Radial Layered Matrix Visualization of Dynamic Graphs

Corinna Vehlow, Michael Burch, Hansjörg Schmauder, and Daniel Weiskopf
VISUS – Visualization Research Center, University of Stuttgart, Germany
{vehlowca,burchml,weiskopf}@visus.uni-stuttgart.de, schmaugh@studiu.informatik.uni-stuttgart.de

Abstract—We propose a novel radial layered matrix visualization for dynamic directed weighted graphs in which the vertices can also be hierarchically organized. Edges are represented as color-coded arcs within the radial diagram. Their positions are defined by polar coordinates instead of Cartesian coordinates as in traditional adjacency matrix representations: the angular position of an edge within an annulus is given by the angle bisector of the two related vertices; the radial position depends linearly on the angular distance between these vertices. The exploration of time-varying relational data is facilitated by aligning graph patterns radially. Furthermore, our approach incorporates several interaction techniques to explore dynamic patterns such as trends and counterevations. The usefulness is illustrated by two case studies analyzing large dynamic call graphs acquired from open source software projects.

I. INTRODUCTION

Relational data occurs in many application domains in real life, e.g., in software development and social networking, where the relations within the graph can be directed, weighted, and may be hierarchically organized. In various application scenarios, the relations even change over time, resulting in the need for dynamic graph visualization supporting fast and reliable exploration of such data.

Node-link diagrams are most prominent for visualizing graphs [1], [2]. They are suited for tasks related to path finding but have limitations for large and particularly for dense graphs due to inherent visual clutter [3] caused by many link crossings. Hence, for dense graphs, adjacency matrix representations outperform node-link diagrams [4] because they eliminate explicit link crossings.

Furthermore, using matrix representations, cluster structures and other patterns can become apparent as block structures, given that the vertices are ordered appropriately at the matrix axes. Hence, with these representations, symmetries and contact clusters can be easily detected, at least for vertices that are close to each other in the matrix layout. If large and dense graphs become dynamic, this introduces further challenges for the visualization to support the analysis of dynamic patterns, in node-link diagrams as well as matrix representations.

In general, dynamic data can be visualized in either static or animated diagrams. Animation is oftentimes used due to its intuitive and natural time-to-time mapping but can lead to high cognitive efforts due to our limited short term memory [5], [6]. Therefore, it is hard to explore dynamic graph data for trends and counterevations with animation.

The visualization of dynamic graphs using a static diagram, such as small multiples of node-link diagrams or adjacency matrices, has several benefits compared to graph animation. However, the small multiples representation also suffers from high cognitive load associated with the visual comparison of subnetworks or groups of nodes in the different diagrams. Node communities have to be located in each of the matrices separately and then these have to be compared one-by-one over long distances on the display. This can lead to misinterpretations caused by change blindness [7]. It can be difficult to compare clusters, since it is generally hard to decide whether two clusters at different points in time are identical or if they show a shifting, shrinking, or enlarging behavior.

II. VISUAL DESIGN

To facilitate the comparison of clusters over time, our novel visualization uses layered radial matrices instead of small multiples of the Cartesian adjacency matrix. Our approach is based on a time-to-space mapping, where the time axis corresponds to the radius of our radial layered matrix representation. The graphs are visualized as annuli that are stacked consecutively, where the innermost annulus represents the first time step. Similar to the Cartesian adjacency matrix, clusters become only apparent, if the vertices are ordered adequately. Using a hierarchical structure is a first step towards ordering the vertices and to reveal clusters.

A key advantage of our layout is that clusters of relations for consecutive points in time become much closer in the radial layered visualization (see Figure 2) than in the small multiples of Cartesian matrices (Figure 1), i.e., they become aligned to radial axis starting in the circle center and leading radially to the circle circumference. Hence, our approach rather addresses the investigation of temporal changes than the investigation of the graph structure at a particular point in time, see Figures 1 and 2 for a dynamic graph consisting of eight timesteps. The graph contains three large clusters, where cluster III involves the same nodes over all points in time. In contrast, cluster I and cluster II are shifted vertically and/or horizontally (in the Cartesian matrix, see Figure 1) and increase in size at several points in time. Cluster I is shifted vertically concerning its start nodes (in the Cartesian matrix) from time $t = 2$ to $t = 3$ and $t = 5$ to $t = 6$ and changes its size from time $t = 3$ to $t = 4$. Cluster II is shifted vertically (in the Cartesian matrix) from time $t = 1$ to $t = 2$, from time $t = 4$ to $t = 5$ and $t = 6$.
Figure 1. Dynamic directed graph shown in a small multiples representation by adjacency matrices. The graph consists of eight timesteps and contains three large clusters marked in the first graph \( t = 0 \) by Roman numerals.

