Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data

Danny Holten

Abstract—A compound graph is a frequently encountered type of data set. Relations are given between items, and a hierarchy is defined on the items as well. We present a new method for visualizing such compound graphs. Our approach is based on visually bundling the adjacency edges, i.e., non-hierarchical edges, together. We realize this as follows. We assume that the hierarchy is shown via a standard tree visualization method. Next, we bend each adjacency edge, modeled as a B-spline curve, toward the polyline defined by the path via the inclusion edges from one node to another. This hierarchical bundling reduces visual clutter and also visualizes implicit adjacency edges between parent nodes that are the result of explicit adjacency edges between their respective child nodes. Furthermore, hierarchical edge bundling is a generic method which can be used in conjunction with existing tree visualization techniques. We illustrate our technique by providing example visualizations and discuss the results based on an informal evaluation provided by potential users of such visualizations.

Index Terms—Network visualization, edge bundling, edge aggregation, edge concentration, curves, graph visualization, tree visualization, node-link diagrams, hierarchies, treemaps.

1 Introduction

There is a large class of data sets that contain both hierarchical components, i.e., parent-child relations between data items, as well as non-hierarchical components representing additional relations between data items. Parent-child relations are henceforth called *inclusion* relations, whereas additional, non-hierarchical relations are henceforth called *adjacency* relations. Some examples of such data sets are:

- A hierarchically organized software system, e.g., source code divided into directories, files, and classes, and the relations between these elements, for instance, dependency relations;
- Social networks comprised of individuals at the lowest level of the hierarchy and groups of individuals at higher levels of the hierarchy. Relations could indicate if (groups of) people are acquainted and what the nature of their relationship is;
- A hierarchically organized citation network consisting of publications at the lowest level of the hierarchy and departments and institutes at higher levels of the hierarchy. Links between publications indicate one publication citing the other.

If we want to gain more insight in the hierarchical organization of each of the examples mentioned above, we can visualize the hierarchical structure using one of the many tree visualization methods that have been proposed in the past [2, 7, 15, 18, 24, 27]. However, if we want to visualize additional adjacency edges on top of this by adding edges in a straightforward way, this generally leads to visual clutter [3] (see figure 2a).

A possible way to alleviate this problem is to treat the tree and the adjacency graph as a single graph. Let the tree be represented by $T = (V, E_I)$ and the adjacency graph by $G(V, E_A)$. If the inclusion edges E_I and the adjacency edges E_A are merged into a single set of uniform edges, then the graph $G' = (V, E_I, E_A)$ can be visualized using a generic graph layout algorithm [2, 12, 15]. The problem that results from resorting to such a generic algorithm is that the inclusion and adjacency edges become intertwined, which can make it difficult to visually separate both types of edges from each other.

At present, only few techniques are available that are specifically designed to display adjacency relations on top of a tree structure, as is also mentioned by Neumann et al. [20]. Hence, the focus of this paper is on the construction of a generic technique for the visualization of compound graphs and compound directed graphs (digraphs) comprised of a tree and an additional (directed) adjacency graph.

We present hierarchical edge bundles, as described below, for the visualization of compound (di)graphs. Hierarchical edge bundling is based on the principle of visually bundling adjacency edges together analogous to the way electrical wires and network cables are merged into bundles along their joint paths and fanned out again at the end, in order to make an otherwise tangled web of wires and cables more manageable. The main features of the proposed technique are as follows:

- Hierarchical edge bundling is a flexible and generic method that can be used in conjunction with existing tree visualization techniques to enable users to choose the tree visualization that they prefer and to facilitate integration into existing tools;
- Hierarchical edge bundling reduces visual clutter when dealing with large numbers of adjacency edges;
- Hierarchical edge bundling provides an intuitive and continuous way to control the strength of bundling. Low bundling strength mainly provides low-level, node-to-node connectivity information, whereas high bundling strength provides high-level information as well by implicit visualization of adjacency edges between parent nodes that are the result of explicit adjacency edges between their respective child nodes.

The remaining part of this paper is organized as follows. In section 2 we give an overview of tree visualization techniques and existing techniques for visualizing additional adjacency edges. Section 3 describes the proposed hierarchical edge bundling technique in detail, followed by section 4, in which we present example visualizations and an informal evaluation. Finally, section 5 presents conclusions and possible directions for future work.

2 Related Work

Since hierarchical edge bundles can be used in conjunction with existing tree visualization techniques, we first give an overview of techniques that are commonly used for visualizing trees, followed by an

• *Danny Holten is with Technische Universiteit Eindhoven, E-mail: d.h.r.holten@tue.nl.*

Manuscript received 31 March 2006; accepted 1 August 2006; posted online 6 November 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

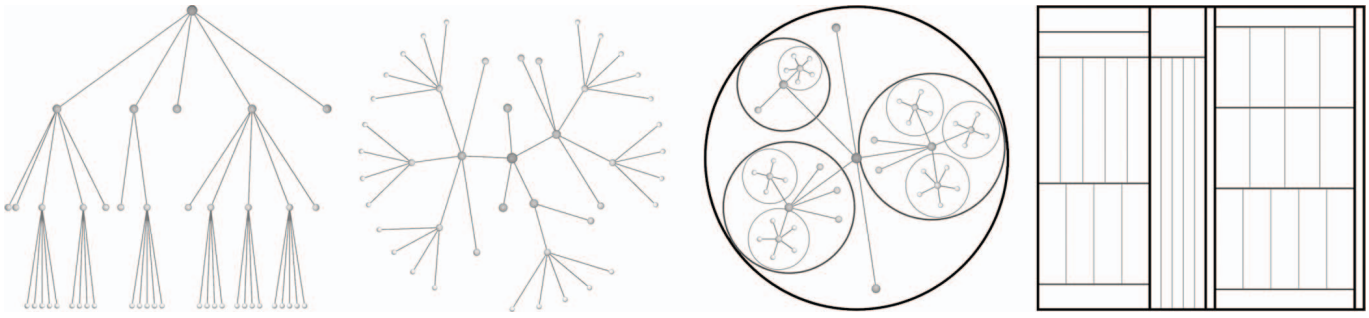

Fig. 1. Common tree visualization techniques. From left-to-right: rooted tree, radial tree, balloon tree, and treemap layout.

overview of methods that can be used to display adjacency edges on top of a tree visualization. Current methods for aggregating edges are mentioned as well.

