 11(6):70–77, November 1994.

[120] J. Stasko and E. Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *IEEE Symposium on Information Visualization 2000. INFOVIS 2000. Proceedings*, pages 57–65. IEEE Comput. Soc, 2000.

[121] John Stasko, Richard Catrambone, Mark Guzdial, and Kevin Mcdonald. An evaluation of space-filling information visualizations for depicting hierarchical structures. *International Journal of Human-Computer Studies*, 53(5):663–694, 2000-11.

[122] John Stasko and Eugene Zhang. Focus+ context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *IEEE Symposium on Information Visualization 2000. INFOVIS 2000. Proceedings*, pages 57–65. IEEE, 2000.

[123] C. Stolte, D. Tang, and P. Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.

[124] Masahiro Takatsuka and Mark Gahegan. GeoVISTA Studio: A codeless visual programming environment for geoscientific data analysis and visualization. *Computers & Geosciences*, 28(10):1131–1144, December 2002.

[125] Alvin Tarrell, Ann Fruhling, Rita Borgo, Camilla Forsell, Georges Grinstein, and Jean Scholtz. Toward visualization-specific heuristic evaluation. In *Proceedings of the Fifth Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization*, pages 110–117, 2014.

[126] Alexandru Telea and David Auber. Code flows: visualizing structural evolution of source code. In *Proceedings of the 10th Joint Eurographics / IEEE - VGTC Conference on Visualization*, EuroVis'08, pages 831–838, Chichester, GBR, 2008. The Eurographs Association & John Wiley & Sons, Ltd.

[127] Soon Tee Teoh. A study on multiple views for tree visualization. In *Visualization and Data Analysis 2007*, volume 6495, pages 99–110. SPIE, 2007-01-29.

[128] Soon Tee Teoh and Kwan-Liu Ma. RINGS: A technique for visualizing large hierarchies. In *Revised Papers from the 10th International Symposium on Graph Drawing*, GD '02, pages 268–275. Springer-Verlag, 2002.

[129] Christian Tominski, James Abello, Frank Van Ham, and Heidrun Schumann. Fisheye tree views and lenses for graph visualization. In *Tenth International Conference on Information Visualisation (IV'06)*, pages 17–24. IEEE, 2006.

[130] Ying Tu and Han-Wei Shen. Balloon Focus: a Seamless Multi-Focus+Context Method for Treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1157–1164, 2008.

[131] Edward R. Tufte. *The visual display of quantitative information.* Graphics Press, USA, 1986.

[132] D. Turo and B. Johnson. Improving the visualization of hierarchies with treemaps: design issues and experimentation. In *Proceedings Visualization '92*, pages 124–131, 1992.

[133] Frank Van Ham and Jarke J van Wijk. Beamtrees: Compact visualization of large hierarchies. *Information Visualization*, 2(1):31–39, 2003.

[134] Emily Wall, Meeshu Agnihotri, Laura Matzen, Kristin Divis, Michael Haass, Alex Endert, and John Stasko. A heuristic approach to value-driven evaluation of visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):491–500, 2019.

[135] Guanqun Wang, Tsuneo Nakanishi, and Akira Fukuda. 2-D layout for tree visualization: a survey. In *MATEC Web of Conferences*, volume 56. EDP Sciences, 2016.

[136] Yue Wang, Soon Tee Teoh, and Kwan-Liu Ma. Evaluating the effectiveness of tree visualization systems for knowledge discovery. In *Proceedings of the Eighth Joint Eurographics / IEEE VGTC Conference on Visualization*, EUROVIS'06, pages 67–74. Eurographics Association, 2006. Lisbon, Portugal.

[137] Colin Ware. *Information Visualization: Perception for Design.* Morgan Kaufmann Publishers Inc., 4$^{\text{th}}$ edition, 2019.

[138] Chris Weaver. Building highly-coordinated visualizations in Improvise. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*, pages 159–166, Austin, TX, October 2004. IEEE Computer Society.

[139] Chris Weaver. Is coordination a means to collaboration? In *Proceedings of the International Conference on Coordinated & Multiple Views in Exploratory Visualization (CMV)*, pages 80–84, Zürich, Switzerland, July 2007. IEEE Computer Society.

[140] Chris Weaver. Patterns of coordination in Improvise visualizations. In *Proceedings of SPIE (Visualization and Data Analysis)*, pages 1–12, San Jose, CA, January 2007. SPIE.

[141] Chris Weaver. Cross-filtered views for multidimensional visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):192–204, March-April 2010.

[142] Chris Weaver. Multidimensional data dissection using attribute relationship graphs. In *2010 IEEE Symposium on Visual Analytics Science and Technology*, pages 75–82, Salt Lake City, UT, USA, October 2010. IEEE.

[143] Max Wertheimer. Untersuchungen zur lehre von der gestalt. II. *Psychologische Forschung*, 4(1):301–350, 1923-01-01.

[144] Richard Wettel and Michele Lanza. Visual exploration of large-scale system evolution. In *2008 15th Working Conference on Reverse Engineering*, pages 219–228. IEEE, 2008.

[145] Kai Wetzel. Pebbles-using circular treemaps to visualize disk usage. `https://lip.sourceforge.net/ctreemap.html`, 2004. Volume: 2; Last accessed: June 18, 2025.

[146] Leland Wilkinson. *The Grammar of Graphics*, pages 375–414. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[147] Linda Woodburn, Yalong Yang, and Kim Marriott. Interactive visualisation of hierarchical quantitative data: An evaluation. In *2019 IEEE Visualization Conference (VIS)*, pages 96–100. IEEE, 2019-10.

[148] Worldometers.info. COVID-19 CORONAVIRUS PANDEMIC. `https://www.worldometers.info/coronavirus`. Last accessed: June 18, 2025.

[149] Bowen Yu and Cláudio T Silva. Flowsense: A natural language interface for visual data exploration within a dataflow system. *IEEE transactions on visualization and computer graphics*, 26(1):1–11, 2019.

[150] Bowen Yu and Cláudio T. Silva. Visflow - web-based visualization framework for tabular data with a subset flow model. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):251–260, 2017.

[151] Shengdong Zhao, Michael J McGuffin, and Mark H Chignell. Elastic hierarchies: Combining treemaps and node-link diagrams. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, pages 57–64. IEEE, 2005.

[152] Shengdong Zhao, M.J. McGuffin, and M.H. Chignell. Elastic hierarchies: combining treemaps and node-link diagrams. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, pages 57–64, 2005-10. ISSN: 1522-404X.

[153] Bin Zhou, Rolf D Vogt, Xueqiang Lu, Xiaoguang Yang, Changwei Lü, Christian W Mohr, and Liang Zhu. Land use as an explanatory factor for potential phosphorus loss risk, assessed by P indices and their governing parameters. *Environmental Science: Processes & Impacts*, 17(8):1443–1454, 2015. Publisher: Royal Society of Chemistry.

[154] Jonathan Zong, Dhiraj Barnwal, Rupayan Neogy, and Arvind Satyanarayan. Lyra 2: Designing interactive visualizations by demonstration. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):304–314, 2021.

[155] Torre Zuk and Sheelagh Carpendale. Theoretical analysis of uncertainty visualizations. In *Visualization and Data Analysis 2006*, volume 6060, pages 66–79. SPIE, 2006-01-16.

[156] Torre Zuk, Lothar Schlesier, Petra Neumann, Mark S. Hancock, and Sheelagh Carpendale. Heuristics for information visualization evaluation. In *Proceedings of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization*, BELIV '06, pages 1–6, New York, NY, USA, May 2006. Association for Computing Machinery.