TreeMatrix: A Hybrid Visualization of Compound Graphs

Sébastien Rufiange[1] and Michael J. McGuffin[1] and Christopher P. Fuhrman[1]

[1]Department of Software and IT Engineering, École de technologie supérieure, Montréal, Canada
sebastien@rufiange.com, {michael.mcguffin, christopher.fuhrman}@etsmtl.ca

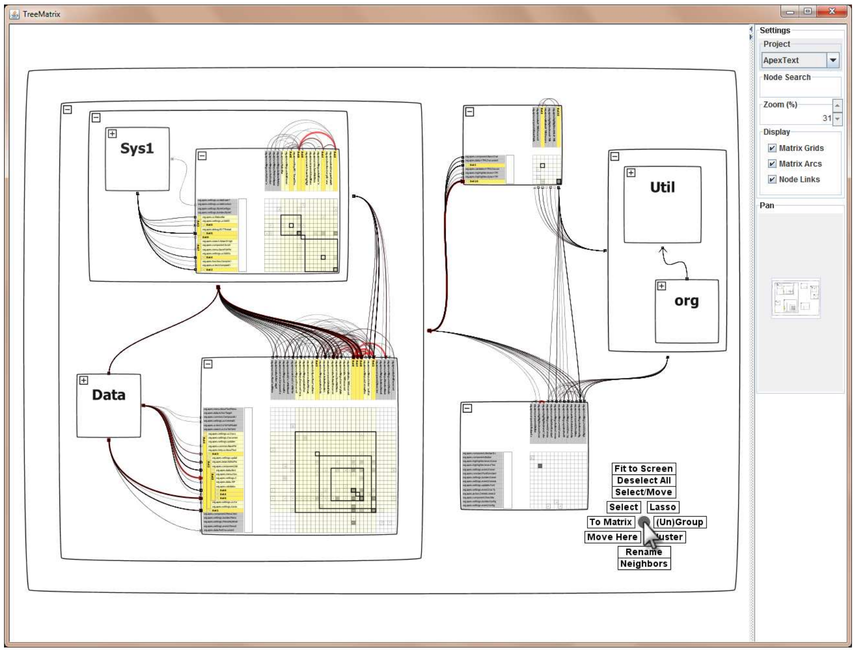

Figure 1: *The TreeMatrix prototype displaying the structure of a set of source code files as a compound graph. Portions of the compound graph are shown as adjacency matrices. Some matrices have had their contents collapsed by the user, and only their names are visible (e.g., "Sys1", "Util"). Rounded rectangles surrounding the matrices show the upper levels of the tree structure of the compound graph. Curves connecting the matrices show graph edges of the compound graph. At lower right, a radial menu is popped up to access commands.*

Abstract

We present a hybrid visualization technique for compound graphs (i.e., networks with a hierarchical clustering defined on the nodes) that combines the use of adjacency matrices, node-link and arc diagrams to show the graph, and also combines the use of nested inclusion and icicle diagrams to show the hierarchical clustering. The graph visualized with our technique may have edges that are weighted and/or directed. We first explore the design space of visualizations of compound graphs and present a taxonomy of hybrid visualization techniques. We then present our prototype, which allows clusters (i.e., subtrees) of nodes to be grouped into matrices or split apart using a radial menu. We also demonstrate how our prototype can be used in the software engineering domain, and compare it to the commercial matrix-based visualization tool Lattix using a qualitative user study.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing algorithms I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques H.5.2 [Information Interfaces and Presentation]: User Interfaces—Interaction styles

1. Introduction

A compound graph, also called a clustered graph [DBF09], is a graph (i.e., network) together with a rooted tree such that the leaves of the tree are the vertices of the graph. The non-leaf nodes of the tree can be thought of as meta-nodes or clusters of nodes. Compound graphs are useful for modelling situations where there is a network whose nodes can be grouped together into clusters. For example, a social network could be modelled as a compound graph, with graph edges representing relationships between people, and the tree clusters representing hierarchically-organized communities. As another example, in biology, a protein-protein interaction (PPI) network forms a graph whose nodes can be clustered hierarchically according to Gene Ontology [Gen00] terms. Many algorithms exist for automatically computing a hierarchical clustering of a graph, essentially converting the graph into a compound graph.