2.1 Tree Visualization Techniques

One of the most well-known tree visualizations is the rooted tree depicted in figure 1 [2, 15, 24]. The rooted tree is an example of a tree visualization based on the intuitive node-link representation: the relationship between parent and child nodes is depicted by means of lines that connect the nodes. The top-down layout positions child nodes below their respective parent nodes and is the most common rooted tree layout; a variation on this is the left-to-right layout. The radial layout shown in figure 1 is another example of a node-link representation. In this layout, nodes are placed on concentric circles according to their depth in the tree [2, 7, 15]. The balloon layout shown in figure 1 is a node-link representation in which sibling subtrees are included in circles attached to the parent node [15].

Although radial and balloon layout techniques utilize the available space somewhat more efficiently than rooted layout techniques, node-link representations do not make optimal use of the available space in general. The treemap layout shown in figure 1 is a space-filling layout technique that displays a tree structure by means of enclosure, which makes it an ideal technique for displaying large trees [27].

However, using enclosure to display the tree structure makes it more difficult for viewers to perceive the hierarchical relationship between nodes. Furthermore, node-link representations can be made more space-efficient by using a focus+context (fisheye) technique; an example is the layout of a tree on a hyperbolic plane which is subsequently mapped onto a circular display region [18].

Visualization techniques that combine node-link representations and enclosure to offer a trade-off between an intuitive display and efficient space usage for displaying large trees exist as well. Examples are elastic hierarchies [32], SHriMP views [28], and space-optimized tree visualization [21].

Finally, 3D visualization techniques provide another way of achieving a more efficient use of space, although occlusion problems usually occur as a result of projecting 3D geometry onto a 2D screen. This makes interaction techniques – particularly options for changing the viewpoint – an essential part of these visualization techniques. Examples of 3D tree visualization techniques are cone trees [26] and H3, a 3D hyperbolic visualization [19].

2.2 Displaying Adjacency Relations

As mentioned earlier, simply adding adjacency edges to a tree visualization in a straightforward way quickly leads to visual clutter if a large number of edges is visualized (figure 2a). Fekete et al. [10] present a technique that displays the hierarchical structure as a treemap and the adjacency edges as curved links. The links are depicted as quadratic Bézier curves that show direction using curvature without requiring an explicit arrow. However, figure 2b shows that this technique suffers from visual clutter as well when many links are visualized. SHriMP views also use lines and curves added in a straightforward way for visualizing adjacency relations [28]. The 3D hyperbolic visualization H3 mentioned in section 1 also supports showing or hiding adjacency edges (added in a straightforward way) for a selected node or subtree [19].

Methods for drawing clustered graphs, which are graphs with recursive clustering structures over the vertices, are presented by Eades et al. [8, 11]. Furthermore, Kaufmann et al. provide a more general survey on drawing clusters (and hierarchies) [16]. In a sense, clustered graphs are similar to compound graphs in that they contain a hierarchical component as a result of their recursive clustering structure as well as non-hierarchical connections between the vertices of said clusters. However, the drawing of clustered graphs as presented by Eades et al. cannot be used as a method for drawing adjacency relations in conjunction with existing tree visualization techniques, since the layouts provided by their methods are fixed.

A similar remark holds for the use of force-directed algorithms for the layout of compound graphs, like the method presented by Dogrusoz et al. [6]. Issues with regard to computational complexity and layout stability have also been associated with force-directed methods [15]. However, these issues have recently been treated by various approaches [14, 17]. Most of these approaches also use a deterministic model to prevent the highly unpredictable layouts that were often produced by previous force-directed methods.

Another method for drawing compound graphs that originated in the graph drawing community is described by Sugiyama et al. [29]. In this method and its variations, nodes are drawn as rectangles, inclusion edges by the geometric inclusion among the rectangles, and adjacency edges by polylines connecting them [4, 23, 29]. Although this approach works fine for small compound digraphs as depicted in figure 2c, it does not scale very well for compound graphs containing a large hierarchy due to the inefficient usage of space.

A similar remark holds for ArcTrees as depicted in figure 2d, a hierarchical view derived from traditional treemaps that is augmented with arcs to depict adjacency relations [20]. This is a result of the fact that ArcTrees were primarily designed as an informative interactive tool for viewing documents and one of the requirements was to use as little screen space as possible because the majority of the space will be needed for the document itself.

Matrix views as depicted in figure 2e can be used as an alternative to node-link- and enclosure-based representations for showing adjacency relations between entities. The hierarchy is displayed along the axes of the matrix and adjacency relations are shown within the matrix as shaded cells [30, 33]. Matrix views present a stable and clean layout of the adjacency relations, but they are less intuitive than node-link- and enclosure-based representations [13, 30].

2.3 Edge Aggregation Techniques

Existing techniques that are related to edge aggregation are confluent graph drawing [5, 9] and flow map layouts [22].

Confluent graph drawing is a technique for visualizing non-planar graphs in a planar way by allowing groups of edges to be merged and drawn together [5, 9]. However, not every graph is confluently drawable and in general, it appears difficult to quickly determine whether or not a graph can be drawn confluently.

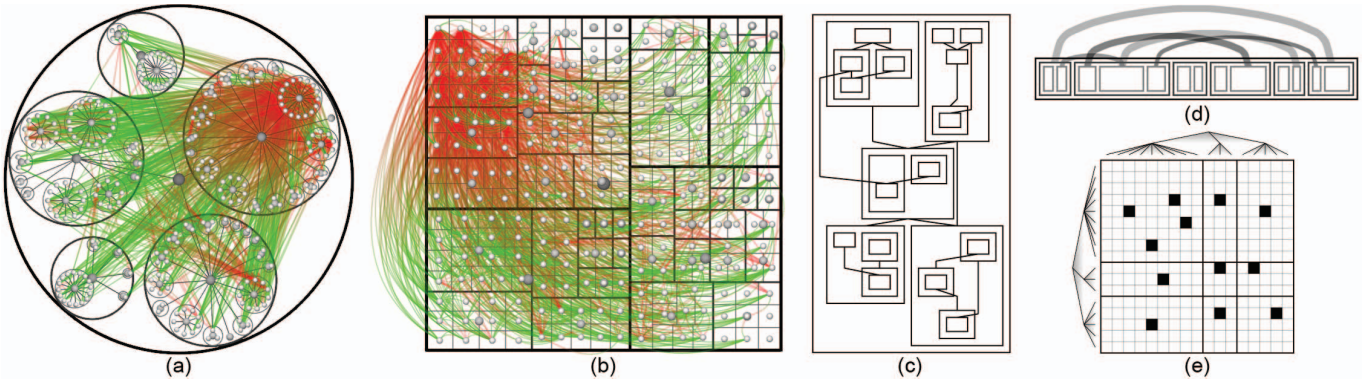

Fig. 2. Displaying adjacency relations using existing methods. A call graph visualized on top of the associated source code tree using (a) color-coded directed straight edges and (b) curved link edges (caller = green, callee = red); (c) standard compound digraph drawing; (d) ArcTrees for visualizing relations in hierarchical data; (e) a matrix view for showing relations between entities. (a) and (b) suffer from visual clutter, (c) and (d) furthermore suffer from the problem that they do not scale well for compound graphs containing a large hierarchy, and (e) is less intuitive than node-link- and enclosure-based representations.