Figure 2. Radial layered matrix representation showing the same dynamic directed graph as in Figure 1.

If we pick out a single cluster and try to inspect its behavior over time in the small multiples representation, we have to retrieve it in each of the matrices separately. Recognizing and relating the clusters at different points in time becomes much more complicated and hence time-consuming, if also the edge weights and hence colors change over time, due to change blindness [7]. What is even more difficult than the retrieval of clusters at different points in time, is the identification of changes of a cluster over time. Using small multiples, a vertical shift (change of start nodes) is recognizable at least to some extent due to the horizontal alignment of matrices (see clusters I and II in Figure 1). In contrast, a horizontal shift (change of end nodes) is not clearly visible within a sequence of horizontally aligned matrices. Such a comparison of clusters would become even harder using a matrix of Cartesian matrices instead of a linear sequence of Cartesian matrices.

In contrast to cluster I, cluster II and cluster III both involve nodes of the beginning and the end of the linearly arranged nodes and, hence, appear as 4 (2) separate clusters within the corners (at the borders) of the Cartesian matrix. As cluster II is directly adjacent to the boundaries of the matrix, the vertical and horizontal shifts can be detected more easily than for cluster I. In contrast, for cluster II it is even harder to perceive the growth in size compared to cluster I because of the displacement of cluster II.

The exploration of such changes over time is much easier using our radial visualization (see Figure 2) because a time-varying cluster is aligned along a single line starting in the circle center and leading radially outward to the circle circumference. Hence, we can easily detect the change of start and end nodes due to the radial shift of the complete cluster. If the boundaries of the cluster do not lie along one radial line, this indicates the occurrence of a shift or change of size. The former occurs if at the same time the cluster center (thickest part of the cluster within the annuli) is shifted in angular direction.

The boundaries of cluster I are both shifted clockwise, which indicates that both vertex groups, start and end vertices, change toward higher vertex numbers concerning the vertex ordering. Another advantage of our radial layered matrix is that the first and the last nodes, concerning the linear sorting of nodes, are brought together due to the radial mapping and hence cluster II and cluster III appear as one compact cluster instead of several separated blocks.

### III. Related Work

In many application scenarios, graphs do not remain static but evolve over time. The additional time dimension can be displayed by an intuitive time-to-time mapping using graph animation as demonstrated in online and off-line approaches by Diehl and Görg [8] and Frishman and Tal [9], respectively. Besides the already mentioned drawbacks, another problem with animation is the fact that algorithms are needed with high runtime complexities to produce sequences of graph layouts that have a high degree of dynamic stability. This concept allows viewers to better preserve their mental map [10], [11] when inspecting the graph sequence. Generally, it is difficult to apply interaction techniques to an animated sequence of graphs and to attach an additional hierarchical organization of the vertices. These drawbacks will also be addressed by our visualization approach of radial layered matrices.
Burch et al. [12] introduced a static representation of dynamic graphs based on a time-to-space mapping, named parallel edge splatting. Using this technique, vertices are mapped to fixed positions on parallel vertical lines and edges are drawn as straight links from left to right. Each graph of the sequence is mapped to a vertical stripe similar to the visual metaphor of parallel coordinates. Visual clutter is reduced by computing edge coverage and by showing this information as a color-coded density field. To improve the scalability of this approach regarding the time dimension, Beck et al. [13] used Rapid Serial Visual Presentation to rapidly browse time-varying graph data. If the time-varying graphs become dense, parallel edge splatting may also suffer from visual clutter. The radial edge splatting approach by Burch et al. [14] is similar to ours, showing time-varying graph cluster structures in a radial fashion; however, our mapping produces much less overdraw because it uses radially distorted pixels (tiny arcs) instead of explicit links as edge representatives.

Dynamic graphs can also be represented using small multiples, showing the graphs for each time step next to each other, e.g., as Cartesian matrix representations. Visual matrices, such as in Bertin’s “Semiology of graphics” [15], are an intuitive way to represent the logical adjacency matrix of a graph, in particular, for large and dense graphs. Unfortunately, the exploration of dynamic patterns, such as cluster trends, is difficult by means of sequences of Cartesian matrices because a viewer has to inspect one matrix after the other and identify the former pattern in each of the matrices again. If a pattern disappears for some time steps, even the contextual information has to be used in this search process. Consequently, such small multiples representations can lead to high cognitive loads depending on the dataset under examination.

