Visualizing Dynamic Hierarchies in Graph Sequences

Corinna Vehlow, *Student Member, IEEE,* Fabian Beck, *Member, IEEE Computer Society,* and Daniel Weiskopf, *Member, IEEE Computer Society*

Abstract—Graphs are used to model relations between objects, where these objects can be grouped hierarchically based on their connectivity. In many applications, the relations change over time and so does the hierarchical group structure. We developed a visualization technique that supports the analysis of the topology and the hierarchical group structure of a dynamic graph and the tracking of changes over time. Each graph of a sequence is visualized by an adjacency matrix, where the hierarchical group structure is encoded within the matrix using indentation and nested contours, complemented by icicle plots attached to the matrices. The density within and between subgroups of the hierarchy is represented within the matrices using a gray scale. To visualize changes, transitions and dissimilarities between the hierarchically structured graphs are shown using a flow metaphor and color coding. The design of our visualization technique allows us to show more than one hierarchical group structure of the same graph by stacking the sequences, where hierarchy comparison is supported not only within but also between sequences. To improve the readability, we minimize the number of crossing curves within and between sequences based on a sorting algorithm that sweeps through the sequences of hierarchies.

Index Terms—Dynamic graph, hierarchical graph, graph visualization.

1 Introduction

Graphs represent relations between objects, e.g., calls between methods of a software system, interactions between biological entities, or social relationships between people. A hierarchical group structure of these objects leads to a compound graph that gives insight into the high-level organization. Clustering methods are commonly used to obtain hierarchically organized groups that have a higher density of edges within than to other groups. The hierarchical group structure can be used to navigate through or to abstract the graph [1]. There are several explicit visualization techniques for hierarchically structured graphs, where the hierarchy visualization is either juxtaposed to, superimposed on, or integrated with, the graph visualization [2].

In many applications, including software engineering or social networks, relations change over time. There is much previous work on the visualization of dynamic graphs in general [3] that can be used for typical graph analysis tasks for graph evolution [4], [5]. Among existing approaches, several visualize a static hierarchical group structure of the dynamic graph [2], i.e., one hierarchy for all points in time. This might be sufficient if the changes of the graph topology are only minor, i.e., only the edge weights change or only few edges are added or removed. For significantly changing dynamic graphs, in contrast, it is more suitable to determine the hierarchical group structure for each graph of the sequence individually. In this way, we can analyze changes at the high-level organization of the graph. To identify whether the static or the dynamic structure is more suitable, it is helpful to visualize and compare both. In addition, different approaches exist to build the dynamic group structure from the graph topology at only one point in time or a subset of points in time. To compare these structures, it is necessary to visualize them together. There are some visualization techniques for dynamic hierarchies [6], [7], but there is almost none for dynamic hierarchical group structures of dynamic graphs [2]. In addition, there are no techniques that support the comparison of multiple sequences of hierarchical group structures. With this paper, we want to address these gaps.

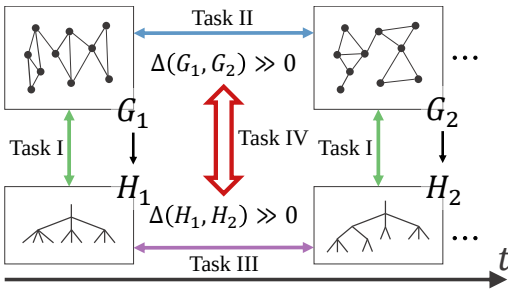


Fig. 1. Illustration of the two dynamic components—graph topology and hierarchical group structure—and the tasks connected to both.

The hierarchical group structure describes an abstraction of the graph (indicated by the black arrows in Figure 1). Given that the changes of the graph are severe (top row), the hierarchical group structure changes over time as a consequence (bottom row). In this case, the following analysis tasks arise (illustrated by colored arrows in Figure 1):

I) Topology-to-Hierarchy Task: analyze the consistency between hierarchical group structure and graph topology of individual graphs (green).
II) Topology-to-Topology Task: compare graph topologies over time (blue).
III) Hierarchy-to-Hierarchy Task: compare hierarchical group structures over time (purple).

• *The authors are with VISUS, University of Stuttgart, Germany E-mail: {corinna.vehlow, fabian.beck, weiskopf}@visus.uni-stuttgart.de.*

Manuscript received July 29, 2015; revised November 19, 2015.

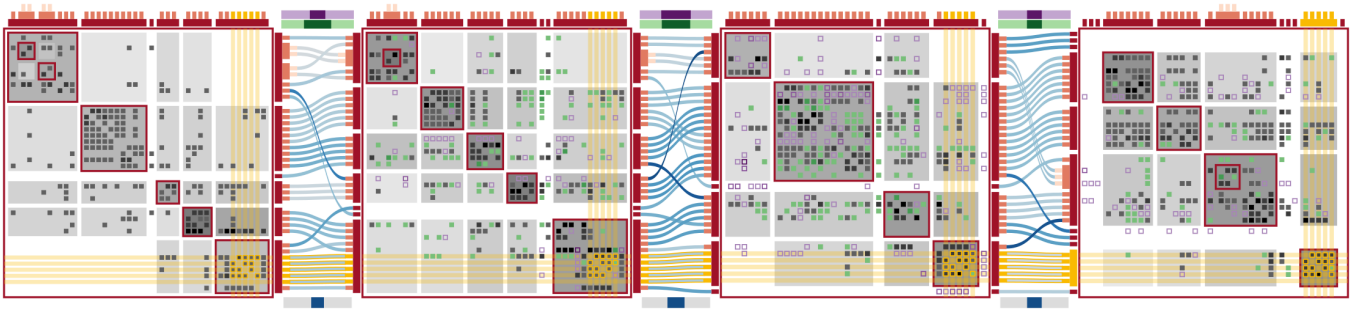


Fig. 2. Friendship network among students over time [8] from left to right, where the intensity of the relationship is encoded in the edge weights mapped to the brightness of shades of gray, green (added edges), or purple (removed edges). For each point in time, the hierarchical group structure is encoded within the matrix using indented nested contours and visualized by icicle plots attached to the matrix. To compare hierarchies, we draw time transition edges that connect corresponding leaves of the icicle plots. Groups of friends change over time, while some groups remain relatively stable, e.g., the group selected at the last time point and highlighted in yellow over all points in time.

IV) Change-to-Change Task: analyze how far changes in the graph topology are reflected in changes in the hierarchical group structure (red).

The first three tasks are supported by several visualization techniques but only individually and not all by the same technique. Using independent techniques to work on all these tasks can hinder the workflow due to multiple separated views and inconsistent representations. We developed a novel visualization technique for dynamic hierarchically structured graphs supporting all of the tasks. This also includes the Change-to-Change Task (red double arrow), which is not supported by any of the available techniques. This task arises from the combination of both dynamic components, graph structure and hierarchy: large changes in the graph topology ($\Delta(G_1, G_2) \gg 0$) imply large changes in the hierarchical group structure ($\Delta(H_1, H_2) \gg 0$). In comparison to existing techniques for dynamic graphs that show only a static hierarchical group structure of the graph (hence, only supporting the Topology-to-Topology Task), our technique supports the visualization and comparison of dynamic hierarchies over time. To the best of our knowledge, there are only two techniques visualizing the dynamic hierarchy in combination with the underlying dynamic graph [9], [10]. However, these do not facilitate all four tasks. In particular, comparisons over time (Topology-to-Topology, Hierarchy-to-Hierarchy, and Change-to-Change Tasks) are not sufficiently supported using animation [10] or diagrams stacked in 3D [9] without visually linking elements at successive points in time.

Using our technique (Figure 2), the dynamic graph is shown as horizontal sequence of adjacency matrices. For the Topology-to-Hierarchy Task, it is necessary to visualize both hierarchy and graph in an integrated way. Our technique encodes the hierarchical group structure in each of the matrices using visual nesting, where edge densities within and between subgroups are encoded in the matrix using a gray scale. To facilitate the analysis of changes, time transitions between the hierarchical group structures are visualized between the matrices using a flow metaphor; time transitions are curves connecting matching vertices of successive time steps. The design of our technique allows us to visualize and compare different hierarchical group structures of the same dynamic graph using comparison transitions (Figure 6). Hence, it is even possible to compare a dynamic and a static hierarchical group structure. This addresses the aspect of group structure comparison, which is a current research challenge [2]. To minimize the edge crossings of time and comparison transitions, we extended a sorting algorithm that reorders the internal nodes and leaves of all hierarchies over several sequences considering proximity. To highlight vertices that change with respect to their integration into two hierarchies, we designed a dissimilarity metric that is mapped to the transitions.

The integrated visualization of the dynamic hierarchical group structure together with the dynamic graph supports the analysis of both dynamic aspects. It allows us to investigate whether the hierarchical group structure represents a reasonable abstraction of the graph and whether the group structure changes in analogy to the graph. We present the following contributions:

- an integrated visualization of the dynamic hierarchical group structure together with the dynamic graphs (all tasks),
- a visualization that supports the comparison of dynamic hierarchical group structures (Hierarchy-to-Hierarchy Task),
- an improved encoding of the hierarchical group structure in an adjacency matrix representation (Topology-to-Hierarchy and Change-to-Change Tasks),
- a sorting algorithm for sequences of hierarchies to facilitate their comparison (Hierarchy-to-Hierarchy Task), and
- a transition-based metric that describes the dissimilarity between hierarchies (Hierarchy-to-Hierarchy Task).

In contrast to existing techniques, the design of our integrated visualization allows us to handle all four tasks.

2 Related Work

In this section, we summarize the work in fields related to our visualization technique, including the visualization of dynamic graphs, group structures, and flow-oriented data.

Dynamic graphs: Graph sequences are commonly visualized using timeline- or animation-based methods for node-link diagrams or adjacency matrices [3]. We decided to use adjacency matrices instead of node-link diagrams to visualize the dynamic graph because it allows us to encode the hierarchical structure within each graph without visual clutter and—at the same time—visually link neighboring graphs for comparison using a flow metaphor. Intra-cell timeline techniques [11], [12] and layered matrix visualizations [13], [14], [15], [16], [17] are examples of matrix-based approaches. In layered approaches, the matrices are either juxtaposed in 2D or stacked into 3D. Intra-cell timeline methods only support the visualization together with the global hierarchical group structure—static over the full time span, e.g., by attaching a layered icicle-like representation to the matrix [11].

They are not suitable for the visualization of dynamic hierarchical group structures and varying vertex order.

Layered matrices, in contrast, allow us to sort the vertices for each graph of the sequence individually based on its hierarchical group structure. So far, this approach was mainly used for dynamic graphs without an additional hierarchical group structure [13], [14], [18] or for dynamic graphs with a static hierarchical group structure [15], [16], [17]. The matrices representing individual points in time are placed next to each other in one dimension [14], [18], they can be arranged in radial layers [15], [16], [17], or stacked in three-dimensional space [13], [14]. The static hierarchical group structure of a dynamic graph can be visualized using hierarchy representations, such as a tree node-link diagram [15], indented plot [16], or icicle plot [17], attached to, or superimposed on, the layered dynamic graph representation [2]. However, these techniques do not support the visualization of a dynamic hierarchical group structure.

Dynamic group structures: The visualization technique for dynamic group structures in dynamic graphs by Vehlow et al. [19] is similar to our technique. However, their method represents the graphs by node-link diagrams and it only supports the visualization of flat—but not hierarchical—group structures [2].