Flow map layouts use hierarchical, binary clustering on a set of nodes, positions, and flow data to route edges [22]. As mentioned by Phan et al. [22], the biggest drawback is that all edge splits are binary; binary splits introduce too many extra routing nodes and lead to clutter if there are too many nodes in a small area.

3 Hierarchical Edge Bundles

This section provides a detailed description of our technique. Section 3.1 describes the basic idea behind hierarchical edge bundles, followed by section 3.2, in which the principles mentioned in section 3.1 are described in more detail. Additional design decisions for improving the layout are mentioned here as well. Finally, in section 3.3, details regarding the actual rendering of the bundles are discussed that further improve the final visualization.

3.1 Principle

Since we want our approach to be usable in conjunction with existing tree visualization techniques, we propose to use the layout provided by a tree visualization as a guide for bundling the adjacency edges. Figure 3 illustrates how this is done by using a balloon tree layout as an example. The approach is to use the path along the hierarchy between two nodes having an adjacency relation as the control polygon of a spline curve; the resulting curve is subsequently used to visualize the relation. The control points P_i that make up the control polygon are the points along the hierarchy from P_{Start} through $LCA(P_{Start}, P_{End})$ to P_{End}, where $LCA(P_{Start}, P_{End})$ is the *least common ancestor* of P_{Start} and P_{End} (see figure 3).

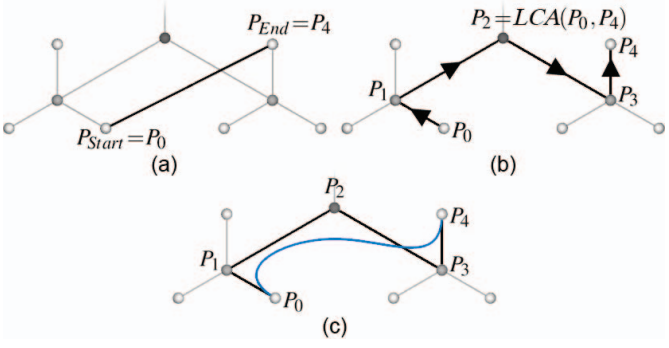

Fig. 3. Bundling adjacency edges by using the available hierarchy. (a) Straight line connection between P_0 and P_4; (b) path along the hierarchy between P_0 and P_4; (c) spline curve depicting the connection between P_0 and P_4 by using the path from (b) as the control polygon.

If this approach is used directly for bundling adjacency edges, ambiguity problems as depicted in figure 4a may arise. These problems can be reduced by diminishing the bundling strength. The bundling strength is controlled by a parameter β, $\beta \in [0,1]$, that effectively controls the amount of bundling by straightening the spline curve. Figure 4d shows the effect of this parameter and figure 4e illustrates how this can be used to resolve ambiguity problems.

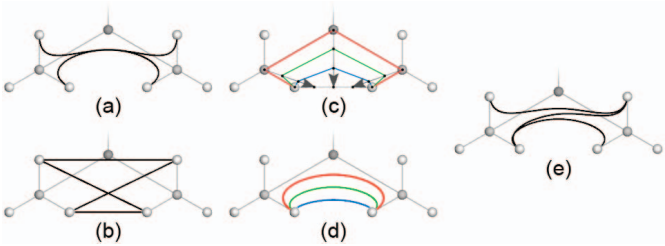

Fig. 4. Resolving bundling ambiguity. The bundle in (a) might contain each edge depicted in (b). (c) and (d) show how different values of β ($\text{red} = 1$, $\text{green} = \frac{2}{3}$, and $\text{blue} = \frac{1}{3}$) can be used to alter the shape of spline curves. As shown in (e), a fairly high bundling strength ($\beta = 0.8$) can be chosen to retain visual bundles while still resolving ambiguity.

3.2 Spline Models

Different spline models were investigated for visualizing the curves. We used weighted as well as non-weighted Bézier, B-spline, and Beta-spline curves [1] of different degrees and we also investigated hybrid approaches in which different spline curves were blended together using a weighted blending model.

Bézier curves lack the local control necessary to produce coherent and distinct bundles. Different combinations of the additional bias and tension parameters provided by Beta-splines did not result in better bundling behavior when compared to traditional B-splines. Using weighted instead of non-weighted B-splines did not readily improve the bundling either. We settled on a piecewise cubic B-spline representation, since this representation provided the amount of local control necessary for producing coherent and distinct bundles while keeping the degree – and with it the computational complexity – low. An open uniform knot vector of order 4 (degree = 3) which has degree -1 equal-valued knots at each end is employed to make the cubic B-spline interpolate its start and end points. The degree is automatically reduced to 2 or 1 if the number of control points is 3 or 2, respectively, since it is required that the degree is lower than the number of control points.

We investigated two ways of straightening a spline curve. The first method straightens the control polygon (as depicted in figure 4) by straightening each of its control points P_i and subsequently uses these straightened control points P_i' as the new control polygon to generate the spline curve:

$$P_i' = \beta \cdot P_i + (1-\beta)(P_0 + \frac{i}{N-1}(P_{N-1} - P_0)), \quad (1)$$

with
N : number of control points,
i : control point index, $i \in \{0,\ldots,N-1\}$,
β : bundling strength, $\beta \in [0,1]$.

The second method straightens each spline point $S(t)$ when the curve is evaluated to create a new, straightened spline point $S'(t)$:

$$S'(t) = \beta \cdot S(t) + (1-\beta)(P_0 + t(P_{N-1} - P_0)), \quad (2)$$

with
t : spline curve parameter, $t \in [0,1]$.

Due to the open uniform knot vector that is used to make the B-spline interpolate its start and end points, these methods yield somewhat different results. However, these differences are minimal from a visual point of view, as is illustrated in figure 5.

Let M denote the number of line segments that is used to draw the spline. In general, $M \geq 50$ is required to get a smooth spline representation. Hence, the first straightening method is preferable, since this only involves N straightening operations, with $N \leq 2D+1$ for hierarchies of depth D, whereas the second method requires $M+1$ straightening operations.