Clusters are typically very meaningful to users in a specific application domain; they help users understand the structure of the compound graph. These clusters, whether computed or provided with the input graph, can be used to aid visualization (e.g., [GKN05, BD07]) to limit the amount of detail the user sees at any given moment, e.g., by collapsing clusters of nodes and representing them as a single meta-node in the compound graph's tree. If the compound graph is large, such collapsing may be more convenient than trying to visualize all details of the graph at once. However, as pointed out in [BD07], a disadvantage of collapsing a cluster to a single meta-node is that detail is lost.

A second problem with many visualizations of graphs generally (not just compound graphs) is that of edge crossings obscuring information, especially in dense networks. To address this second problem, visualizations based on adjacency matrices (e.g., [Ber83, SJSJ05, GFC05]) have been proposed, which completely eliminate edge crossings. It is also possible to mix adjacency matrices with traditional node-link representations, as done with NodeTrix [HFM07]. The current work presents a novel visualization for compound graphs that extends NodeTrix by combining multiple previous visualization techniques for trees and graphs. We call our visualization TreeMatrix (Figures 1, 2, 9), because it makes use of adjacency matrices for visualizing parts of a graph, and simultaneously displays the tree structure in a compound graph. The use of matrices addresses the problem of edge crossings mentioned above. It also helps mitigate the first problem mentioned above, that collapsing a cluster to a single meta-node can remove too much detail: instead of completely collapsing a cluster of nodes, the user may instead convert it to a matrix, to provide a visual summary of the nodes within it. As discussed later, our technique also combines aspects of MatLink [HF07] and Lattix [SJSJ05].

Before presenting our visualization in detail, we first investigate the design spaces of hybrid visualizations of trees, hybrid visualizations of graphs, and visualizations of compound graphs. The taxonomies we present of these design spaces build upon the previously presented taxonomy in [ZMC05] and clarify the reasons behind our design choices. We then present our prototype implementation and demonstrate its use for visualizing, interpreting, and reverse engineering the design of software source code. Our contributions are (1) new taxonomies of hybrid visualizations of trees, graphs, and compound graphs; (2) a new visualization technique for compound graphs that extends and combines aspects of NodeTrix [HFM07], MatLink [HF07], and Lattix [SJSJ05]; (3) a description of a software prototype implementing this technique, and (4) the results of a user study where our prototype was compared with Lattix, a status quo commercial product, for software design tasks.

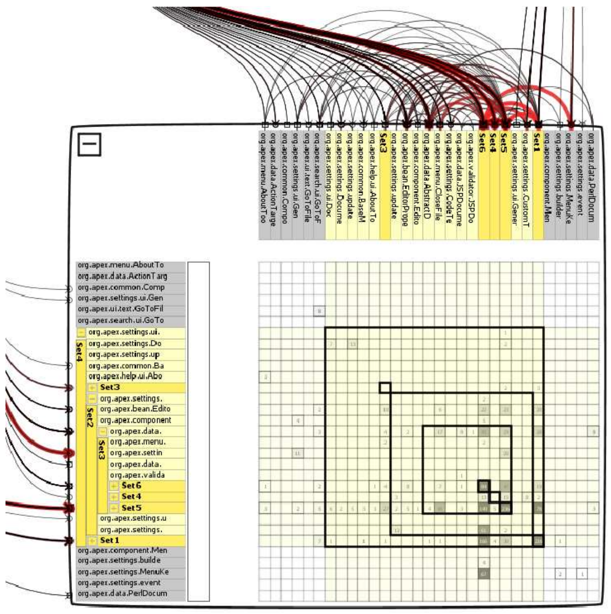

Figure 2: *A zoomed view of the adjacency matrix in the lower left of Figure 1. Within each matrix of the visualization, the tree structure of the compound graph is shown with nested black squares inside the matrix, as well as with an "icicle diagram" listing nodes along the left edge of the matrix. Also, within each matrix, the weighted edges of the graph structure are shown by the numbers and colors in the cells of the matrix, as well as by 180-degree arcs along the top of the matrix.*