Only two techniques visualize the changes of both the hierarchical group structure and the graph topology. The animated node-link technique by Reitz et al. [10] uses the hierarchical group structure to control the animation and to automatically aggregate subhierarchies that do not change. Although only those parts of the data that change are presented in detail, it is still difficult to perceive and understand the changes in the animation. Moreover, it is difficult to keep track of the vertices' placements in the course of the sequence of hierarchical group structures, especially as a vertex may not even be visualized at some points in time. Ahmed et al. [9] visualize each hierarchically structured graph using radial representations and stack them in 3D. Because of the 3D positioning of the layers, it is difficult to keep track of vertices, their relations, and their hierarchical placement over time. Also the change of the layout over time makes it difficult to compare the graphs because no visual links connect matching elements between successive points in time. Within both techniques, it is hard to monitor the changes in the hierarchical group structure as well as the graph topology and to investigate the consistency between the hierarchical group structure and the graph topology. In contrast, our technique supports all these tasks due to its design, the dissimilarity metric, and the sorting algorithm.

Hierarchical group structures are often visualized using tree visualization techniques [20], [21]. The most common approaches include node-link diagrams, stacking approaches like icicle or indented plots, or nesting approaches like treemaps. Graham and Kennedy [22] summarize visualization techniques for pairs of trees but also sequences of trees. They classify existing approaches into five categories: edge drawing, coloring, animation, matrix representation, and agglomeration. Using edge drawing, two hierarchies visualized—e.g., as icicle plots—are juxtaposed and matching leaves are connected by straight [23] or bundled [24] links. Coloring approaches, such as TreeJuxtaposer [25], visualize links by color coding. Within matrix representations, such links are visualized using an adjacency matrix with two hierarchies attached to it [26]. Agglomeration methods merge two hierarchies into one representation; some approaches encode differences visually [27], [28], [29], others superimpose both tree visualizations [30]. Techniques that support the comparison of multiple hierarchical group structures mainly use edge drawing [7], [31], [32] or coloring [6], [33].

CodeFlows [7] visualizes the temporal evolution of hierarchies using juxtaposed mirrored icicle plots. Alternatively, the hierarchies can be visualized using node-link diagrams on individual planes stacked in 2.5D [31] or 3D [32]. Using coloring, hierarchies are arranged as small multiples with all differences to a reference hierarchy color-coded within each hierarchy visualization [6], [33]. Our visualization technique uses an edge-drawing approach to compare juxtaposed hierarchically structured graphs.

Flow-oriented visualization: There are several flow-like visualization techniques that are visually similar to our technique, employing a timeline from left to right where groups of elements literally flow from one state to the other [34], [35], [36], [37]. In contrast to our technique, these techniques do not visualize graph sequences. Sankey diagrams [36] and ThemeRiver [35] show continuous flow-like shapes, whereas in Domino [34] and ConnectedCharts [37] the flow is interrupted by small charts. Domino [34] visualizes multiple interconnected datasets—e.g., heatmaps shown as matrix—aligned horizontally, where relationships between them are encoded by links. In ConnectedCharts [37], multidimensional multivariate data is displayed using multiple charts (bar charts and scatter plots) and relationships between them are represented by curves to show conceptual links. An alternative to flow-like representations of transitions is to use visual transition matrices instead [38].

3 Overview and Design Choices

Our visualization technique aims at the visual analysis of the changes within a dynamic graph and its hierarchical structure, especially the consistency between the hierarchical group structure and the graph topology. This requires two types of comparisons: the comparison of successive graphs as well as successive hierarchical group structures. Different group structures can be defined on the dynamic graph. Hence, the hierarchy comparison task involves not only the visual comparison of adjacent hierarchically structured graphs within sequences, i.e., of successive points in time, but also between graph sequences, i.e., of different hierarchical group structures of the same dynamic graph sequence. These could include several dynamic and static hierarchical structures.

To support these aims, our visualization technique combines a small multiples representation of the dynamic graph, icicle plots, contours, and edge-based comparisons of hierarchies. Although the individual visual components are not new by themselves, the combination of them is novel. However, the naive hybrid of these techniques implies several problems with respect to the tasks mentioned in Section 1. Therefore, we adjusted the different components to a visualization technique that is applicable to dynamic hierarchically structured graphs and supports all tasks.

First, naive contours within adjacency matrices—superimposed contours without changing the layout of the matrix—do not scale very well with respect to the graph size, i.e., they are barely recognizable if rows and columns are only one or few pixels wide. For better scalability, we developed an extended encoding of the hierarchical group structure within the matrix using indentation and edge density mapping. Second, it is difficult to compare graph topologies by using small multiples of common adjacency matrix representations. Topological changes are therefore encoded within each matrix as well as in an aggregated way between them. Third, using the edge-based

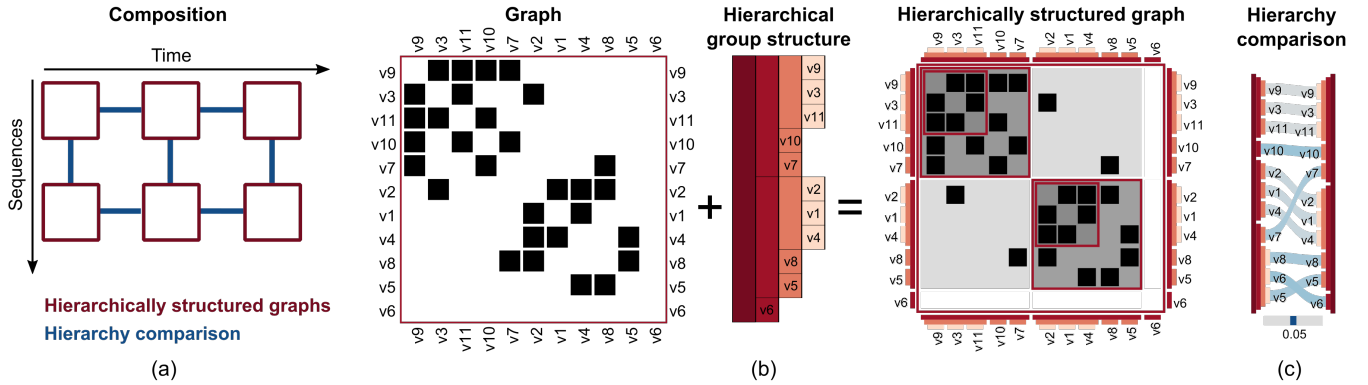


Fig. 3. Illustration of the visualization technique. (a) Our visualization consists of a matrix of adjacency matrices, where the time points of the dynamic graph are aligned horizontally and each row represents a sequence with a different hierarchical group structure of the dynamic graph. (b) The hierarchical group structure of each graph is encoded within the matrix using indented nested contours, edge density mapping, and attached icicle plots. (c) Changes between adjacent hierarchies are visualized by connecting the leaves of the icicle plots with cubic Bézier curves, where the hierarchy dissimilarity is mapped to color (for individual vertices) and visualized by a bar (for the complete hierarchy).

approach to compare hierarchies, the comparison can become a demanding task for more than one pair of hierarchies, because the user needs to investigate all edges and hierarchies to understand how pairs of hierarchies are dissimilar. Mapping our developed dissimilarity metric to individual edges as well as encoding the total dissimilarity between two hierarchies helps tackle this problem. Not least, a naive sorting of vertices based on the hierarchical group structures poses a problem for the hierarchy comparison and tracking of vertices within and across sequences due to the large amount of crossing edges. To minimize these crossings, we extended a sorting technique for the comparison of two hierarchies toward the comparison within and between an arbitrary number of sequences of variable length.

Visualizing the dynamic graph: We decided to use a time-to-space mapping instead of animation, because animation comes with higher cognitive load when trying to track elements over time to understand changes [39]. Moreover, the small multiples representation allows us to extend our technique to the comparison of multiple sequences. Not least, using a timeline layout has the advantage that we can visualize the time transitions between the hierarchical group structures of adjacent graphs explicitly using a flow metaphor. In particular, the dynamic graph is visualized using a layered matrix approach similar to Matrix Cubes [13]. In contrast to Matrix Cubes, we arrange the matrices horizontally in one dimension rather than in three dimensions and allow for different vertex orders based on the time-dependent hierarchical group structure rather than using a global sorting. In contrast to our approach, Matrix Cubes does not visualize any group structure explicitly. If different sequences of hierarchical group structures are defined on the same dynamic graph, we stack these sequences vertically (Figure 3(a)).

Adjacency matrix visualization: Adjacency matrix representations have a linear order of vertices, which allows us to include visual links between hierarchically structured graphs to track the vertices over time. Alternative graph representations that use a linear arrangement of vertices include parallel edge splatting [40], TimeArcTrees [41], or adjacency lists [42]. Although adjacency matrices require more space than these alternatives, we decided to use adjacency matrices for several reasons: First, matrices can reveal group structures and substructures (clusters) if the vertices are ordered appropriately [43] and the hierarchical group structure can be encoded within each graph representation individually. Second, using matrices, we can easily extend our technique to compare the hierarchical group structures not only within a graph sequence but also between different graph sequences. Third, matrices are well suited for dense graphs and support the visual encoding of edge densities within and between groups, which improves the scalability.

Each graph of a sequence is visualized using an adjacency matrix with the hierarchical group structure encoded in the matrix using nested contours (Figure 3(b)) like in TreeMatrix [44]. In contrast to TreeMatrix, in our adjacency matrices, the contours are indented with each hierarchy level. The same holds for the internal nodes of the icicle plots attached to the matrix. Moreover, we visualize the density of relations within and between groups of the hierarchical group structure within the matrix using a gray-scale mapping. Our extended visual mapping of the hierarchy within the matrix using indentation and density encoding reinforces the perception of the hierarchical group structure. Hence, our technique supports the perception of the consistency between the hierarchical group structure and the graph topology.

Comparison: Our application scenario involves the comparison of hierarchical group structures within and between sequences of changing hierarchies. Since we want to visualize the graph topology together with its hierarchical group structure, the hierarchy comparison approaches "matrix representations" and "agglomeration approaches" [22] are not suitable, especially considering that we need to visualize more than two hierarchical group structures. The comparison is facilitated using an edge-based approach using a flow metaphor (Figure 3(c)) similar to CodeFlows [7]. Within CodeFlows, hierarchies are visualized by icicle plots [45] facing each other such that the matching leaves in neighboring hierarchies can be connected by visual links.

We use cubic Bézier curves as visual links to generate continuous transitions between neighboring groups and subgroups. To integrate this hierarchy comparison into our single- or multi-sequence visualization of dynamic graphs, icicle plots are attached to the matrices at all sides and matching leaves are connected by horizontal (vertical) visual links for the within-sequence (between-sequence) comparison (Figure 3(a)). Visualizing the transition between hierarchically structured graphs allows us to track vertices along a sequence and across sequences. Moreover, we visualize a

dissimilarity metric for individual vertices and complete hierarchies using color mapping and bars, which helps identify changes in the hierarchical group structure.

4 Visualization Technique

In this section, we detail the design of our visualization technique introduced in the previous section. After describing our data model, the two visual components—the representation of hierarchically structured graphs as well as time and comparison transitions between them—are presented individually. Then we describe our sorting strategy that aims at minimizing the number of crossing transitions within and between the sequences. Finally, we summarize the interaction techniques supporting the data analysis.