There are also static representations based on the adjacency matrix that do not rely on the small multiples [16], [17]. The matrix representation by Brandes and Nick [16] makes use of small glyphs based on Gestalt principles, which they named dyad gestalts. These glyphs are built of a stack of horizontal lines that rotate around their center of mass, depending on the edge weight. Within the TimeMatrix [17], each cell contains a small bar chart showing the edge count for the respective pair of vertices over a period of time. Both representations allow users to investigate the evolution of single relations, whereas the investigation of evolving groups of relations is rather difficult using these visualizations.

To improve the comparability of subgraphs within the network, we use a radial layered density-based matrix diagram. Based on our approach, relations between subgroups of vertices at different time steps are arranged radially. Hence, they are placed close to each other and can thus be traced easily. Moreover, a radial representation is invariant under the shifting operation, i.e., the shifting of clusters as described in the previous section. Our approach supports a combined representation of the graph and the hierarchical organization of the vertices, which is represented by a sunburst representation [18], also known as InterRing [19].

IV. VISUALIZATION TECHNIQUE

We use a radial adjacency matrix representation because it has several benefits compared to its Cartesian counterpart. Due to the radial mapping, only one representative is needed for each vertex at the circle circumference. Moreover, vertices whose distance within the matrix is greater than \( n/2 \), where \( n \) is the number of vertices, become closer to each other due to the circular instead of linear alignment of vertices. This also becomes clear, if we look at the circular layout of the node-link diagram within that the vertices are also arranged radially (see Figure 3(a)). Hence, the maximal distance between two vertices is halved compared to the Cartesian matrix.

By using circle annuli for the time-varying matrices, as presented in this work, dynamic patterns can be easily analyzed because they are aligned along the radius of the annuli. Hence, a viewer just has to follow a line starting in the circle center and pointing radially outward. If a specific pattern disappears for example and reoccurs some time steps later, this phenomenon can easily be observed. In this section, we describe our novel visualization technique and show how the radial transformation of a matrix is generated.

A. Data Model

We model a directed weighted graph \( G = (V, E) \) as a set of vertices \( V \) and edges \( E \subseteq V \times V \). Each edge \( e \in E \) is associated with a weight \( w_e \in \mathbb{R} \), which is later mapped to color within the radial matrix representation. A dynamic graph \( \mathcal{G} := G_1, \ldots, G_m \) is modeled as a sequence of \( m \) subsequent graphs \( G_t, 1 \leq t \leq m \), where \( t \in \mathbb{N} \) and \( m \) is the number of points in time. The vertices of the dynamic graph may be hierarchically organized, where hierarchical organization should be based on the union of all vertex sets

\[
V_{\text{all}} := \bigcup_{1 \leq i \leq m} V_i.
\]

B. Radial Transformation

Based on the radial transformation, the upper-right and lower-left triangles of the adjacency matrix are mapped to the outer and inner rings of an annulus respectively, where the diagonal is mapped to the center line of that annulus (see Figure 3). Similar to the Cartesian matrix, we use a 1D mapping of vertices; however, vertices are no longer arranged along a straight axis but along a closed circular line similar to circular layouts of node-link diagrams (see Figure 3(a)). Hence, the first vertex \( v_1 \) and the last vertex \( v_n \) are placed next to each other.

More precisely, each vertex \( v_i \in V_{\text{all}}, 1 \leq i \leq n \), \( n = |V_{\text{all}}| \), is mapped to a position on a circle circumference by its corresponding angle \( \theta_i \in [0, 2\pi) \): \( \theta_i = (2\pi \cdot (i-1))/n \).
the relation between two vertices $v_k$ and $v_l$ is visually encoded via the position in Cartesian coordinates: $(x, y) := r(d(\theta_k, \theta_l)) \cdot (\cos(\bar{\vartheta}), \sin(\bar{\vartheta}))$, where $\bar{\vartheta}$ is the angle bisector (the angle obtained by computing the arithmetic means) of the corresponding angles $\theta_k$ and $\theta_l$.