2. Background

2.1. Visualization of Trees

Many techniques exist for visualizing trees [JS10], including node-link diagrams, icicle diagrams, and nested enclosure approaches such as treemaps (Figure 3). As the depth of a tree increases, the number of nodes often increases exponentially, leading to crowding at the deeper levels.

Treemaps [JS91, BHvW00, Wat05] are a technique for drawing trees that make more efficient use of area than traditional node-link approaches, allowing more area to be allocated to each node. Treemaps are part of a more general category of tree representations which use *nested enclosure* to depict the tree structure. In an analysis of the space-efficiency of representations of trees [MR10], nested enclosure approaches (including treemaps) were ranked as asymptotically more efficient than node-link approaches. However, representations based on nested enclosure can also be more confusing than node-link diagrams. For example, judging the depth of a node within a treemap is more confusing than in a classical layered node-link diagram. Hence, elastic hierarchies [ZMC05] were proposed, allowing users to visualize parts of a tree with treemaps, and other parts in node-link form.

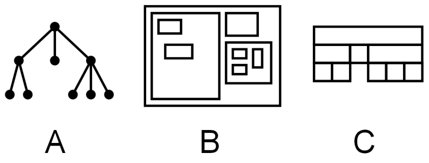

Figure 3: *Three ways of drawing the same tree. A: node-link diagram. B: nested enclosure. C: icicle diagram.*

2.2. Visualization of Graphs

Many layout algorithms have also been proposed for visualizing graphs [DBETT99, HMM00, KW01], most of these based on node-link diagrams. Node-link diagrams, however, suffer from edge congestion when depicting graphs with many edges. An alternative to using node-link diagrams to depict graphs is to use adjacency matrices. Matrices have the advantage of completely eliminating edge crossings, but have the disadvantage of making path-following tasks more difficult [GFC05]. For this reason, enhancements have been proposed to matrices [SM07], including MatLink [HF07] which adds arcs along the edge of the matrix. Arcs allow node adjacencies to be shown along a 1-dimensional layout, as previously shown in [Ber83] and applied to the visualization of repeating substrings in [Wat02].

Matrices also have a significant area cost ($\Theta(N^2)$), and so probably don't offer much advantage, if any, when the graph to be depicted is sparse. For this reason, NodeTrix [HFM07] was proposed, with the idea of depicting the dense portions of a graph with matrices, and the remaining portions in node-link form. Notice the analogy between elastic hierarchies [ZMC05] and NodeTrix [HFM07]: both allow two representations to be mixed in the depiction of the data, according to the density of data and the user's desires.

2.3. Visualization of Compound Graphs

There exists much previous work on the computation of hierarchical clusters of nodes within a graph (e.g., [ACJM03, GKN05, AvHK06, AMA09]). The TreeMatrix technique that we present is concerned primarily with *how to depict* the resulting compound graph, and is independent of the method used to determine the clusters.

There is also previous literature on computing the layout of node-link diagrams of compound graphs (e.g., [SM91, San96, BM99]). More recent visualizations of compound graphs have proposed new variants of node-link diagrams [Hol06, PvW06, BD07, GBD09] (see von Landesberger et al. [vLKS*10] for a survey of techniques for visualizing compound graphs, and Elmqvist and Fekete [EF10] for a related survey of techniques for aggregation).

Lattix [SJSJ05] is the earliest matrix-based technique we know of for visualizing compound graphs (Figure 4). Edges of the graph are weighted, and these weights are indicated in the cells of the matrix. The tree structure is shown using an icicle diagram along the left of the matrix; this icicle diagram can be used to expand or collapse portions of the compound graph. The tree structure is also shown, redundantly, with nested squares within the matrix.