4.1 Data Model

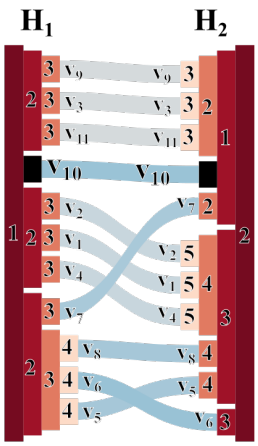
A dynamic graph $\mathscr{G} := (G_1, \ldots, G_T)$ is defined as a sequence of T subsequent graphs G_t. We model each of these graphs as a directed weighted graph $G_t = (V_t, E_t)$ that consists of a set of vertices V_t and a set of edges $E_t \subseteq V_t \times V_t$. Thereby, each edge $e_j \in E_t$ describes a tuple of two vertices (v_{i_1}, v_{i_2}) and each edge is assigned a weight $w_{e_j} \in \mathbb{R}^+$. The number of vertices $|V_t|$ may change over time. In the following, V denotes the global set of vertices containing all vertices v in any V_t.

An associated hierarchical group structure $\mathscr{H} := (H_1, \ldots, H_T)$ of the dynamic graph $\mathscr{G}$ is defined as a sequence of T subsequent hierarchies H_t. The hierarchical group structure H_t of the vertices $v \in V_t$ can be defined as a family of sets $H_t = \{S_0, S_1, \ldots, S_L\}$, where each S_{l_1} is a set of other group elements $S_{l_2} \in H_t$, with $l_1 \neq l_2$ or graph vertices $v_i \in V_t$. These groups represent the elements of a hierarchy where S_0 forms the root element and the vertices $v_i \in V_t$ the leaves. For all S_l from H_t where $l = 1, \ldots, L$, there is exactly one parent group $S_{l'} \in H_t$ with $S_l \in S_{l'}$; since also each graph vertex is contained in exactly one parent group, the same applies to all $v_i \in V_t$ ($\forall v_i \in V_t \; \exists!\; l' \in \{0, \ldots, L\} : v_i \in S_{l'}$).

Our visualization technique supports the visual comparison of M different sequences of hierarchical group structures $\mathscr{H}_m$, with $m = 1, \ldots, M$, for the same dynamic graph $\mathscr{G}$. Therefore, the comparison of hierarchical group structures H_t within a sequence $\mathscr{H}_{m_1}$ (comparing $H_{m_1,t}$ with $H_{m_1,t+1}$) and between sequences $\mathscr{H}_{m_1}$ and $\mathscr{H}_{m_2}$ (comparing $H_{m_1,t}$ with $H_{m_2,t}$) is supported. Such sequences could be the result, e.g., of different clustering algorithms. Also the comparison of a local (time-dependent) with a global (static) hierarchical group structure is conceivable. If $\mathscr{H}$ is static, all $H_t \in \mathscr{H}$ are identical.

4.2 Dissimilarity Computation

The comparison of hierarchies within and between sequences can be supported on a higher level using a dissimilarity metric. In the context of software clustering, different global metrics for hierarchy similarity were suggested, e.g., based on edit operations similar to the Levenshtein distance [46]. The END framework [47] is based on similarity metrics that estimates the distance between two flat decompositions and compares the hierarchies level by level. TreeJuxtaposer [25] and other comparison approaches [48] calculate the similarity of two internal nodes of the hierarchies using the set-based Jaccard coefficient. As only the leaves are included in the score calculation, the hierarchical structure is ignored. Consequently, two subgroups containing the same leaves



v_i	$p(v_{10},v_i,H_1)$	$p(v_{10},v_i,H_2)$	$c_1(v_{10},v_i,H_1)$	$c_2(v_{10},v_i,H_2)$
v_1	3	5	0.50	0.25
v_2	3	5	0.50	0.25
v_3	3	3	0.50	0.50
v_4	3	5	0.50	0.25
v_5	4	4	0.33	0.33
v_6	4	3	0.33	0.50
v_7	3	2	0.50	1.00
v_8	4	4	0.33	0.33
v_9	3	3	0.50	0.50
v_{11}	3	3	0.50	0.50
cos similarity $(\vec{c_1},\vec{c_2})$:				0.90
dissim(v_{10},H_1,H_2):				0.10

Fig. 4. Illustration of the calculation of the dissimilarity between the integration of v_{10} within the two hierarchies H_1 and H_2. Left: Transitions between H_1 and H_2, where the path lengths p from V_{10} to any other v_i or internal node S_l are included in the layered icicle plot. Right: Summary of path lengths p and closeness values $c = 1/(p-1)$. The dissimilarity is based on the cosine similarity between $\vec{c}_1$ and $\vec{c}_2$.

are classified as similar even if their structure may differ substantially. Bremm et al. [6] extended this score by incorporating the inner nodes of the hierarchical structure. All these metrics support only the calculation of the distance (similarity) between two hierarchies or subgroups but do not describe the difference of integration of a vertex into two hierarchies, i.e., where the vertex is positioned in the hierarchy.

In contrast, the novel metric we designed is able to compute a dissimilarity value for each vertex v describing the difference of containment of v in two hierarchies (hence, transition edge). In particular, the metric can be used to compare the integration of a vertex $v \in V_t \cup V_{t+1}$ into two hierarchies $H_{m,t}$ and $H_{m,t+1}$ (or $v \in V_t$ into $H_{m_1,t}$ and $H_{m_2,t}$). For simplification, these hierarchies are only referred to as H_1 and H_2, with V_1 and V_2 as the vertex sets of the respective graphs G_1 and G_2 (for $H_1 = H_{m_1,t}$ and $H_2 = H_{m_2,t}$, $G_1 = G_2$). This dissimilarity can be described based on the cosine similarity:

$$\text{dissim}(v, H_1, H_2) = 1 - \frac{\overrightarrow{c_1} \cdot \overrightarrow{c_2}}{|\overrightarrow{c_1}| \, |\overrightarrow{c_2}|},$$

where $\overrightarrow{c_1}$ and $\overrightarrow{c_2}$ are vectors of length $|V'| - 1$ with $V' = |V_1 \cup V_2|$. Each component of these vectors describes the closeness of v to all $v_i \in V_1 \cup V_2$, with $v \neq v_i$. The closeness between two vertices v_{i_1} and v_{i_2} in a hierarchy H is defined as $c(v_{i_1}, v_{i_2}, H) = 1/(p-1)$, where p describes the length of the shortest path in H from v_{i_1} to v_{i_2}. The shortest possible path length is therefore $p = 2$ and the longest path length is $p = 2(d-1)$, where d is the depth of the hierarchy H. In case that v_{i_1} or v_{i_2} is not contained in $V_1 \cap V_2$, p is set to the maximum path length. We use $1/(p-1)$ as closeness metric, to maximize the possible value range and to increase the weight on the direct neighbors of v. Hence, using this metric, changes within any subhierarchy S_l, with $v \in S_l$, have a stronger influence on the dissimilarity dissim(v, H_1, H_2) than changes in a subhierarchy S'_l, with $v \notin S'_l$.

Figure 4 illustrates the calculation of the dissimilarity for vertex v_{10} between two hierarchies of depth $d = 4$. The dissim-

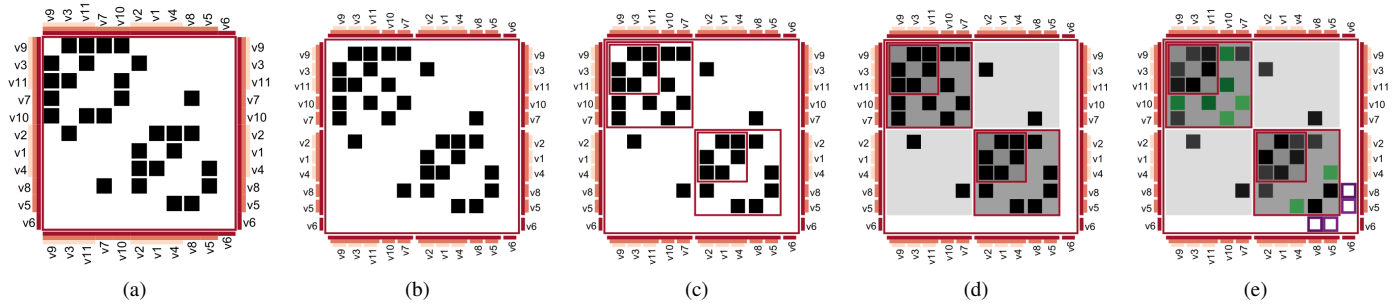


Fig. 5. Each graph G_t is visualized using an adjacency matrix, where H_t is visualized as indented icicle plot attached to the matrix (a). To encode the hierarchical group structure H_t within the matrix, identation (b) and nested contours (c) are used. (d) The density of edges within and between groups of the hierarchical group structure H_t is mapped to the gray value of the respective areas in the matrix. (e) Edges added (removed) compared to the last time point G_{t-1} are highlighted in green (purple).

ilarity metric for a single vertex v can be used to compute the dissimilarity between two hierarchies:

$$\text{dissim}(H_1, H_2) = \frac{1}{|V'|} \sum_{n=1}^{|V'|} \text{dissim}(v_i, H_1, H_2).$$

4.3 Matrix–Hierarchy Visualization

Our matrix visualization incorporates three components together with the graph topology G_t itself: the hierarchical group structure of G_t, edge densities within and between groups, and topological changes compared to the previous graph G_{t-1}.

Hierarchical group structure: To analyze the consistency between the graph topology G_t and the hierarchical group structure $H_{m,t}$, both are integrated within a matrix representation (Figure 5(a)), where vertices v_i are represented by rows and columns. The vertex order depends on the hierarchical structure $H_{m,t}$. Each edge $e_j := (v_{i_1}, v_{i_2})$ is encoded by a square within the matrix. The hierarchical group structure H_t of a graph G_t is visualized using icicle plots attached to the matrix but also within the matrix using nesting. First, the rows and columns are indented with respect to their hierarchical group structure (Figure 5(b)). Second, the internal nodes $S_l \in H_t$ are visualized as contours surrounding the respective rows and columns (Figure 5(c)). The contours representing the group elements S_l are indented with respect to their depth in the hierarchy. Within both the matrix and icicle plot, the hierarchical group structure is encoded with red, where within the icicle, the depth of the internal nodes of the hierarchy is mapped to brightness (Figure 6).

Edge density: To support the analysis of the consistency between the graph topology and the hierarchical group structure, the density of relations within and between groups S_l of the hierarchy are encoded within the matrix (Figure 5(d)). The density $\rho_l = |E_l|/(|V_l|^2 - |V_l|)$ with V_l the set of leaves of group S_l and E_l the set of edges among vertices in V_l is mapped to a gray value used to fill the contour for S_l: groups S_l with $\rho_l = 0$ ($\rho_l = 1$) are white (dark gray). The groups S_l of the hierarchy show up as squares on the diagonal. To analyze how strongly a graph is coupled and how far subgroups are connected, for any two groups S_{l_1} and S_{l_2} that have the same parent group, we also visualize the density $\rho_{l_1,l_2} = |E_{l_1,l_2}|/(|V_{l_1}||V_{l_2}|)$ with V_{l_1} (V_{l_2}) the set of leaves of group S_{l_1} (S_{l_2}) and E_{l_1,l_2} the set of edges that connect a leaf of S_{l_1} with a leaf of S_{l_2}, or vice versa. Both densities ρ_l and ρ_{l_1,l_2} are based on the definition of the density in directed graphs. We decided for this restriction to not fragment the adjacency matrix too much. The density ρ_{l_1,l_2} is mapped to the rectangle described by the two groups S_{l_1} and S_{l_2} using the same gray-scale mapping as for ρ_l.

Using our visual mapping of the density within and between groups, we can immediately analyze if the hierarchical group structure matches the graph topology. If the hierarchical group structure is derived based on a clustering algorithm, the squares on the diagonal will be much darker in comparison to the surrounding rectangles. The reason is that clustering algorithms usually aim at detecting groups and subgroups of vertices v_i that are more densely connected among each other than with vertices of other groups.