For this arithmetic means computation, we choose the smaller intermediate circle sector between both vertices $v_k$ and $v_l$, i.e., the one that is smaller than, or equal to, $\pi$ (see Figure 3(b), within that cells are colored with different shades of green depending on the angle $\bar{\theta}$, starting with light green at $0\pi$ and ending with dark green at $\frac{\pi}{2}$). This corresponds to the course of straight edges in circular node-link diagrams, within that edges naturally intersect the smaller intermediate circle sector (see Figure 3(a)).

The function $d : [0, 2\pi) \times [0, 2\pi) \rightarrow [-1, +1]$ maps the relation between the vertices $v_k$ and $v_l$ to the normalized distance

$$d(\theta_k, \theta_l) = \min \left[ \frac{(\theta_k - \theta_l) \mod 2\pi}{\pi}, \frac{(\theta_l - \theta_k) \mod 2\pi}{\pi} \right] \sgn(\theta_l - \theta_k)$$

which is based on the angle of the smaller intermediate circle sector (see Figure 3(c), within that cells are colored with respect to the sign and value of $d$).

We use the convention that the modulo operator maps values to $[0, 2\pi)$ in this case. The sign function guarantees that the normalized distance is negative if $k > l$ and positive if $k < l$, i.e., they are mapped to positions within the inner (outer) annulus, respectively (see Figure 3(c) and 3(d)).

Therefore, the idea of $d$ is to represent the angle difference of the two vertices, taking into account the periodic structure of angles with periodicity $2\pi$. The highest angular distance with $|d()| = 1$ therefore occurs for vertices that are positioned exactly opposite on the circle (see edges $d(5 \rightarrow 2)$ and $e(3 \rightarrow 6)$ in Figure 3). Where several edges can be mapped to the same angular position $\bar{\vartheta}$, they can still be differentiated in the radial mapping due to their different radii $r$ (see, e.g., edges $d$ and $h$ in Figure 3).

Finally, the function $r : \mathbb{R} \rightarrow \mathbb{R}$ maps $d$ to the corresponding distance from the center of the annulus of width $b = R_2 - R_1$ for point $t$ in time:

$$r(d) = R_1 + b(t - 1) + b(d - 1)/2.$$

For the dynamic case, i.e., a time-varying graph, we add as many circle annuli of width $b$ to the representation as graphs have to be visualized, starting with the oldest one in the sequence in the circle center. This approach produces a radial transformation of a sequence of matrices that allows us to easily explore a time-varying weighted and directed graph for dynamic patterns and serves as a good overview representation.

To cover the domain without leaving holes even with large radius $r$, in fact, the edges are not plotted as pixels but as small circular arcs as if the pixels were distorted radially. The length of the arc is thereby proportional to the radius $r$. As mentioned before, these plotted elements are colored with respect to the edge weights.

**C. Attaching the Hierarchy**

Our radial graph representation is surrounded by a radial space-filling representation of the hierarchical structure of the vertices $v_i \in V_{all}$ (see Figure 4 and 5). Our sunburst-like representation adopts the radial space-filling tree visualization of Yang et al. [18]. Within the original approach, the root of the tree is represented as inner circle. We adapted this approach and embedded the annuli for all graphs $G_i$ within the inner circle of radius $r_{\text{max}} = R_1 + tb$. The first annulus surrounding this inner circle represents the root of the hierarchy.

**D. Interaction Techniques**

Our radial visualization for dynamic directed and weighted graphs follows the Visual Information Seeking Mantra [20]: it supports the user with an overview of the
large dataset in the vertex, edge, and time dimension, and with details-on-demand functions. To support the analysis process, brushing-and-linking is used as interaction technique. When hovering over a leaf node within the hierarchy and hence, vertex $v_i$, all edges starting $e(v_i, v_j)$ or ending $e(v_j, v_i)$ in this vertex will be highlighted and plotted as curves, connecting the endpoints $v_i$ and $v_j$ of the respective edge (based on the angular positions $\theta_i$ and $\theta_j$ and the outer radius of the annulus for the respective time step) and running through the position of the plotted relation. Furthermore, the leaf nodes for all vertices $v_j$ that are connected to $v_i$ are highlighted within the hierarchy. Using this interaction technique, the user can easily detect whether a vertex of interest is connected to vertices within another cluster or subgroup of the graph and how many connections the vertex has in general. Moreover, this interaction technique can be used to identify vertex insertions and deletions. If edges of a vertex $v_i$ only occur during some points in time $t > 1$ and/or $t < n$, this strongly indicates that the vertex itself was only present (part of the graph) during that interval.