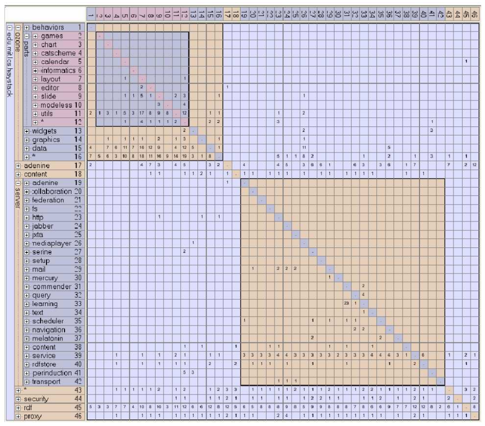

Figure 4: *A screenshot of Lattix [SJSJ05].*

Note that all of these previous approaches use only one representation for the graph structure of the compound graph. None of them are hybrid, in the sense of using a mixture of representations for either the tree structure, or for the graph structure. (Lattix does use two duplicated representations for its tree structure: an icicle diagram and nested squares, but these are not mixed in a way that allows a user to tradeoff between them, as a hybrid like elastic hierarchies does.) Thus, these previous techniques will suffer from at least one of the disadvantages mentioned in sections 2.1-2.2. For example, most of the previously published techniques use node-link diagrams to show the graph structure of the compound graph, and thus will suffer from edge congestion if there are many edges. Lattix uses a single, large matrix for the entire compound graph, rather than a hybrid like MatLink or NodeTrix, and thus suffers from the disadvantages of matrices that paths are difficult to follow and the area required is $\Theta(N^2)$.

To summarize, our work is concerned not with clustering algorithms or layout algorithms, but hybrid depictions of the compound graph. A tantalizing possibility is to combine the approaches of elastic hierarchies and NodeTrix to show both the tree and graph structure of a compound graph in hybrid forms, allowing the user to use the most appropriate representations for each part of the data. As we will show, however, our final solution is not simply "elastic hierarchies + NodeTrix", as there are several subtle design issues to consider.

In the next section, we illustrate these issues with taxonomies of visualizations of trees and graphs. Contrary to the prior survey of techniques in von Landesberger et al. [vLKS*10], we will not consider techniques for time-varying structures in our taxonomy, but instead focus on constructing taxonomies of hybrid visualizations.

3. Taxonomy of Hybrid Visualizations of Trees and Graphs

Consider a dataset D and a function $A(D)$ that maps D to a visual representation. Consider further that some alternative visual representation may be used, given by function $B(D)$. To create a hybrid mixture of the two representations, we take some subset $S \subset D$ and visualize it as $A(S)$, and visualize the remaining data $D \setminus S$ as $B(D \setminus S)$. Finally, some means is used to combine the two visualizations to yield a single hybrid $A(S) \oplus B(D \setminus S)$. The exploration of different hybrid visualizations for a given data type can thus be seen as choosing different functions for A and B, choosing an appropriate subset S that we may wish to represent differently from the rest of the data, and choosing one or more operators $\oplus$ for combining representations.

If D is a tree, then each of the functions A and B could be a node-link representation or a treemap representation (other representations of trees are possible, but for our purposes these two will provide a useful contrast). Many subsets of S of the tree could be considered, however one subset that occurs naturally when dealing with tree data is a *subtree* of some node n. So, let S be the subtree $S(n)$ under node $n \in D$. We will refer to the remaining nodes $D \setminus S$ as the "surrounding nodes". Finally, in the expression $A(S(n)) \oplus B(D \setminus S(n))$, the operator $\oplus$ could either insert $A(S(n))$ under n's parent (call this $\oplus_{insert}$), or could display $A(S(n))$ some distance away from $B(D \setminus S(n))$ with a connective line (call this $\oplus_{connect}$). Choosing a hybrid is then equivalent to choosing between node-link and treemap for each of A and B, crossed with $\{\oplus_{insert}, \oplus_{connect}\}$, yielding 8 possible hybrid visualizations. Of these, two are eliminated as uninteresting because there is little difference between $\oplus_{insert}$ and $\oplus_{connect}$ when B is a node-link representation. The remaining 6 visualizations are shown in Figure 5.