Topological changes: To support the tracking of changes in the graph topology, within each of the graphs G_t with $t > 1$, the differences to the previous graph G_{t-1} are encoded explicitly if activated by the user. Edges that were added or removed compared to the previous point in time $(t-1)$ are highlighted using green or purple, respectively (Figure 5(e)). The weight w_{e_j} of an edge is mapped to the brightness of the cell. For edges that remain, w_{e_j} is mapped non-linearly to a gray scale . For added or removed edges, the edge weights w_{e_j} are mapped in similar fashion as for the gray scale, adapting not only value, but also saturation. In contrast to the remaining and added edges (shown as filled squares), for removed edges, we only draw the border of the square. Alper et al. [49] use the inner and outer area of the cell to encode the edge weights of the two graphs using the same color map. Our technique aims at highlighting points in time at which many edges are added or removed.

4.4 Transitions

Our visualization technique shows the dynamic graph $\mathscr{G} := (G_1, \ldots, G_T)$ using a timeline approach [3], where the matrix–hierarchy visualizations are arranged horizontally (Figure 3). If more than one sequence of hierarchical group structures $\mathscr{H}_m$ is defined, M sequences (timelines) representing $\mathscr{G}$ are stacked vertically. As mentioned before, the number of vertices $|V_t|$ may change over time. If $|V_t| < |V|$, the matrix is scaled such that it fills the maximum available space.

Our visualization approach aims at the visual analysis of changes in the graph topology as well as the hierarchical group structure. With the graphs next to each other, it is still difficult to compare the hierarchical group structure between neighboring graphs, although the hierarchical group structure is encoded within each of the graph representations. To facilitate the analysis of changes in the hierarchy, we use an edge-drawing approach for the comparison of hierarchies [22] that are visualized as icicle plots

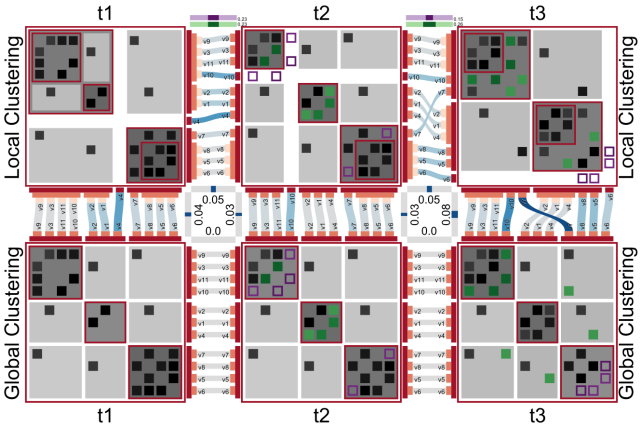


Fig. 6. Dynamic graph ($n = 11$ and $T = 3$) visualized together with its local (dynamic) and global (static) hierarchical group structure. The transitions of adjacent hierarchical group structures H_1 and H_2 are shown as cubic Bézier curves, where the dissimilarity $\text{dissim}(v, H_1, H_2)$ is mapped to the color of the curves. The bars spanning between neighboring matrices visualize the dissimilarity $\text{dissim}(H_1, H_2)$ between the hierarchies. The green (purple) bars at the top show the percentage of added (removed) edges from G_t to G_{t+1}.

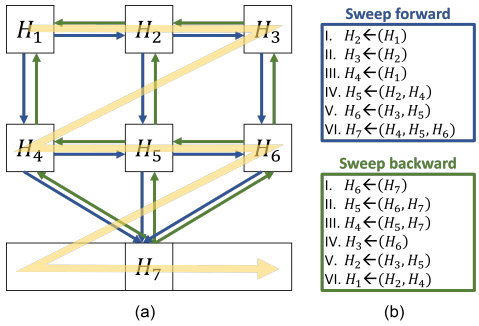


Fig. 7. Illustration of the sorting algorithm. (a) A dynamic graph with $T = 3$ and $M = 3$ sequences $\mathscr{H}_m$, where the hierarchies in $\mathscr{H}_1$ and $\mathscr{H}_2$ change over time and $\mathscr{H}_3$ models a global hierarchy. Here, the hierarchical group structures $H_{m,t}$ are named based on their global hierarchy IDs H_i. The hierarchies H_i are scanned line by line. (b) Based on the sweep, we derive the sequence of sorting steps, where each hierarchy has one or more dependencies illustrated by links in (a).

attached to matrices such that edges (transitions) can be drawn in the space between the matrices (Figure 6).

Using an edge-drawing approach, time and comparison transitions between two hierarchies $H_{m,t}$ and $H_{m,t+1}$ or $H_{m_1,t}$ and $H_{m_2,t}$—in the following only referred to as H_1 and H_2—are visualized explicitly (Figure 6). In particular, each transition from is drawn using a cubic Bézier curve. In doing so, we generate smooth transitions between successive hierarchical group structures . The dynamic graph together with the transitions of hierarchical group structures is reminiscent of a flow—as already used for group evolution in other contexts [35], [36], [50]—employing a timeline from left to right where vertices v literally flow from one hierarchically structured graph to the other. Similary, rows are connected. Therefore, our visualization of transitions helps track a vertex and its relations.

To help identify vertices v whose placement within neighboring hierarchies H_1 and H_2 changes, the dissimilarity $\text{dissim}(v, H_1, H_2)$ is mapped non-linearly to the color of the curves using a blue single-hue color map (Figure 6). To make vertices with high dissimilarity visible even when many edge crossings occur, the time and comparison transitions are sorted and drawn as opaque curves in ascending order based on their associated dissimilarity value. The total dissimilarity $\text{dissim}(H_1, H_2)$ between two neighboring hierarchies H_1 and H_2 is visualized by a bar that spans between the two matrices (Figure 6). The dissimilarity value is thereby mapped to the length of a blue bar.

The visualization of the dissimilarities on vertex and hierarchy level helps immediately detect transitions, between neighboring hierarchies within or between sequences, where many changes occur. The exact dissimilarity values for the vertices are available via tooltips. Hence, the dissimilarity can be used for the identification and analysis of differences in the hierarchical group structure on the higher level, before drilling down and investigating the differences between the two graphs and their hierarchical group structures in more detail.

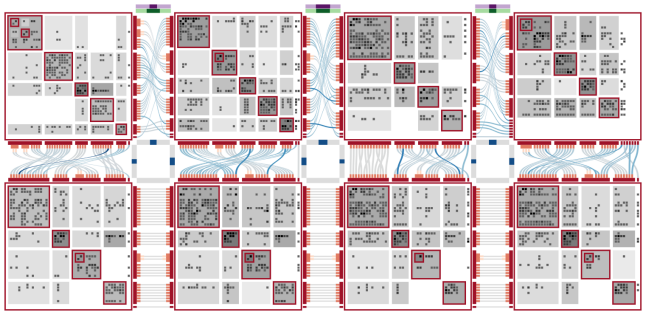
The total amount of added p_{add} and removed p_{rem} edges between two successive graphs G_{t-1} and G_t is visualized by two additional bars that span between the two matrices just above the first sequence (Figure 6). The amount of added edges $p_{\text{add}} = |E_{\text{added}}|/|E_t|$ describes the percentage of edges in G_t that are not part of G_{t-1}. Vice versa, $p_{\text{rem}} = |E_{\text{removed}}|/|E_{t-1}|$ describes the percentage of edges in G_{t-1} that are not contained in G_t. The value p_{add} (p_{rem}) is mapped to the length of a dark green (purple) bar.

4.5 Sorting of Vertices

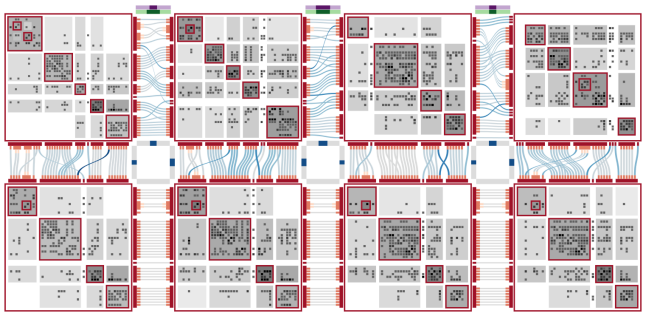
The hierarchy transitions are supposed to improve the analysis of changes or differences between hierarchical group structures. However, the readability of the visualization suffers with increasing number of crossing time and comparison transitions. To address this problem, we integrate an extended sorting algorithm that minimizes edge crossings within and between sequences. There are sorting algorithms that perform edge crossing minimization between two non-binary hierarchies [24]. Only few techniques extend this approach to a sequence of hierarchies. Dwyer and Schreiber [31] apply an edge crossing minimization to their dynamic phylogenetic tree visualization, but their approach can only be applied to a single sequence of binary trees. To the best of our knowledge, there are no techniques that sort dynamic non-binary hierarchies for more than one sequence. In contrast, our sorting algorithm allows us to sort non-binary hierarchies within and between different sequences $\mathscr{H}_m$.

In our visualization, the different sequences of hierarchically structured graphs are not visualized independently but are also connected via comparison transitions. Therefore, it is not sufficient to sort each of the sequences individually. Instead, our sorting algorithm considers all within- and between-sequence transitions, sweeping through all sequences by scanning them line by line (indicated by the yellow arrow in Figure 7(a)).

Figure 7(a) shows an example of a dynamic graph with $T = 3$ and $M = 3$ sequences $\mathscr{H}_m$, where the hierarchies $H_{1,t} \in \mathscr{H}_1$ and $H_{2,t} \in \mathscr{H}_2$ change over time and $\mathscr{H}_3$ models a global hierarchy defined over the complete dynamic graph $\mathscr{G}$. As $\mathscr{H}_3$ models a global (static) hierarchy, all hierarchies $H_{3,t} \in \mathscr{H}_3$ should have the same order of vertices. Therefore, they are incorporated only once (H_7) in the sorting process with the respective dependencies. Dependencies between hierarchical group structures occur between any two neighboring hierarchies of the sequences, i.e., between



(a) Without sorting (1,778 crossing transitions)



(b) With sorting (257 crossing transitions)

Fig. 8. Our visualization technique without (a) and with (b) the applied sorting algorithm to minimize the number of crossing transitions.

any two hierarchies $H_{m,t}$ and $H_{m,t+1}$ or $H_{m,t}$ and $H_{m+1,t}$ (indicated by the visual links in Figure 7(a)).

To sort all hierarchies taking into account all dependencies, the sweep is performed in both directions—doing a forward and backward sweep. The algorithm first performs a forward sweep (indicated by blue arrows in Figure 7(b)) from the first hierarchy H_1 to the last hierarchy H_7, which is followed by a backward sweep (indicated by green arrows). At each sorting step (e.g., I. $H_2 \leftarrow (H_1)$), we apply a sorting technique similar to the approach by Holten and van Wijk [24] to reorder the leaves of the depending hierarchy (H_2) based on the reference hierarchy (H_1). The sorting is based on the Sugiyama-style crossing reduction technique [51], where the nodes of dependent hierarchy are reordered with respect to matching nodes in the reference hierarchy.

The crossing reduction is performed per hierarchy level using the barycenter method. The algorithm, thereby, starts with the root element and recursively sorts the subsets until the leaves are reached. Due to our comparison not only along but also across sequences, a hierarchy can have more than one dependency (e.g., IV. $H_5 \leftarrow (H_2, H_4)$). Therefore, we extend the standard barycenter method that takes only the index positions of one hierarchy into account such that different reference hierarchies $\mathscr{H}_{\text{ref}} := \{H_{i_1}, \ldots, H_{i_K}\}$ with $K = |\mathscr{H}_{\text{ref}}|$ are considered. This is done by taking the average of the respective index positions from all reference hierarchies, where the indexing is independent in each hierarchy, starting with 1.