The other way round, it is also possible to hover over the elements within the layered radial matrix representation, which represent edges $e(v_i, v_j)$ within the graph, to highlight the endpoints $v_i$ and $v_j$ of the selected edge within the hierarchy and to show the curve representation of the edge. As an alternative to the highlighting of leaves to improve the identification of endpoints, we implemented a circle sector-based highlighting that supports users to radially follow the evolution of vertices or rather the edges between them. Due to the radial transformation, each vertex $v_i \in V_{all}$ is allocated to a particular circle sector around $\theta_i$, where all sectors have the same angular size $\Delta \theta = \frac{2\pi}{n}$. Using this highlighting, the respective two sectors for the endpoints $v_i$ and $v_j$ of a selected edge will be highlighted.

V. CASE STUDIES
The usefulness of our visualization technique is illustrated by means of two case studies from the application domain of evolving call graphs in software development processes. Software systems are typically hierarchically structured and elements on the function or method level can call each other, building a large call graph. Since software is developed over time, these call graphs are evolving over time.

For both case studies, we first extracted the dynamic call graphs, i.e., we extracted the information about which software artifacts call each other on the method level of the respective program for each revision of the project. The hierarchical structure of the call graph, which is naturally defined by the structure of packages (classes and methods), is useful as a hierarchical organization of the vertices and can be represented as tree as described in Section IV-C. Since this data is not a compound graph, i.e., relations exist not only between the leaves but also between internal nodes of the tree, all nodes of the hierarchy represent vertices $v_i \in V_{all}$ of the graph. Hence, angular space $\Delta \theta$, i.e., circle sectors, is reserved for each node of the tree, internal nodes as well as leaf nodes.

From a software developer’s perspective, we are interested in the evolution of the call relations over time, i.e., over the revisions of the project. Our visualization provides software developers with an overview and serves as a starting point for further and more detailed exploration processes. Software developers that are familiar with the software project can even use the visualization to detect unexpected behavior. To do so, the layered sunburst-like representation of the hierarchy can be used to understand in which parts of the evolving project many call relations occur and how these evolve.

A. PMD Software Development

PMD is an open source software project that supports the inspection of source code written in the Java programming language. It is able to uncover inefficient structures such as local variables that are not used during execution, package imports that occur several times, empty try/catch blocks, and the like. Hence, programmers can use PMD to clean up their source code. The generated dynamic call graph consists of seven graphs that contain 1,102 vertices and a total number of 33,015 weighted directed edges, i.e., 8,130, 2,054, 3,608, 3,570, 4,542, 6,038, and 5,066 edges at single time intervals.

As illustrated in Figure 4, we can identify many structures, i.e., clusters of calls, that show various behaviors. By inspecting the orientation of the pixels in an annulus, we can recognize the involved software artifacts of the
weighted directed relations. There are several groups of strongly interconnected methods, shown as clusters that are tightly packed to the center lines of the annuli, that in most cases belong to the same package.

Within Figure 4, three such groups are highlighted (framed by radial lines of different color): the first one (I: brown) consists of the direct methods of the sourceforge/pmd package, the second cluster (II: blue) contains the direct methods of the sourceforge/pmd/swinggui package, and the third cluster (III: black) contains mainly direct methods of the sourceforge/pmd/rules package.

Cluster I persists for the first five revisions, whereas cluster II persists over the complete time interval but becomes weaker during the third and fourth revision. Cluster III is strong at the beginning, disappears for one time step, then reoccurs, and gradually becomes weaker.

Hovering over the leaves of the hierarchy shows that the methods of the subpackages jsp, java, ecmascript, dfu, cpp, and ast as well as several direct methods of the package sourceforge/pmd/lang are not related to any other method during the first five time steps. Moreover, this becomes clear as there are almost no edges placed within the lower sectors of the first five annuli (highlighted with the light red framed annulus segment in Figure 4). This implies that these methods (highlighted by the red arc outside of the radial matrix within Figure 4) were added only at the sixth revision. The method calls plotted in the middle of the red annulus segment in Figure 4 are placed far away from the center line. Hence, they represent method calls between methods of long radial distance and are not part of the above mentioned packages, which was confirmed when hovering over the edge representatives to highlight the respective methods. Figure 4 shows how the highlighting (light blue arcs and radial lines) looks like when hovering over edges within the annuli (here the position is indicated by the black arrow).