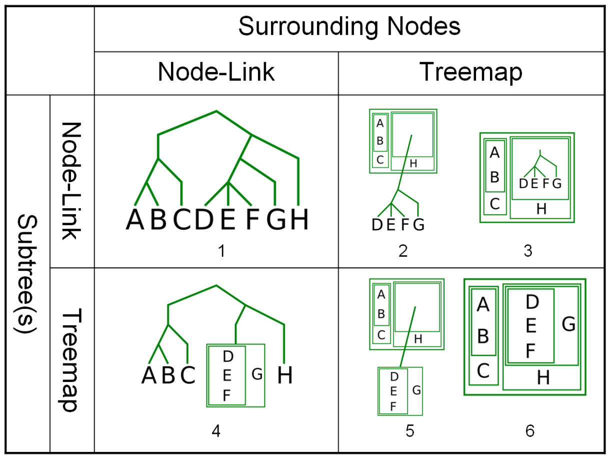

Figure 5: *Hybrid visualizations of trees. 1 shows a classical layered node-link diagram, and 6 shows a "pure" treemap. 2, 3, 4, and 5 show various instances of hybrid Elastic Hierarchies [ZMC05].*

An analogous taxonomy can be generated for hybrid visualizations of a graph D. In this case, the subset $S \subset D$ may be any subgraph, but would typically be a cluster of nodes that the user wishes to represent differently from the rest of the graph. We will refer to this subset S as a "local subgraph", since it may be thought of as a meaningful set of nodes that are somehow close to each other, perhaps close to a focal node. Next, the functions A and B can each be node-link or matrix representations, and the operators $\oplus_{insert}$ and $\oplus_{connect}$ are analogous to the previously defined ones. Again, 8 possible hybrids emerge, and again 2 can be eliminated because there is essentially no difference between the two operators if B is a node-link representation. Figure 6 shows the result.

Notice that the rows and columns of Figures 5 and 6 are analogous: the rows correspond to the choice for A, the columns to the choice of B, and in both figures the subfigures 2 and 5 are the result of using the $\oplus_{connect}$ operator rather than $\oplus_{insert}$. The $\oplus_{connect}$ operator draws the subset A as if it were "extracted" from the rest of the data, which could be useful for providing a kind of focus + context visualization.

Figure 7 pertains to compound graphs, having both a graph structure and tree structure. However, in this figure, each of the tree structures, and each of the graph structures, is shown using only *one* kind of visual representation, hence these are not hybrids in the sense of those in Figures 5 and 6. Notice also that, for tree structure, the categories "Treemap" and "Node-Link" of Figure 5 have been replaced with the

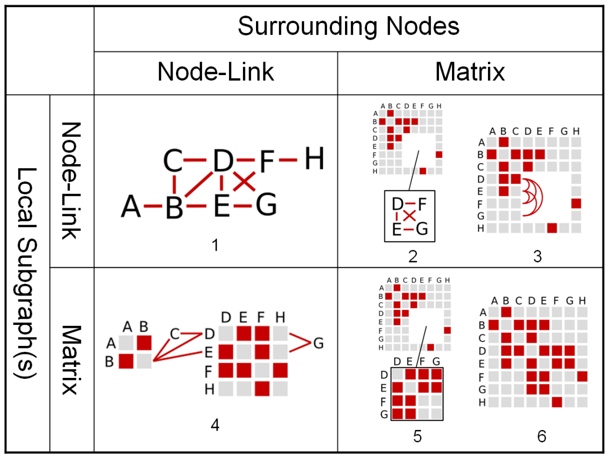

Figure 6: *Hybrid visualizations of graphs. 1 shows a node-link diagram, and 6 shows a "pure" adjacency matrix. 4 shows NodeTrix [HFM07], and 2, 3, 5 show other possible hybrids.*