The sweep is performed in both directions several times—each time starting with the result from the last run. After each run, the number of crossing transitions is computed. The sorting algorithm stops after a maximum number of 100 runs. As a resulting vertex ordering, we take the one with the least number of crossing transitions. Figure 8 shows the friendship network from Figure 2 together with a dynamic and a global (static) hierarchical group structure. The initial number of 1,778 crossing transitions could be reduced to 257 by applying our sorting algorithm.

4.6 Interaction Techniques

Our visualization technique facilitates the analysis of changes and differences in the hierarchical group structure as well as the consistency between the graph topology and hierarchical group structure. Both can be performed on the higher level (zooming out) and on the lower level (zooming in and panning). The hierarchy dissimilarities, the hierarchical structures, as well as the edge densities within and between groups can be perceived even when the visualization is scaled down such that the matrix cells are only few pixels wide. To gain further insight and to analyze changes in more detail, other interaction than just zooming and panning is necessary, at least for large data sets.

Vertex labels can be displayed if desired but are also available as tooltip when hovering over a vertex v_i, i.e., a leaf node in the icicle or transition between hierarchies. For the latter, the tooltip also contains the dissimilarity mapped to the color of the transition. The tooltips on edges can be used to determine the source and the target vertex as well as the exact weight of that edge.

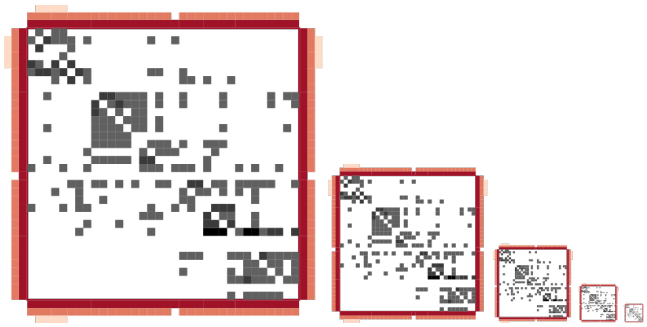
Our visualization approach facilitates brushing and linking: by selecting a single vertex or a group of vertices, their course is highlighted through all sequences using yellow ribbons (Figure 2). Using this highlighting, we can easily identify points in time when vertices or groups emerge or disappear and how they change with respect to their integration into hierarchical group structures. A video demonstrating the interaction techniques is available in the supplemental material.

5 Scalability and Limitations

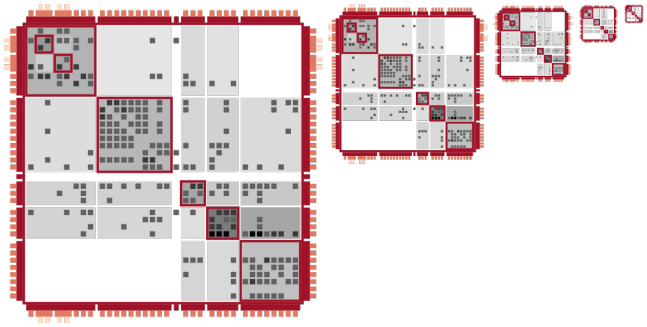
A general challenge in information visualization is visual scalability: even with an increasing number of elements, the generated visualization should remain readable.

Graph size: In static graphs, the number of nodes and the density of edges are two crucial parameters for scalability. As shown by several scalability comparisons, matrix-based representations are better suited than node-link diagrams for most tasks on large and dense graphs but path-related tasks [52], [53]. The hierarchical group structure of the graph improves the scalability with respect to the size of the graph as it can be used to abstract the set of nodes. In the traditional adjacency matrix representation, the hierarchy is used to sort the vertices and shown by a tree visualization such as a layered icicle plot attached to the matrix. However, perceiving the hierarchical group structure using this matrix representation becomes difficult with increasing number of nodes because fewer pixels are available for each vertex and edge (Figure 9(a)).

In contrast, our matrix–hierarchy visualization additionally encodes the hierarchical group structure visually within the matrix using indentation and contours (Figure 9(b)). When decreasing the size of the matrix while zooming out, the thickness of the contours and the indentation within the matrix stays constant, hence, increases relative to the size of the matrix. In this way, the hierarchical group structure can also be perceived at a coarse level. Moreover, our visual mapping of the edge density within and between subgroups also improves the scalability. When scaling the visualization such that matrix cells are only a few pixels wide or less, edges become barely readable. However, the density within



(a) Traditional adjacency matrix with icicle plots attached



(b) Our matrix–hierarchy visualization

Fig. 9. Comparing a traditional adjacency matrix (a) with our matrix visualization technique (b) that integrates the hierarchical group structure in the matrix. The matrices are shown at different zoom levels to demonstrate how the approaches scale. All matrices show the same hierarchically structured graph.

and between groups can still be perceived, giving the impression of a partially aggregated graph.

Time: Besides the scalability with respect to the size and density of the graph, the number of points in time needs to be considered as a further dimension [54]. A limitation of our technique is that the diagrams become very elongated with increasing number of points in time T. Figure 10(a) shows a graph with $T = 21$ points in time and at most 146 vertices per graph G_t. Scaling down this diagram such that the complete data set is visible on the screen, individual relations of the graphs and transitions between the hierarchical group structures cannot be perceived. However, using the bars that visualize the dissimilarity of two successive hierarchical group structures $H_{m,t}$ and $H_{m,t+1}$ and the number of added or removed edges from G_t and G_{t+1}, we can identify time transitions $t \rightarrow t+1$ with severe changes in hierarchical group structure and graph topology.

The dissimilarity can be used to analyze differences in the hierarchical group structure on the higher level and to identify successive points in time with strong differences. The user can then drill down and investigate the differences between two graphs of interest and their hierarchical group structures in more detail. To this end, the dissimilarities $\text{dissim}(v, H_{m,t}, H_{m,t+1})$ can be used to aggregate intervals $(t_1, \ldots, t_k)$ (with $t_1 < t_k$) within the dynamic graph $\mathscr{G}$. If only one sequence of hierarchical group structures $\mathscr{H}_1$ is defined, we can use a threshold θ_{dissim} and aggregate all graphs G_t of an interval $(t_1, \ldots, t_k)$, with $\text{dissim}(v, H_t, H_{t+1}) < \theta_{\text{dissim}}$ for all $t \in (t_1, \ldots, t_k)$. For multiple sequences (i.e., $M > 1$), we can use the minimum or maximum norm or the average of the M dissimilarity values as overall hierarchy dissimilarity between two successive points in time t and $t+1$. This overall hierarchy dissimilarity can then be used to partially aggregate the dynamic graph based on the defined threshold θ_{dissim}.

Figure 10(b) shows the result of aggregating intervals within the dynamic graph from Figure 10(a) that are stable with respect to the hierarchical structure. The aggregation was done using the minimum norm and a threshold $\theta_{\text{dissim}} = 0.08$. When aggregating subsequences from a dynamic graph, different representations can be used for the aggregate, as in MultiPiles [14], e.g., showing the mean, trend, variation, or adjacency differences of the graph subset. We decided to use the mean of the subset $(G_{t_1}, \ldots, G_{t_k})$.

When aggregating dynamic hierarchically structured graphs, we also need to consider how to aggregate the hierarchies $(H_{m,t_1}, \ldots, H_{m,t_k})$. The hierarchies can either be merged manually or the aggregated graph needs to be clustered again. As the aggregation of hierarchies and hierarchical clustering of graphs was done by external tools, the whole aggregation procedure was not yet done interactively within our prototype but was outsourced as a preprocessing step. For the data set visualized in Figure 10(b), two different hierarchy aggregation methods were applied: for the hierarchical clustering sequence, the aggregated graph was clustered again; for the package structure sequence, the package structures were merged manually.

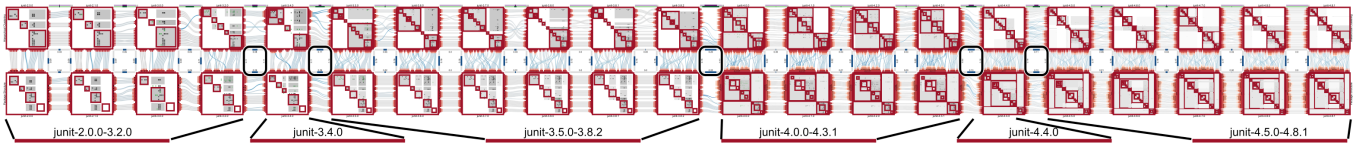
Both aspects—graph size and time—have an influence on the scalability and interact. The larger the graphs, the fewer graphs fit on the screen without zooming out. Besides the temporal scalability, our technique has some other limitations. The combination of the components we integrated into our hybrid (Sections 4.3 and 4.4) require a learning process for the user. Furthermore, if more than two sequences are compared, the result of the sorting algorithm depends on the sequence order and the ordering per sequence will not be optimal. In addition, non-adjacent matrices cannot be compared. Curved transitions might be misinterpreted as changes in the hierarchies, but could be only an artifact of sorting. Similarly, straight transitions may appear even if the hierarchical integration of that vertex changes. However, the color mapping of the dissimilarity provides means to better distinguish false from true changes in the hierarchies.

6 Case Studies

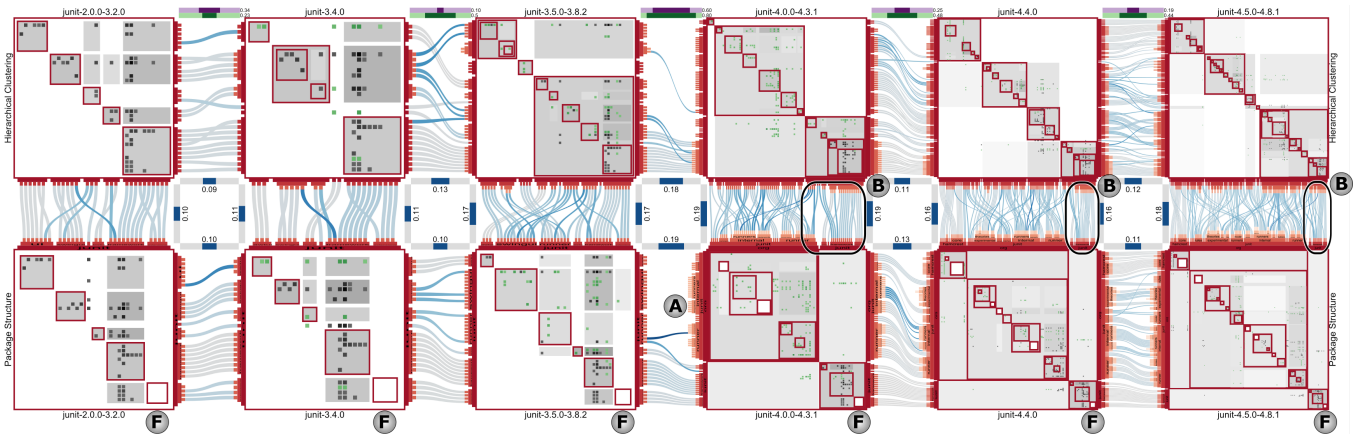
We applied our visualization technique to two different application scenarios to show its utility. Within a case study on software releases, we explore the evolution of dependency graphs and the consistency between the topology-based hierarchical group structure and the package structure of the software system. In a case study on international soccer matches, we use different hierarchical group structures to analyze local patterns that are visible in either one—but not all—of the hierarchical group structures.