B. JUnit Software Development

JUnit is an open source software project that can be used for regression testing. The generated dynamic call graph comprises 21 graphs including 2,817 vertices connected by 15,339 edges in total. Similar to the first case study, we can identify many clusters of calls that show various behaviors: clusters (dis-)appear at certain revisions, become stronger or weaker (see Figure 5). The JUnit software includes mainly method calls between methods of the same package and only few calls between messages of different packages. This becomes clear, as most clusters appear as arcs that are lying on or at least very close to the center lines of the annuli. Only a few clusters appear at the outer borders of the annuli, e.g., the clusters highlighted by frame I in Figure 5.

The only packages for that method calls persist over all revisions are the extensions and the framework package (see highlighting II in Figure 5). Method calls of the packages awtui and runner are introduced at revision 3.4, where the majority of methods and method calls was introduced at revision 4.0. The latter becomes clear, as there are no edges placed within the second, third and fourth quadrant of the first eleven annuli (highlighted with the light red annulus segment in Figure 5).

A further drastic increase in methods and method calls appears at revision 4.4 (marked by the blue circle in Figure 5), as lots of clusters appear only from this revision on. Most clusters of method calls persist after they have been introduced, whereas some of the method calls between the packages framework, runner, and swingui disappear with revision 4.0 (see highlighting III in Figure 5).

CONCLUSIONS AND FUTURE WORK

In this paper, we have illustrated how a density-based radial matrix representation can be used to explore dynamic directed and weighted graphs. Our visualization approach aims at the investigation of temporal patterns instead of static patterns of a single graph. The evolution of clusters can be tracked easily due to spatial proximity, whereas different clusters within one graph might be positioned distant from each other if they lie opposite within the annulus, particularly for later graphs (outer annuli). The appearance of clusters as such thereby depends on the ordering the vertices, where sorting based on a natural hierarchy, such as the hierarchical package structure of the call graphs of our case study, is a good basis to reveal visual patterns. If no natural hierarchy is available, a hierarchical clustering algorithm could be employed to retrieve a meaningful structure and hence vertex ordering.
Due to the layering of annuli outward, for inner annuli and thus earlier graphs, less space is allocated than for annuli representing more recent graphs. Hence, our radial mapping approach puts a strong focus on later points in time of the graph. However, the layering could be easily inverted to shift the focus to earlier points in time.

Similar to the Cartesian matrix, the main drawback of our radial matrix approach is the fact that the visual grouping of clusters depends on the ordering of vertices. At least, clusters that are visually split within the matrix as they lie at the border of the matrix, are joined using our approach. Another drawback of our approach is the fact that edge representatives can be retraced to its vertices less easily than compared to its Cartesian counterpart, where one solely needs to follow the row and column. However, this drawback was addressed by interaction techniques that support the user in search processes and in getting used to this radial mapping.

Our visualization has several benefits compared to alternative visualization approaches:

- Similar to node-link diagrams, only one representative element is needed for each vertex of a graph. In contrast, in a Cartesian matrix, rows and columns are used, resulting in two representative elements for each vertex that cannot be easily aligned and compared visually.
- A hierarchical organization among the vertices can be displayed by one—not two as in a Cartesian matrix—radial sunburst diagram [18] or one radial indented pixel tree plot [21] together with the graph structure, supporting a data analysis at different levels of granularity.
- Large datasets, i.e., graphs with many vertices that are very dense, can be displayed by using a density-based heat map-like clutter-free representation.
- Distances between end points in a Cartesian matrix are reduced by connecting the ends in the radial diagram. Consequently, vertices can have a maximum distance of $\pi r$ on the circumference.
- A radial diagram is invariant under the vertex shifting operation (i.e., uniformly moving the vertex ordering). This means that visual patterns remain the same apart from rotation around the circle.

The major advantage of the static radial representation is that it supports the visual extraction of patterns, such as trends, counter-trends, or anomalies, within dynamic relational data. Due to the radial alignment and layering of the graphs, the evolution of clusters can be traced easily by following the radius outward. The usefulness of the novel radial technique is illustrated by two case studies from the domain of software evolution and social networking. Interaction techniques can be applied to further explore the data in more detail.

In future work, we plan to evaluate the visualization in a comparative eye tracking study and measure its performance against small multiples of the traditional Cartesian matrix representation. Furthermore, we want to add further interaction techniques combined with automatic data analysis, including automatic cluster detection.

REFERENCES