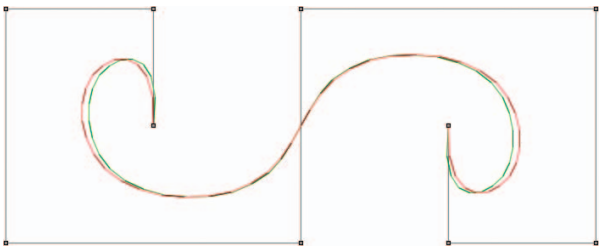

Fig. 5. Spline curve straightening by means of control polygon straightening (green) and spline point straightening (red) yield somewhat different results, but these differences are minimal from a visual point of view.

In general, it is not desirable to have connections between sibling nodes curve toward their common parent node, since the curvature does not provide additional high-level connectivity information; in this case, the curving would only unnecessarily clutter the display space around the common parent node. Removing the *LCA* from the original control polygon of a spline curve effectively prevents connections between sibling nodes from curving toward their common parent node, as is illustrated in figure 6.

However, figure 7a shows how removal of the *LCA* can lead to ambiguity problems if the original control polygons only contains three control points. *LCA* removal reduces a spline curve to a straight line in this case and the pictured ambiguity is the result of these straight line segments completely coinciding.

Figure 7b shows how this problem can be resolved while retaining the aforementioned benefit of *LCA* removal in the general case. This is accomplished by only allowing the *LCA* to be removed if the original control polygon is comprised of more than 3 control points.

3.3 Rendering

An important aspect of visualizing the curves is the order in which they are drawn. Since short curves only occupy a small amount of screen space, they tend to become obscured by long curves. This problem can be resolved by drawing short curves on top of long curves. In addition,

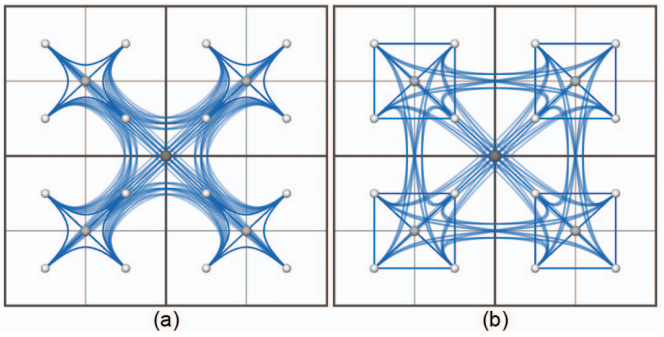

Fig. 6. Removing the *LCA* from the original control polygon. A tree of depth 2 and its additional adjacency graph ($\beta = 0.85$) visualized as a treemap with (a) *LCA* present, and (b) *LCA* removed. *LCA* removal effectively prevents connections between sibling nodes from curving toward their common parent node.

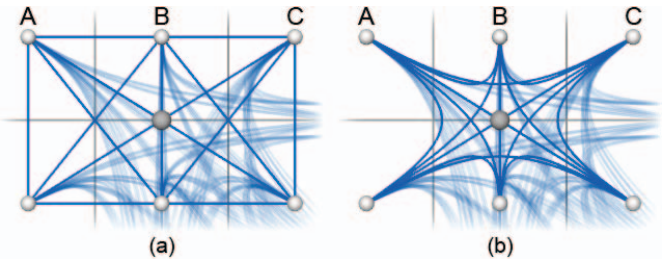

Fig. 7. Removal of the *LCA* can lead to ambiguity problems if the original control polygons only contains three control points. (a) It is unclear how sibling nodes A, B, and C are connected to each other due to straight line segments fully coinciding; (b) the problem can be resolved by requiring that the *LCA* may only be removed if the original control polygon is comprised of more than 3 control points.

we use alpha blending to further emphasize short curves by drawing long curves at a lower opacity than short curves; figure 8 shows an example of this.

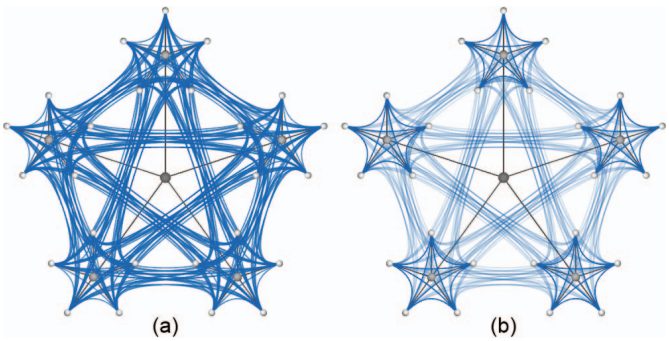

Fig. 8. Alpha blending can be used to emphasize short curves by drawing long curves at a lower opacity than short curves. (a) Alpha blending disabled; (b) alpha blending enabled.

As is illustrated in figure 10, alpha blending also helps to more easily discern individual curves or subbundles within a bundle.

Curved links [10] show how curvature can be used to indicate direction. We cannot readily use curvature to show direction, because we already use it for generating bundles. Furthermore, we do not use explicit arrows since doing so would clutter the visualization [10]. Instead, we show the direction of an edge by using an RGB interpolated color gradient to indicate the edge running from source (green) to destination (red), as is already illustrated by the non-bundled examples shown in figure 2.

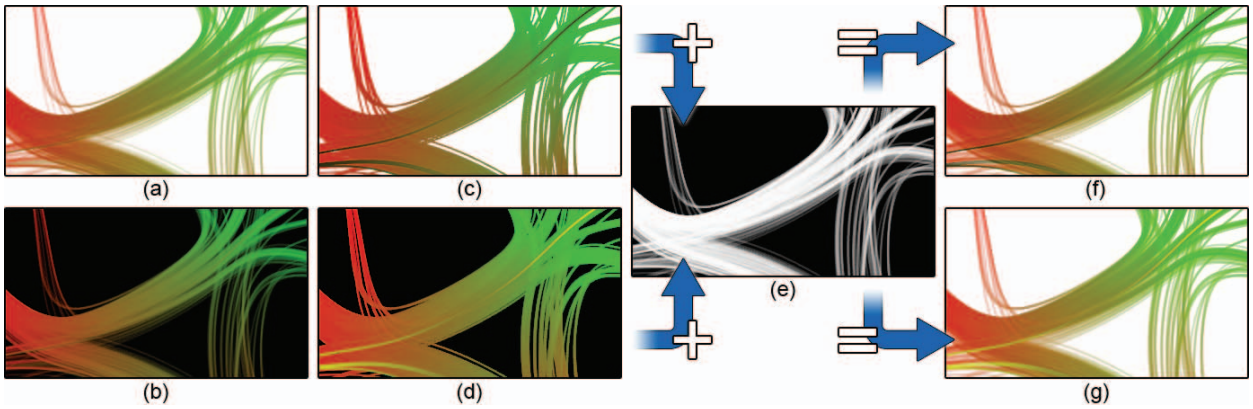

Fig. 9. Using OpenGL's `EXT_blend_minmax` extension. (a) and (b) show normal alpha blending on a white and black background, respectively; (c) and (d) show the usage of standard `MIN_EXT` (minimum) and `MAX_EXT` (maximum) blending, respectively. An individual curve within the bundle having an opposite direction can easily be discerned; (e) shows a transparency mask generated using normal alpha blending, which can be combined with the results depicted in (c) and (d); (f) and (g) show how this provides these results with additional levels of opacity (as is the case with standard alpha blending as depicted in (a) and (b)). The transparency mask furthermore allows them to be combined with other background colors than black and white. This combines the benefits of normal alpha blending and standard `MIN_EXT` and `MAX_EXT` blending.