6.1 Visualization of Software Evolution

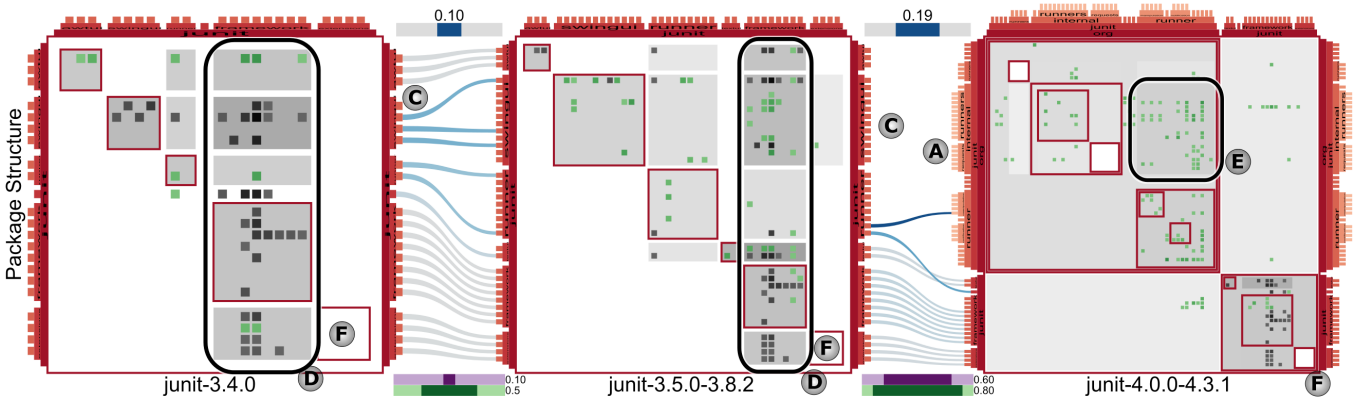
The first data set shows the evolution of a software system as a dynamic graph. The structural dependencies (i.e., method calls and usage of variables, inheritance, aggregation, etc.) were extracted using *DependencyFinder* (http://depfind.sourceforge.net). The vertices of the graph model classes and edge weights represent the number of dependencies between two classes. The package structure of each revision provides a natural dynamic hierarchy on the classes. To analyze the suitability of the package structures, we compare them to a dynamic hierarchical group structure derived using the hierarchical clustering algorithm by Rosvall and



(a) JUnit framework including all 21 releases of the data set with at most 146 vertices per point in time



(b) Call graphs after aggregating subsets of graphs whose dissimilarity in the hierarchical clustering and package structure is less than 0.08



(c) Zoom into (b) showing a subset of three partially aggregated revisions from (b)

Fig. 10. Structural dependencies of the JUnit framework. The results of a hierarchical clustering of the classes is compared to the package structure of the framework.

Bergstrom [55]. We look at changes in the hierarchical group structure and in the graph topology, i.e., how the classes within and between groups depend on each other in different releases. Software developers might use our visualization approach to understand their software system and its history. They may use it to analyze when particular packages and dependencies were introduced, or to identify design flaws like unwanted dependencies between packages.

In this case study, the JUnit regression testing framework (http://junit.org)—including 21 releases—is used as an example (Figure 10(a)). The aim of this case study was to (1) identify phases of changes in the hierarchical group structure and graph topology and (2) analyze the consistency between the package structure and the graph topology of individual revisions. Scrolling through time and investigating the dissimilarity bars as well as the amount of added and removed edges visualized by further bars made clear that the changes in the graph topology and hierarchical group structure are only minor between many releases. Based on the aggregation procedure described in the previous section, we extracted a dynamic graph over six partially aggregated releases applying a threshold of $\theta_{\text{dissim}} = 0.08$ (Figure 10(b)). In Figure 10(a), temporal transitions with a hierarchy dissimilarity of $\theta_{\text{dissim}} \geq 0.08$ are highlighted by black boxes. The strongest change occurs between revision *3.8.2* and *4.0.0*.

The visualization shows that the JUnit framework constantly grows, where the most substantial changes are introduced with revision *4.0.0* as the package `org` is introduced (Figures 10(c) A) next to the main package `junit`, which was the only main package in revisions *3.**. The package `junit` is loosely coupled with classes in different subpackages of `org.junit`. In contrast, in the *Hierarchical Clustering* sequence, the package `junit` is integrated into a bigger hierarchical cluster together with several classes from `org.junit` (Figure 10(b) B). This is just one difference that contributes to the high dissimilarity between the two hierarchical group structures for the graph *4.0.0-4.3.1*. Across the revisions, more and more packages and classes are introduced and relatively few removed, e.g., the packages `junit.ui`, `junit.awtui`, and `junit.swingui` (Figure 10(c) C).

Across all revisions, some subpackages are coupled as dependencies between them exist. Classes of the package `junit.framework`, for instance, are strongly coupled with classes of other `junit` packages (Figure 10(c) D). Moreover, there are

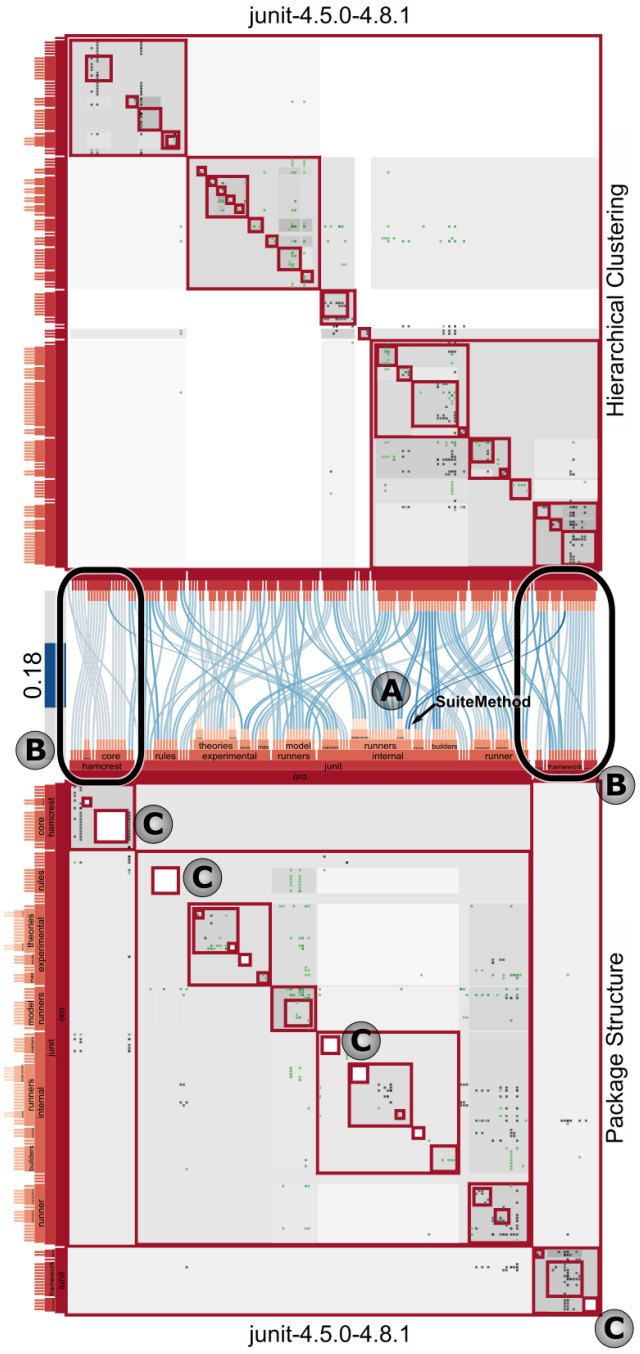


Fig. 11. Comparing the results of a hierarchical clustering on the classes to the package structure of the aggregated JUnit revisions *4.5.0-4.8.1*.

many method dependencies from classes of `org.junit.internal` to classes of `org.junit.runner` (starting with revision *4.0.0*; e.g., Figure 10(c) E).

In the first sequence (*Hierarchical Clustering*), classes are clustered hierarchically based on their dependencies. A high dissimilarity between the hierarchical group structures—*Hierarchical Clustering* and *Package Structure*—may point out design flaws and can be used to detect unwanted dependencies between packages. Looking at the dissimilarity of the hierarchical integration of vertices (blue curves), we can identify classes that should be moved into a different package, e.g., the class `SuiteMethod` of the package `org.junit.internal.runners` (Figure 11 A). However, some packages are presented relatively well within the *Hierarchical Clustering* sequence, e.g., `org.hamcrest` and `junit` in revision *4.5.0* (Figure 11 B). An artifact that we detected using the *Package Structure* sequence is the occurrence of several packages whose classes are only related to classes of other packages but not among each other; they show up as white squares along the diagonal, e.g., subpackages of `org` including `org.junit.runners`, `org.junit.internal.requests`, `org.hamcrest.core`, and others (Figure 11 C). Classes within `junit.extensions`, e.g., only depend on classes within `junit.framework` during all revisions (Figure 10(c) F). Within the *Hierarchical Clustering* sequence, the classes of these two packages are clustered to one subhierarchy. Hence, it may be reasonable to combine both.

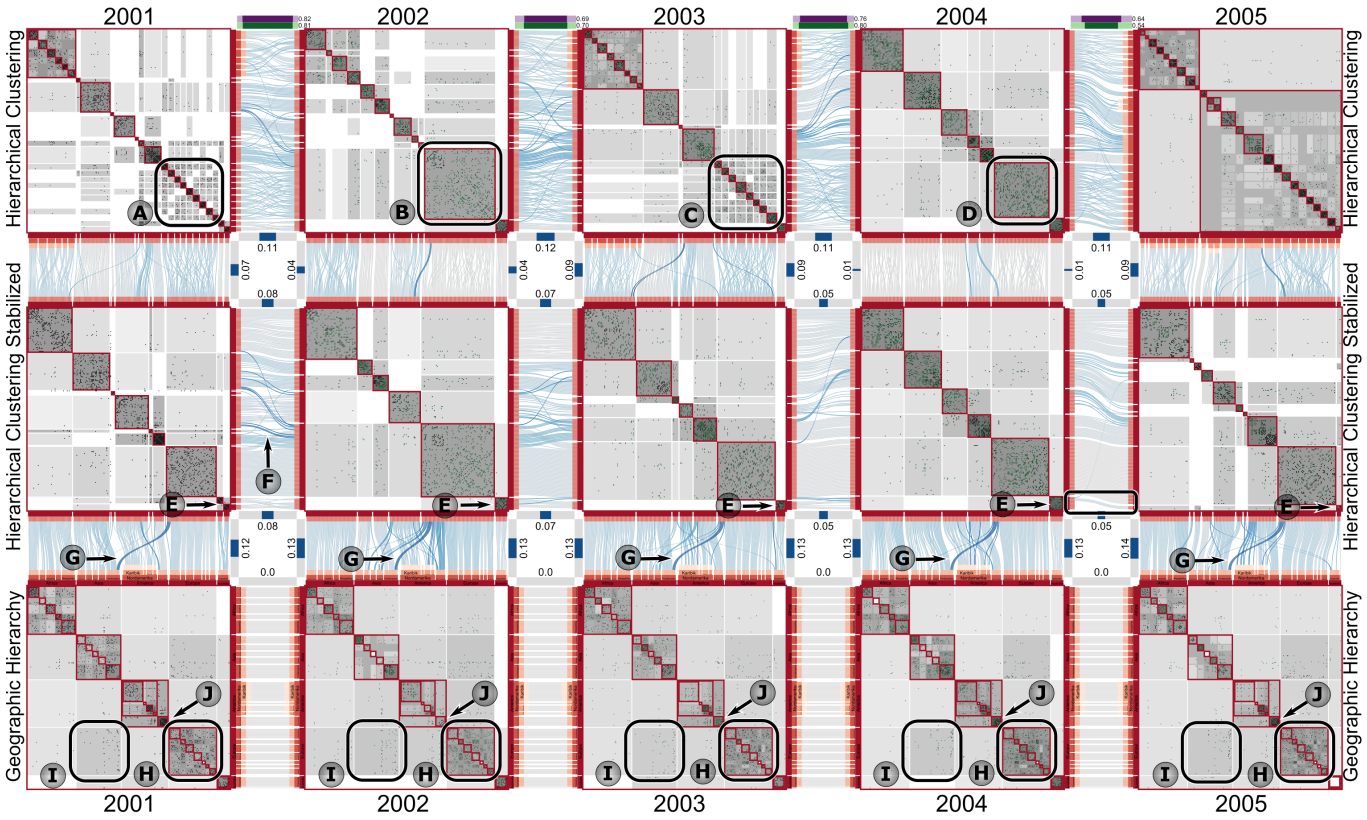
Using our visualization, we can analyze the consistency between the package structure of a software system and the hierarchical clustering based on the graph topology over time. Based on the visual mapping of hierarchy dissimilarities, we can identify points in time of substantial changes as well as versions with strong differences between the two hierarchical group structures. The consistency between package structure and graph topology can be analyzed using our mapping of the density of dependencies within and between sub-packages. Hence, we can identify design flaws and identify aspects to improve the software system.