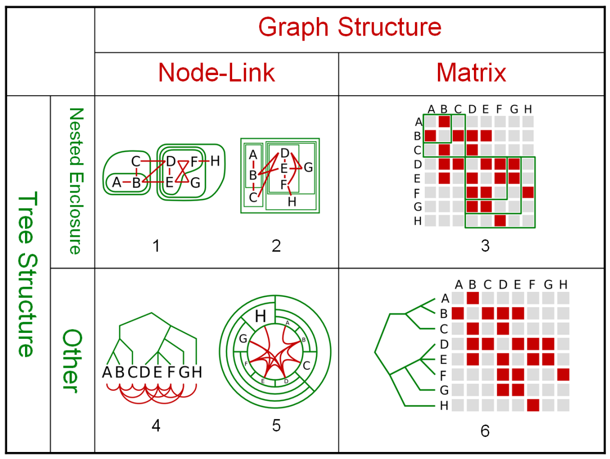

Figure 7: *Non-hybrid visualizations of compound graphs showing both graph and tree structure. The lower row, labelled "Other", shows tree structures using node-link or icicle diagrams. 1 is comparable to [Har88, SGJ93]. 2 is similar to [FWD*03]. 4 is similar to [PvW06], and also similar to TimeArcTrees [GBD09] if limited to only one moment in time, i.e., a node-link representation of a tree with arcs between the tree's leaf nodes to show graph edges. 5 is similar to the radial visualization shown in [Hol06]. 3 and 6 are approaches for showing tree structure within a matrix and are similar to Lattix [SJSJ05].*

more general categories "Nested Enclosure" and "Other", respectively, to allow for more possibilities in Figure 7.

To generate a taxonomy of *hybrid* visualizations of compound graphs, we again consider choices for the formula $A(S) \oplus B(D \setminus S)$. Let S again be a subtree $S(n)$ of some node n, which corresponds to a subset of both the tree structure and the graph structure. Notice that each of A and B may be any of the representations in Figure 7. Without even considering different choices of $\oplus$, we end up with a large space of possibilities, shown in Figure 8. Again, rows and columns correspond to choices of A and B, respectively.

Many possible hybrids exist in this space, and we have not tried to depict them. Notice simply that the non-hybrid forms of Figure 7 lie along the diagonal of this new taxonomy, whereas hybrids lie off-diagonal.

To help understand Figures 5 through 8, we have always depicted the same underlying tree and/or graph structure with 8 (leaf) nodes, and have always used green for tree structure and red for graph structure. Figure captions also cite corresponding previous work, when applicable.

To decide which hybrids within Figure 8 would be useful, we can start with an approach in the spirit of NodeTrix: subtrees that correspond to dense subgraphs should be depicted using matrices, while the surrounding graph could be depicted in node-link form. This limits us to the lower left quarter of Figure 8. Next, to depict the tree structure outside the matrices, we choose to use a nested enclosure technique, since such representations are more space-efficient, as shown in [MR10]. This further limits us to the left-most column of the taxonomy, i.e., the two starred ("*") cells in Figure 8. Notice that rather than having to choose between

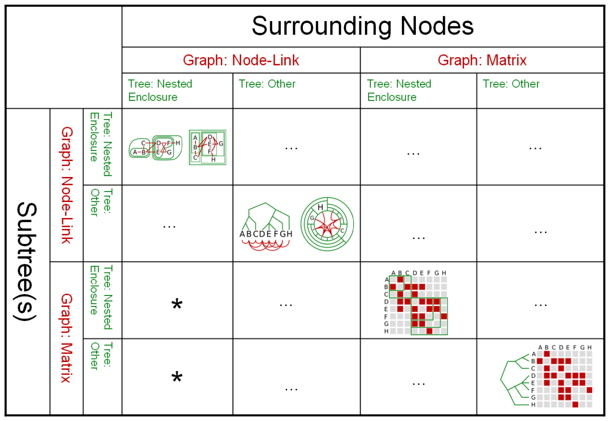

Figure 8: *Hybrid visualizations of compound graphs. The TreeMatrix in Figure 9 (middle) corresponds to the two cells in the lower-left corner shown here containing stars ("*").*