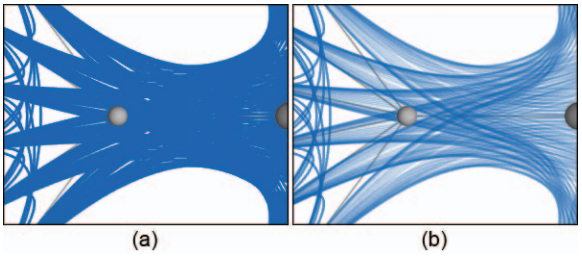

Fig. 10. Alpha blending helps to distinguish individual curves or sub-bundles within a bundle. (a) Alpha blending disabled; (b) alpha blending enabled.

An interesting variation on standard alpha blending is provided by the `EXT_blend_minmax` extension available in OpenGL 1.2 and higher. This extension can return the minimum or the maximum of two colors (source and destination) on a per-color-channel basis. In case of bundles that are mainly comprised of curves having a similar direction, these blending modes can be used to spot individual curves within the bundle having an opposite direction. Moreover, the blending modes are commutative, resulting in identical visualizations regardless of the order in which curves are drawn. Figure 9 shows an example of using the extension.

4 Results

The examples shown in this section are visualizations of a hierarchically organized software system and its associated call graph. The software is part of a medical scanner and was provided by Philips Medical Systems Eindhoven. Three hierarchy levels – layers, units, and modules – consisting of 284 nodes are used together with the associated call graph for the elements at the lowest level of the hierarchy, i.e., 1,011 adjacency relations representing module-to-module calls. Figures 2a and 11 show non-bundled visualizations using a balloon, radial, and squarified treemap layout; figures 13 and 15 show the bundled versions. Figure 12 depicts how the radial visualization shown in figure 13 was generated.

The non-bundled visualizations mainly show hot spots; the actual connectivity information is more difficult to discern due to visual clutter. Figures 13 and 15 show how bundling reduces visual clutter, making it easier to perceive the actual connections. The bundled visualizations also show relations between sparsely connected systems more clearly (visible within the encircled regions); these are almost completely obscured in the non-bundled versions.

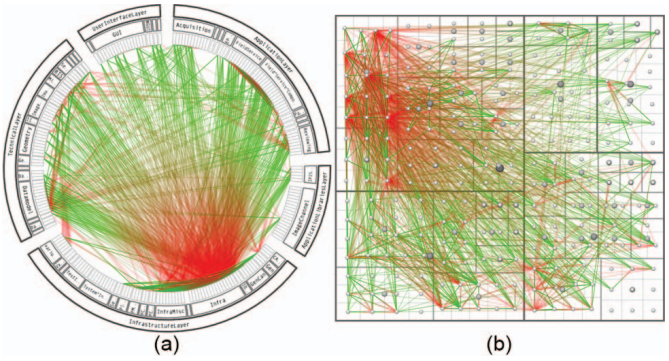

Fig. 11. A software system and its associated call graph (caller = green, callee = red). (a) and (b) show the system without bundling using a radial and a squarified treemap layout (node labels disabled), respectively. (a) and (b) mainly show hot spots; the actual connectivity information is more difficult to discern due to visual clutter.

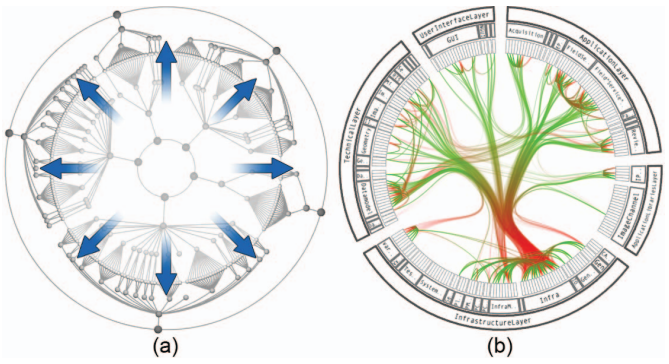

Fig. 12. Radial layout construction. (a) A radial tree layout is used for the inner circle and subsequently mirrored to the outside; (b) the inner layout is hidden and its structure is used to guide the adjacency edges. An icicle plot based on the mirrored layout is used to show the hierarchy.

4.1 User Feedback

We organized informal user studies to demonstrate our application and the resulting visualizations. Participants from academia and industry examined the Philips Medical Systems software by interactively