6.2 International Soccer Matches

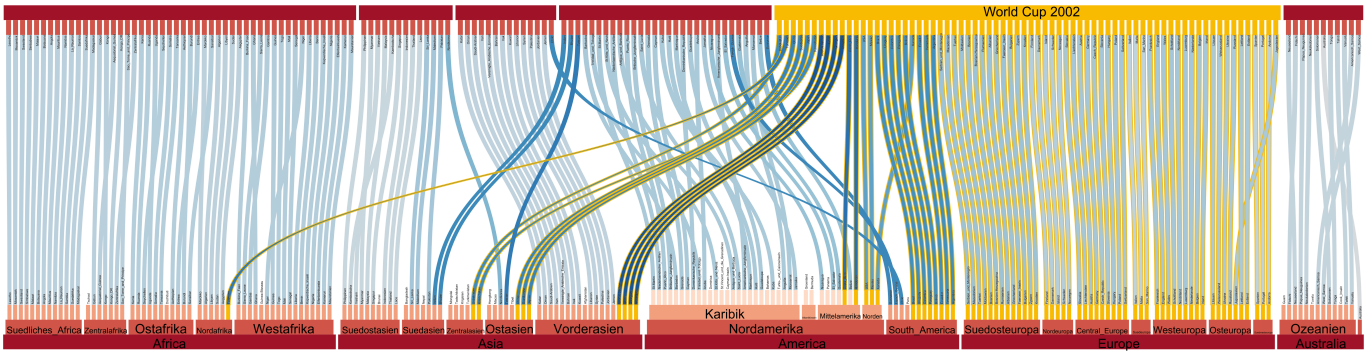
In the second case study, we apply our visualization technique to a dynamic graph to compare the global (static) hierarchical group structure of a graph with its local (time-dependent) hierarchical group structures. The dynamic graph summarizes world-wide international soccer matches of 219 national teams between 1992 and 2005. In each of the graphs G_t, two teams are related if they played against each other in year t. Hence, the graphs G_t are undirected and the matrices are symmetric.

We use our visualization approach to visualize three different hierarchical group structures $\mathscr{H}$: (1) The local group structure H_t (*Hierarchical Clustering*) derived for each year individually using the hierarchical clustering algorithm by Rosvall and Bergstrom [55]. (2) The local but stabilized group structure H_t modeling an inertia of the hierarchical groups by incorporating the hierarchical group structure of the previous graph H_{t-1} (*Hierarchical Clustering Stabilized*). This is done by clustering the graphs G'_t, instead of G_t, where each G'_t incorporates for all pairs of vertices $e_j = (v_{i_1}, v_{i_2})$ the length of the shortest path p_{i_1,i_2} in hierarchy H_{t-1} into the edge weight w'_{e_j}. The lengths are combined with the edge weight w_{e_j} of G_t to form the edges of G'_t with weight $w'_{e_j} = w_{e_j} + \alpha p_{i_1,i_2}{}^{-\beta}$, where α is used to adjust the scale and $\beta > 0$ is used to adjust the rate of decrease of the second term. For our data set, we decided to use $\alpha = 0.5$ and $\beta = 3.0$. (3) The global hierarchical group structure based on the geographic topology (*Geographic Hierarchy*). This hierarchical group structure is time-independent (static).

In contrast to the previous case study, the dissimilarities between successive hierarchies are relatively constant, ranging from 0.10 to 0.14. As the hierarchies are changing substantially during each time transition, we did not find a good threshold to partially aggregate subsequences. For space constraints, in the following, we concentrate on a subset of the dynamic graph including the years 2001–2005. In Figure 12(a), the first sequence (*Hierarchical Clustering*) can be used to identify and analyze temporal local



(a) International soccer matches from 2001 to 2005



(b) Several bundles of teams from Asia and America are merged into the World Cup cluster in 2002

Fig. 12. International soccer matches over 14 years comprising 219 national teams. (a) The dynamic graph is shown using three different hierarchical group structures: *Hierarchical Clustering*, *Hierarchical Clustering Stabilized*, and *Geographic Hierarchy*. Added edges are highlighted in green, which make up a huge amount in all years, which also becomes apparent by the bars at the top of the diagram. Hence, the difference in the graph topology is relatively high. (b) Comparison transition between the *Hierarchical Clustering Stabilized* and *Geographic Hierarchy* sequence at $t = 2002$.

group structures. In 2001, e.g., there are several small clusters along the diagonal (Figure 12(a) A) representing the qualification groups of the World Cup that happened in the following year (B). A similar pattern can be perceived 2003–2004, with the European Championship in 2004 occurring as relatively large cluster (C and D). By selecting a cluster, such as the World Cup cluster, we can easily track their members to the *Geographic Hierarchy* sequence and determine their composition from the different continents.

The *Hierarchical Clustering Stabilized* sequence is more suitable for the analysis of stable hierarchical group structures. The cluster containing Australia and several teams of Oceania, e.g., remains visible during all years (E) but becomes smaller from 2004 to 2005 as several teams are not part of G_{2005} any more. Moreover, the *Hierarchical Clustering Stabilized* sequence helps us identify the strongest changes in the hierarchical group structure as little changes are suppressed by the inertia. In the time transition from 2001 to 2002, several teams from Asia and America stick out as blue curves as they have a relatively high dissimilarity. These teams are merged into the World Cup cluster that mainly consists of European teams (F). The composition of teams during the World Cup in 2002 can also be analyzed investigating the comparison transition between the *Hierarchical Clustering Stabilized* and the *Geographic Hierarchy* sequence. Looking at the hierarchy comparison transitions, we can see several bundles of blue curves from Asia and America merged into the World Cup cluster (Figure 12(b)). One of these bundles from Asia, in particular from the Middle East, appears every year (G).

Finally, using the *Geographic Hierarchy*, we can analyze the coupling between the continents as well as the cohesion within regional subgroups. European teams (H) play many matches against each other in every year. The visualization of the density ρ_{l_1,l_2} (Section 4.3) shows that Europe is coupled strongly with Asia over all years, in particular with teams from the Middle East (e.g., I in 2004). Using this global hierarchy, it also appears that the teams from South America (J) build a very cohesive group over all years but the year of the World Cup (2002).

Our visualization technique can be used to analyze groups of national teams that played many matches against each other in particular years and how these groups changed over time using the *Hierarchical Clustering* and the *Hierarchical Clustering Stabilized* sequence. The visualization of the *Geographic Hierarchy* sequence, in contrast, helps analyze the number of matches of teams within and between continents and their subareas.

7 Conclusion and Future Work

We developed a visualization technique that shows dynamic hierarchical group structures together with their underlying dynamic graph topology. Our technique represents the first approach to visualize both dynamic components in a reasonable, scalable way. In contrast to other visualizations for dynamic graphs, our technique supports all three tasks that arise for hierarchical group structures in dynamic graphs: First, the consistency between the hierarchical group structure and the graph topology of individual graphs can be analyzed. Second, the changes of graph topologies can be tracked on a higher level using the bars and in detail using the encoding of added and removed edges. Not least, due to the transitions connecting hierarchies and the dissimilarity metric encoded in the diagram, it is possible to track the hierarchy changes over time, which is not possible with most other timeline approaches.

To improve the temporal scalability, we plan to integrate the aggregation of intervals as described in Section 5. This will allow users to interactively collapse and expand intervals using drag and drop interactions as described by Bach et al. [14]. Also their piling approach could be integrated into our visualization technique for up to two sequences by attaching the piles above the first and below the second sequence. In addition, the aggregation of subgroups within the graphs would increase the scalability with respect to the graph size. Since the hierarchical structures can change, the one-to-one relation of successive subgroups would not be necessarily given. The transitions as visualized by our technique would therefore need to be extended toward Sankey-like representations that allow splits and merges.

Moreover, our visual comparison technique can be used to compare hierarchical group structures of the same dynamic graph, e.g., different hierarchical clusterings or a time-dependent vs. a global hierarchical structure. Dissimilarities between any two hierarchical group structures become apparent by the dissimilarity bars and blue curves connecting the two hierarchies. Moreover, our visualization technique benefits from ordering internal nodes and leaves of the hierarchies. The ordering minimizes the number of crossing transition curves within each sequence but also between the sequences. Our sorting and coloring based on dissimilarities helps deal with the remaining crossings and to indentify outliers, i.e., vertices that switch into a different subhierarchy. We plan to extend the sorting algorithm to optimize the order of sequences, such that similar sequences of hierarchies are positioned next to each other. In addition, the possibility to interactively adjust the order would help users compare sequences.

Acknowledgments

This work was supported by DFG within SFB 716/D.5 and within project DFG WE 2836/6-1 "Visual analytics of static and dynamic networks taking into account uncertainty and fuzzy clustering".

References

[1] I. Herman, G. Melancon, and M. Marshall, "Graph visualization and navigation in information visualization: A survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, no. 1, pp. 24–43, 2000.

[2] C. Vehlow, F. Beck, and D. Weiskopf, "The state of the art in visualizing group structures in graphs," in *Eurographics Conference on Visualization - STARs*, ser. EuroVis. Eurographics Association, 2015, pp. 21–40.

[3] F. Beck, M. Burch, S. Diehl, and D. Weiskopf, "A taxonomy and survey of dynamic graph visualization," *Computer Graphics Forum*, 2016, to appear.

[4] J.-w. Ahn, C. Plaisant, and B. Shneiderman, "A task taxonomy for network evolution analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 3, pp. 365–376, 2014.

[5] N. Kerracher, J. Kennedy, and K. Chalmers, "A task taxonomy for temporal graph visualisation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 21, no. 10, pp. 1160–1172, 2015.

[6] S. Bremm, T. von Landesberger, M. Hess, T. Schreck, P. Weil, and K. Hamacherk, "Interactive visual comparison of multiple trees," in *Proceedings of the 2011 IEEE Conference on Visual Analytics Science and Technology*. IEEE, 2011, pp. 31–40.

[7] A. Telea and D. Auber, "Code Flows: Visualizing structural evolution of source code," *Computer Graphics Forum*, vol. 27, no. 3, pp. 831–838, 2008.

[8] G. Van de Bunt, M. van Duijn, and T. Snijders, "Friendship networks through time: An actor-oriented statistical network model," *Computational and Mathematical Organization Theory*, vol. 5, pp. 167–192, 1999.

[9] A. Ahmed, X. Fu, S.-H. Hong, Q. Nguyen, and K. Xu, "Visual analysis of history of World Cup: A dynamic network with dynamic hierarchy and geographic clustering," in *Visual Information Communication*. Springer, 2010, pp. 25–39.

[10] F. Reitz, M. Pohl, and S. Diehl, "Focused animation of dynamic compound graphs," in *Proceedings of the 13th International Conference on Information Visualisation*. IEEE, 2009, pp. 679–684.

[11] M. Burch, B. Schmidt, and D. Weiskopf, "A matrix-based visualization for exploring dynamic compound digraphs," in *Proceedings of the 17th International Conference Information Visualisation*. IEEE, 2013, pp. 66–73.

[12] J. S. Yi, N. Elmqvist, and S. Lee, "TimeMatrix: Analyzing temporal social networks using interactive matrix-based visualizations," *International Journal of Human-Computer Interaction*, vol. 26, no. 11-12, pp. 1031–1051, 2010.

[13] B. Bach, E. Pietriga, and J.-D. Fekete, "Visualizing dynamic networks with matrix cubes," in *Proceedings of the SICCHI Conference on Human Factors in Computing Systems*. ACM, 2014, pp. 877–886.

[14] B. Bach, N. Henry-Riche, T. Dwyer, T. Madhyastha, J.-D. Fekete, and T. Grabowski, "Small MultiPiles: Piling time to explore temporal patterns in dynamic networks," *Computer Graphics Forum*, vol. 34, no. 3, pp. 31–40, 2015.

[15] M. Burch and S. Diehl, "TimeRadarTrees: Visualizing dynamic compound digraphs," *Computer Graphics Forum*, vol. 27, no. 3, pp. 823–830, 2008.

[16] M. Burch, M. Höferlin, and D. Weiskopf, "Layered TimeRadarTrees," in *Proceedings of the 15th International Conference on Information Visualisation*. IEEE, 2011, pp. 18–25.

[17] C. Vehlow, M. Burch, H. Schmauder, and D. Weiskopf, "Radial layered matrix visualization of dynamic graphs," in *Proceedings of the 17th International Conference Information Visualisation*. IEEE, 2013, pp. 51–58.

[18] A. Perer and J. Sun, "MatrixFlow: Temporal network visual analytics to track symptom evolution during disease progression," in *AMIA Annual Symposium Proceedings*, vol. 2012. American Medical Informatics Association, 2012, pp. 716–725.

[19] C. Vehlow, F. Beck, P. Auwärter, and D. Weiskopf, "Visualizing the evolution of communities in dynamic graphs," *Computer Graphics Forum*, vol. 34, no. 1, pp. 277–288, 2015.

[20] S. Jürgensmann and H.-J. Schulz, "A visual survey of tree visualization," *IEEE VisWeek 2010 Posters*, 2010.

[21] H.-J. Schulz, "Treevis.net: A tree visualization reference," *IEEE Computer Graphics and Applications*, vol. 31, no. 6, pp. 11–15, 2011.

[22] M. Graham and J. Kennedy, "A survey of multiple tree visualisation," *Information Visualization*, vol. 9, no. 4, pp. 235–252, 2010.

[23] R. Lutz, D. Rausch, F. Beck, and S. Diehl, "Get your directories right: From hierarchy visualization to hierarchy manipulation," in *Proceedings of the 2014 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 2014, pp. 25–32.

[24] D. Holten and J. J. van Wijk, "Visual comparison of hierarchically organized data," *Computer Graphics Forum*, vol. 27, no. 3, pp. 759–766, 2008.

[25] T. Munzner, F. Guimbretière, S. Tasiran, L. Zhang, and Y. Zhou, "TreeJuxtaposer: Scalable tree comparison using focus+context with guaranteed visibility," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 453–462, 2003.

[26] F. Beck and S. Diehl, "Visual comparison of software architectures," *Information Visualization*, vol. 12, no. 2, pp. 178–199, 2013.

[27] M. Graham and J. Kennedy, "Exploring multiple trees through DAG representations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1294–1301, 2007.

[28] J. Guerra-Gómez, M. Pack, C. Plaisant, and B. Shneiderman, "Visualizing change over time using dynamic hierarchies: TreeVersity2 and the StemView," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2566–2575, 2013.

[29] J. A. Guerra Gómez, A. Buck-Coleman, C. Plaisant, and B. Shneiderman, "TreeVersity: Comparing tree structures by topology and node's attributes differences," in *Proceedings of IEEE Conference on Visual Analytics Science and Technology*. IEEE, 2011, pp. 275–276.

[30] M. Burch and S. Diehl, "Trees in a Treemap: visualizing multiple hierarchies," in *Visualization and Data Analysis*, vol. 6060. International Society for Optics and Photonics, 2006, pp. 60 600P–60 600P–12.

[31] T. Dwyer and F. Schreiber, "Optimal leaf ordering for two and a half dimensional phylogenetic tree visualisation," in *Proceedings of the 2004 Australasian Symposium on Information Visualisation*. Australian Computer Society, Inc., 2004, pp. 109–115.

[32] C. A. Stewart, D. Hart, D. K. Berry, G. J. Olsen, E. A. Wernert, and W. Fischer, "Parallel implementation and performance of fastDNAml: A program for maximum likelihood phylogenetic inference," in *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*. ACM, 2001, pp. 20–20.

[33] F. Chevenet, C. Brun, A. L. Banuls, B. Jacq, and R. Christen, "TreeDyn: towards dynamic graphics and annotations for analyses of trees," *BMC Bioinformatics*, vol. 7, no. 1, pp. 439–448, 2006.

[34] S. Gratzl, N. Gehlenborg, A. Lex, H. Pfister, and M. Streit, "Domino: Extracting, comparing, and manipulating subsets across multiple tabular datasets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2023–2032, 2014.

[35] S. Havre, B. Hetzler, and L. Nowell, "ThemeRiver: visualizing theme changes over time," in *Proceedings of the IEEE Symposium on Information Visualization*, 2000, pp. 115–123.

[36] P. Riehmann, M. Hanfler, and B. Froehlich, "Interactive Sankey diagrams," in *Proceeding of the IEEE Symposium on Information Visualization*. IEEE, 2005, pp. 233–240.

[37] C. Viau and M. J. McGuffin, "ConnectedCharts: Explicit visualization of relationships between data graphics," *Computer Graphics Forum*, vol. 31, no. 3, pp. 1285–1294, 2012.

[38] J. Zhao, Z. Liu, M. Dontcheva, A. Hertzmann, and A. Wilson, "MatrixWave: Visual comparison of event sequence data," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2015, pp. 259–268.

[39] B. Tversky, J. Bauer Morrison, and M. Bétrancourt, "Animation: can it facilitate?" *International Journal of Human-Computer Studies*, vol. 57, no. 4, pp. 247–262, 2002.

[40] M. Burch, C. Vehlow, F. Beck, S. Diehl, and D. Weiskopf, "Parallel edge splatting for scalable dynamic graph visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2344–2353, 2011.

[41] M. Greilich, M. Burch, and S. Diehl, "Visualizing the evolution of compound digraphs with TimeArcTrees," *Computer Graphics Forum*, vol. 28, no. 3, pp. 975–982, 2009.

[42] M. Hlawatsch, M. Burch, and D. Weiskopf, "Visual adjacency lists for dynamic graphs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 11, pp. 1590–1603, 2014.

[43] C. Mueller, B. Martin, and A. Lumsdaine, "A comparison of vertex ordering algorithms for large graph visualization," in *Proceedings of the 6th International Asia-Pacific Symposium on Visualization*. IEEE, 2007, pp. 141–148.

[44] S. Rufiange, M. J. McGuffin, and C. P. Fuhrman, "TreeMatrix: A hybrid visualization of compound graphs," *Computer Graphics Forum*, vol. 31, no. 1, pp. 89–101, 2012.

[45] J. B. Kruskal and J. M. Landwehr, "Icicle Plots: Better displays for hierarchical clustering," *The American Statistician*, vol. 37, no. 2, pp. 162–168, 1983.

[46] M. Shtern and V. Tzerpos, "Lossless comparison of nested software decompositions," in *Proceedings of the 14th Working Conference on Reverse Engineering*. IEEE, 2007, pp. 249–258.

[47] M. Shtern and V. Tzerpos, "A framework for the comparison of nested software decompositions," in *Proceedings of the 11th Working Conference on Reverse Engineering*. IEEE, 2004, pp. 284–292.

[48] F. Beck, F.-J. Wiszniewsky, M. Burch, S. Diehl, and D. Weiskopf, "Asymmetric visual hierarchy comparison with nested icicle plots," in *Joint Proceedings of the Fourth International Workshop on Euler Diagrams and the First In-ternational Workshop on Graph Visualization in Practice*, 2014, pp. 53–62.

[49] B. Alper, B. Bach, N. Henry Riche, T. Isenberg, and J.-D. Fekete, "Weighted graph comparison techniques for brain connectivity analysis," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2013, pp. 483–492.

[50] C. Minards, "Carte figurative des pertes succesives en hommes de l'Armée Francaise dans champagne de Russie," 1869.

[51] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical system structures," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 11, no. 2, pp. 109–125, 1981.

[52] M. Ghoniem, J.-D. Fekete, and P. Castagliola, "On the readability of graphs using node-link and matrix-based representations: A controlled experiment and statistical analysis," *Information Visualization*, vol. 4, no. 2, pp. 114–135, 2005.

[53] R. Keller, C. M. Eckert, and P. J. Clarkson, "Matrices or node-link diagrams: Which visual representation is better for visualising connectivity models?" *Information Visualization*, vol. 5, no. 1, pp. 62–76, 2006.

[54] F. Beck, M. Burch, and S. Diehl, "Towards an aesthetic dimensions framework for dynamic graph visualisations," in *Proceedings of the 13th International Conference Information Visualisation*. IEEE, 2009, pp. 592–597.

[55] M. Rosvall and C. T. Bergstrom, "Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems," *PLoS ONE*, vol. 6, no. 4, p. e18209, 2011.

Corinna Vehlow is PhD student at VISUS, University of Stuttgart, Germany. In 2010, she obtained the Diplom (M.Sc.) degree in computational visualistics from the Otto-von-Guericke University Magdeburg, Germany. Her research interests include information visualization and visual analytics with the focus on graph, set, and uncertainty visualization, often in the context of biological networks.

Fabian Beck is a postdoctoral researcher at VISUS, University of Stuttgart, Germany. In 2013, he obtained his Dr. rer. nat. (PhD) degree in computer science from University of Trier. In his dissertation, he investigated multi-dimensional coupling graph structures of software systems. Most of his research is focused on methods for visualizing and comparing large and dynamic graphs and hierarchies, often in the context of software systems and their evolution.

Daniel Weiskopf is a professor at VISUS, University of Stuttgart, Germany. He received his Dr. rer. nat. (PhD) degree in physics from the University of Tübingen, Germany (2001), and the Habilitation degree in computer science at the University of Stuttgart, Germany (2005). His research interests include all areas of visualization, visual analytics, GPU methods, perception-oriented computer graphics, and special and general relativity.