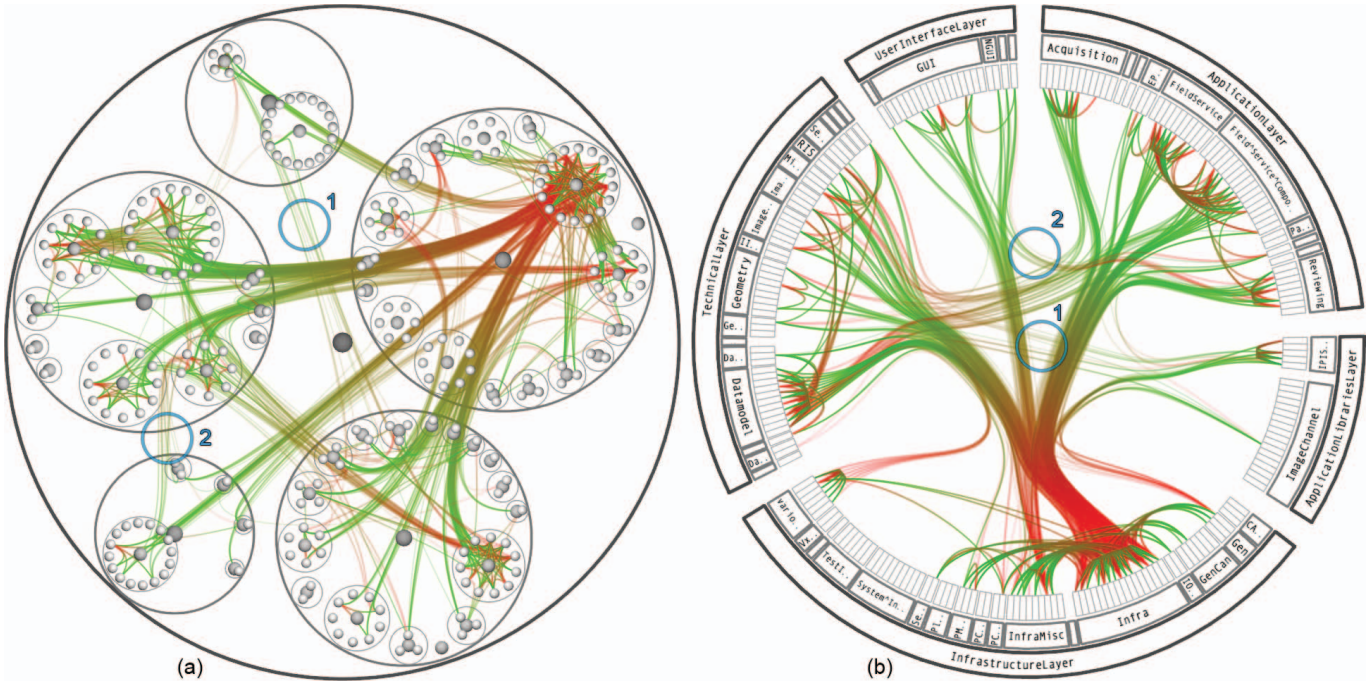

Fig. 13. A software system and its associated call graph (caller = green, callee = red). (a) and (b) show the system with bundling strength $\beta = 0.85$ using a balloon layout (node labels disabled) and a radial layout, respectively. Bundling reduces visual clutter, making it easier to perceive the actual connections than when compared to the non-bundled versions (figures 2a and 11a). Bundled visualizations also show relations between sparsely connected systems more clearly (encircled regions); these are almost completely obscured in the non-bundled versions. The encircled regions highlight identical parts of the system for (a), (b), and figure 15.

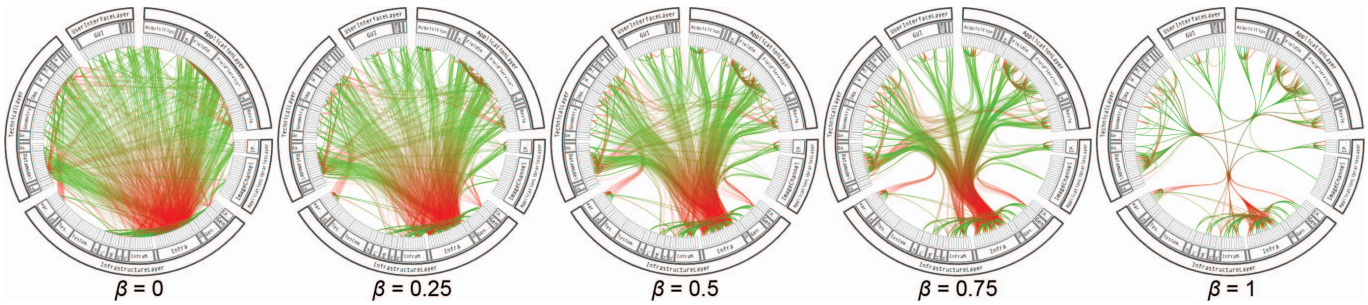

Fig. 14. Using the bundling strength β to provide a trade-off between low-level and high-level views of the adjacency relations. The value of β increases from left-to-right; low values mainly provide low-level, node-to-node connectivity information, whereas high values provide high-level information as well by implicit visualization of adjacency edges between parent nodes that are the result of explicit adjacency edges between their respective child nodes.

changing the bundling strength β and by switching between different tree layouts. The participants from academia were our fellow researchers, PhD students and MSc students from the Computer Science department of the Technische Universiteit Eindhoven. They all had experience with either software development, software visualization, or information visualization in general. Participants from industry were representatives of the Software Improvement Group (SIG) in Amsterdam, which delivers insight in the structure and technical quality of software portfolios, and representatives of FEI Company Eindhoven, which produces software to operate with FEI's range of electron microscopes.

The majority of the participants regarded the technique as useful for quickly gaining insight in the adjacency relations present in hierarchically organized systems. In general, the visualizations were also regarded as being aesthetically pleasing. SIG and FEI Company Eindhoven are currently supporting further development by providing us with additional data sets and feedback regarding the resulting visualizations.

More specifically, most of the participants particularly valued the fact that relations between items at low levels of the hierarchy were automatically lifted to implicit relations between items at higher levels by means of bundles. This quickly gave them an impression of the high-level connectivity information while still being able to inspect the low-level relations that were responsible for the bundles by interactively manipulating the bundling strength.

This is illustrated in figure 14, which shows visualizations using different values for the bundling strength β. Low values result in visualizations that mainly provide low-level, node-to-node connectivity information. High values result in visualizations that provide high-level information as well by implicit visualization of adjacency edges between parent nodes that are the result of explicit adjacency edges between their respective child nodes.

Another aspect that was commented on was how the bundles gave an impression of the hierarchical organization of the data as well, thereby strengthening the visualization of the hierarchy. More specifically, a thick bundle shows the presence of two elements at a fairly

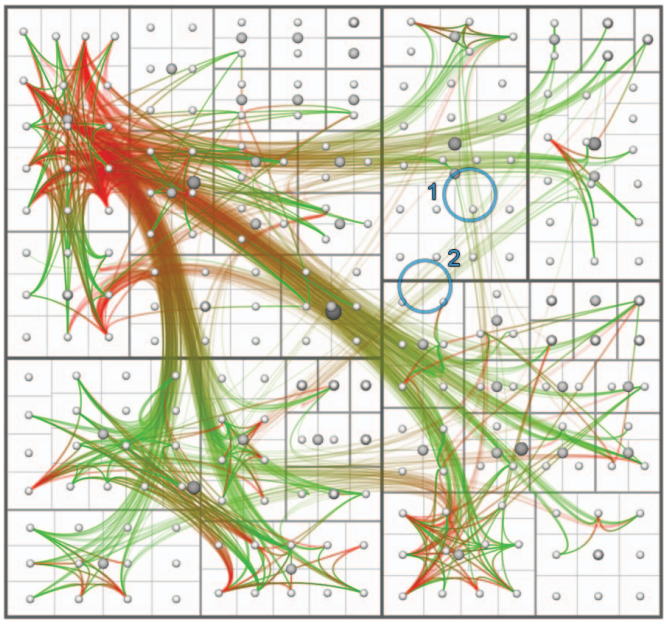

Fig. 15. The software system from figure 13 and its associated call graph (caller = green, callee = red) visualized using a squarified treemap layout (node labels disabled) for comparison with figure 11b. The encircled regions highlight the same parts of the system as in figure 13.

high level of the hierarchy, whereas the fanning out of a bundle shows the subdivision of an element into subelements.

Most participants preferred the radial layout over the balloon layout and the squarified treemap layout. Another finding was the fact that the rooted layout and the slice-and-dice treemap layout were considered less pleasing according to several participants. This is probably due to the large number of collinear nodes within these layouts, which causes bundles to overlap along the collinearity axes. This is illustrated in figure 17.

Although our main focus while developing hierarchical edge bundles was on the visualization itself, interaction is an important aspect in determining the usability of our technique. Based on our own insight and feedback gathered from participants, we contend that bundle-based interaction as described below could provide a convenient way of interacting with the visualizations.

Figure 16 shows how the bundling strength β could be used in conjunction with bundle-based interaction. The use-case scenario illustrated in figure 16 shows a user starting with a low-level view (figure 16a). Since the display is highly cluttered, the user increases the bundling strength, resulting in a bundled visualization in which areas of interest can be spotted as bundles. A bundle can subsequently be selected by dragging a line through it (figure 16b). As a result of this, only selected curves remain visible. Finally, the user decreases the bundling strength again to return to a more detailed, low-level view (figure 16c). In this view, the individual curves comprising the selected bundle can be further inspected without being hindered by the large amount of visual clutter that was initially present.

4.2 Performance

Our prototype application was implemented on Microsoft Windows XP Professional using Borland Delphi 7. OpenGL was used as the graphics API. The application provided real-time performance (10 frames per second or more) for the examples shown in this section on our development systems, a Dell OptiPlex GX280 PC with an Intel Pentium 4 3.0GHz CPU, an ATI Radeon X300 graphics card, and 1GB of RAM.

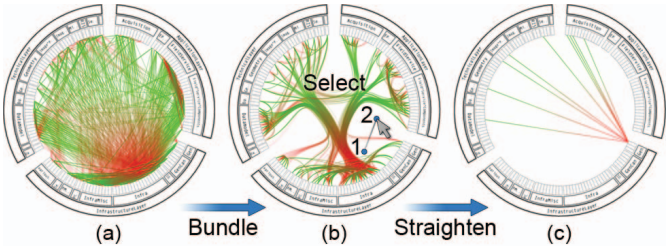

Fig. 16. Bundle-based interaction. (a) A user starts with a low-level view; (b) the bundling strength is increased and areas of interest emerge as bundles, which can be selected by dragging a line through them; (c) only selected curves remain visible and the user returns to an uncluttered low-level view to further inspect individual curves.

5 Conclusions and Future Work

We have introduced hierarchical edge bundling as a flexible and generic technique for the visualization of compound (di)graphs. We have demonstrated how hierarchical edge bundles can be used in conjunction with existing tree visualization techniques, how a bundle-based visualization is capable of reducing visual clutter, and how the bundling strength β can be used to provide a continuous trade-off between a low-level and a high-level view of the adjacency relations. We have also illustrated how the use of more advanced blending modes than standard alpha blending can provide a valuable addition to the rendering of hierarchical edge bundles. Based on our own experiences with our prototype application and feedback from participants of our informal user study, a preliminary, bundle-centric way of interacting with the visualizations has been presented as well.

The majority of our CS staff, students and local companies that participated in our informal user study considered the technique useful for quickly gaining insight in the adjacency relations present in hierarchically organized systems. Furthermore, they regarded the visualizations as being aesthetically pleasing.

As far as limitations are concerned, we currently consider the bundle overlap in case of layouts with a large number of collinear nodes to be the biggest problem of hierarchical edge bundles.

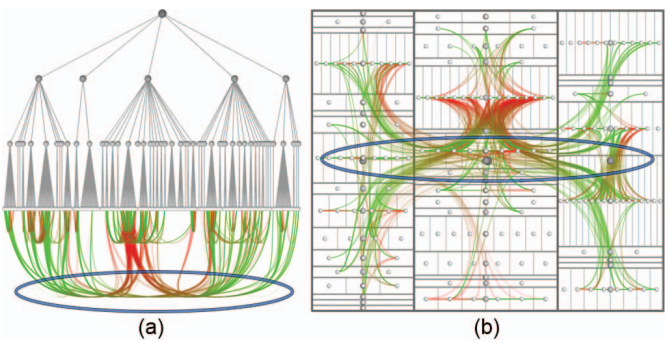

Fig. 17. Collinearity problems. The rooted tree layout (a) and the slice-and-dice treemap layout (b) were found to be less pleasing. This is probably due to the large number of collinear nodes within these layouts, which causes bundles to overlap along the collinearity axes, as is visible within the encircled regions.

The most important direction for future work would be the implementation of the suggested bundle-centric interaction technique as well as zooming techniques to allow users to inspect subparts of a data set in more detail. We are planning to make this extended version of our application available online as well to allow users to examine their own data sets by importing them into our application using Rigi Standard Format (RSF) [25]. Subsequently, user experiments, e.g., in collaboration with SIG and FEI Company, have to be performed to gain insight in the practical usability of our technique in conjunction

with the provided interaction techniques.

Another point of interest for future work would be to provide a way to locally decrease the bundling strength to allow a user to inspect individual curves within a bundle without having to modify the global bundling strength β. The EdgeLens approach presented by Wong et al. [31] provides one way to accomplish this. This approach manages edge congestion by interactively curving edges away from the point of focus, which opens up sufficient space to disambiguate relationships.

While we have focused on reducing visual clutter by bundling edges together, it would moreover be worthwhile to investigate how additional node reordering could be used to further reduce visual clutter in case of data in which node order is irrelevant.

Finally, feedback from participants of our informal user study indicated that in addition to visualizing and exploring hierarchies and adjacency relations, functionality for editing the data, i.e., by interactively reordering data elements to perform what-if investigations, is a future research direction that is worth considering as well.

Acknowledgements

I would like to thank my supervisor, Jarke J. van Wijk, for his guidance, encouragement, and advice during my work on Hierarchical Edge Bundles; without his help, this research would not have been possible. I would also like to thank Philips Medical Systems Eindhoven, The Software Improvement Group (SIG), and FEI Company Eindhoven for their input, discussions, and for providing input data sets. This project is founded by the Netherlands Organization for Scientific Research (NWO) Jacquard program under research grant no. 638.001.408 (*Reconstructor Project*).

References

[1] B. A. Barsky. *The Beta-Spline: A Local Representation based on Shape Parameters and Fundamental Geometric Measures*. PhD thesis, University of Utah, 1981.

[2] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[3] R. A. Becker, S. G. Eick, and A. R. Wilks. Visualizing Network Data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):16–28, 1995.

[4] F. Bertault and M. Miller. An Algorithm for Drawing Compound Graphs. In *Proceedings of the 7th International Symposium on Graph Drawing (GD'99)*, pages 197–204, 1999.

[5] M. T. Dickerson, D. Eppstein, M. T. Goodrich, and J. Y. Meng. Confluent Drawings: Visualizing Non-Planar Diagrams in a Planar Way. In *Proceedings of the 11th International Symposium on Graph Drawing (GD'03)*, pages 1–12, 2003.

[6] U. Dogrusoz, E. Giral, A. Cetintas, A. Civril, and E. Demir. A Compound Graph Layout Algorithm for Biological Pathways. In *Proceedings of the 12th International Symposium on Graph Drawing (GD'04)*, pages 442–447, 2004.

[7] P. Eades. Drawing Free Trees. *Bulletin of the Institute for Combinatorics and its Applications*, 5(2):10–36, 1992.

[8] P. Eades, Q.-W. Feng, and X. Lin. Straight-Line Drawing Algorithms for Hierarchical Graphs and Clustered Graphs. In *Proceedings of the 4th International Symposium on Graph Drawing (GD'96)*, pages 113–128, 1997.

[9] D. Eppstein. Delta-Confluent Drawings. In *Proceedings of the 13th International Symposium on Graph Drawing (GD'05)*, pages 165–176, 2006.

[10] J.-D. Fekete, D. Wang, N. Dang, A. Aris, and C. Plaisant. Overlaying Graph Links on Treemaps. In *Proceedings of the 2003 IEEE Symposium on Information Visualization (InfoVis'03), Poster Compendium*, pages 82–83, 2003.

[11] Q.-W. Feng. *Algorithms for Drawing Clustered Graphs*. PhD thesis, University of Newcastle, 1997.

[12] T. M. J. Fruchterman and E. M. Reingold. Graph Drawing by Force-Directed Placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[13] M. Ghoniem, J.-D. Fekete, and P. Castagliola. A Comparison of the Readability of Graphs Using Node-Link and Matrix-Based Representations. In *Proceedings of the 2004 IEEE Symposium on Information Visualization (InfoVis'04)*, pages 17–24, 2004.

[14] D. Harel and Y. Koren. A Fast Multi-Scale Method for Drawing Large Graphs. *Journal of Graph Algorithms and Applications*, 6(3):179–202, 2002.

[15] I. Herman, G. Melançon, and M. S. Marshall. Graph Visualization and Navigation in Information Visualization: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[16] M. Kaufmann and D. Wagner. *Drawing Graphs: Methods and Models*. Springer, 2001.

[17] Y. Koren, L. Carmel, and D. Harel. Drawing Huge Graphs by Algebraic Multigrid Optimization. *Multiscale Modeling and Simulation*, 1(4):645–673, 2003.

[18] J. Lamping and R. Rao. The Hyperbolic Browser: A Focus + Context Technique for Visualizing Large Hierarchies. *Journal of Visual Languages and Computing*, 7(1):33–55, 1996.

[19] T. Munzner. H3: Laying out Large Directed Graphs in 3D Hyperbolic Space. In *Proceedings of the 1997 IEEE Symposium on Information Visualization (InfoVis'97)*, pages 2–10, 1997.

[20] P. Neumann, S. Schlechtweg, and M. S. T. Carpendale. ArcTrees: Visualizing Relations in Hierarchical Data. In *Proceedings of the 2005 Eurographics / IEEE VGTC Symposium on Visualization (EuroVis'05)*, pages 53–60, 2005.

[21] Q. V. Nguyen and M. L. Huang. A Space-Optimized Tree Visualization. In *Proceedings of the 2002 IEEE Symposium on Information Visualization (InfoVis'02)*, pages 85–92, 2002.

[22] D. Phan, L. Xiao, R. Yeh, P. Hanrahan, and T. Winograd. Flow Map Layout. In *Proceedings of the 2005 IEEE Symposium on Information Visualization (InfoVis'05)*, pages 219–224, 2005.

[23] M. Raitner. Visual Navigation of Compound Graphs. In *Proceedings of the 12th International Symposium on Graph Drawing (GD'04)*, pages 403–413, 2004.

[24] E. M. Reingold and J. S. Tilford. Tidier Drawings of Trees. *IEEE Transactions on Software Engineering*, 7(2):223–228, 1981.

[25] Rigi: A Visual Tool for Understanding Legacy Systems. University of Victoria. `http://www.rigi.csc.uvic.ca/`.

[26] G. G. Robertson, J. D. Mackinlay, and S. K. Card. Cone Trees: Animated 3D Visualizations of Hierarchical Information. In *Proceedings of the 1991 SIGCHI Conference on Human Factors in Computing Systems (CHI'91)*, pages 189–194, 1991.

[27] B. Shneiderman. Tree Visualization with Tree-Maps: 2-d Space-Filling Approach. *ACM Transactions on Graphics (TOG)*, 11(1):92–99, 1992.

[28] M.-A. D. Storey and H. A. Müller. Manipulating and Documenting Software Structures using SHriMP Views. In *Proceedings of the 1995 International Conference on Software Maintenance (ICSM'95)*, pages 275–284, 1995.

[29] K. Sugiyama and K. Misue. Visualization of Structural Information: Automatic Drawing of Compound Digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):876–892, 1991.

[30] F. van Ham. Using Multilevel Call Matrices in Large Software Projects. In *Proceedings of the 2003 IEEE Symposium on Information Visualization (InfoVis'03)*, pages 227–232, 2003.

[31] N. Wong, S. Carpendale, and S. Greenberg. EdgeLens: An Interactive Method for Managing Edge Congestion in Graphs. In *Proceedings of the 2003 IEEE Symposium on Information Visualization (InfoVis'03)*, pages 51–58, 2003.

[32] S. Zhao, M. J. McGuffin, and M. H. Chignell. Elastic Hierarchies: Combining Treemaps and Node-Link Diagrams. In *Proceedings of the 2005 IEEE Symposium on Information Visualization (InfoVis'05)*, pages 57–64, 2005.

[33] J. Ziegler, C. Kunz, and V. Botsch. Matrix Browser: Visualizing and Exploring Large Networked Information Spaces. In *Extended Abstracts of the 2002 SIGCHI Conference on Human Factors in Computing Systems (CHI'02)*, pages 602–603, 2002.