these two cells, we can happily combine the use of different tree visualizations. Finally, adding arcs along the edge of the matrix (in the spirit of MatLink) yields Figure 9 (top) for a single matrix, and Figure 9 (middle) for the entire compound graph. Notice that within each matrix in Figure 9 (middle), both the tree structure and the graph structure are shown twice, redundantly, using two techniques: nested squares and a node-link diagram of the tree, and matrix cells and arcs for the graph. The intention is that this double encoding will allow the user to attend to the most appropriate representation according to their current task and context. The combination

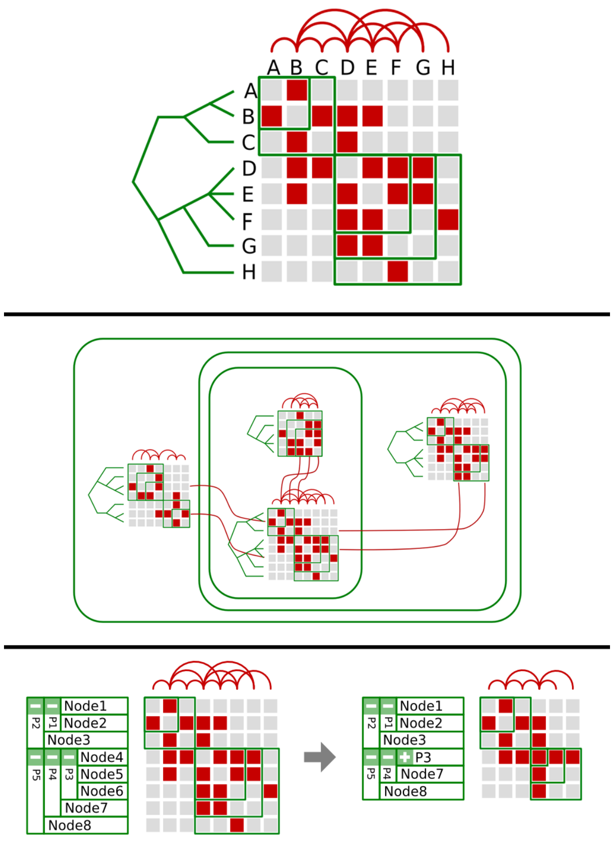

Figure 9: Top: *each matrix in TreeMatrix depicts one cluster (possibly containing sub-clusters) of nodes, corresponding to a subtree of the compound graph's tree. The tree structure is shown in green, and is shown redundantly using both a node-link diagram to the left of the matrix, and nested squares within the matrix. Graph edges are shown in red, and shown redundantly using both filled-in matrix cells and arcs along the top of the matrix.* Middle: *An entire compound graph is shown using multiple matrices, with graph edges between matrices shown in red, and the tree structure outside matrices shown with green rounded rectangles.* Bottom: *Our prototype implements a slight modification to the design, replacing the node-link tree diagrams with icicle diagrams along the left edge of each matrix (see also Figure 2). The matrix here is shown before and after collapsing the parent node P3.*

of techniques in Figure 9 is thus the basis for the TreeMatrix visualization. We next describe our implementation.

4. Prototype

Our prototype (Figures 1, 2) was implemented in Java using the Prefuse toolkit [HCL05]. The compound graph visualized is a directed graph (hence the adjacency matrices are not symmetrical) with weighted edges. Notice that along the left edge of each matrix, we display an icicle tree diagram (Figure 9(bottom)) rather than a node-link diagram (Figure 9(top)), as the icicle diagram allows labels to be displayed more easily and interferes much less with inter-matrix edges. Small +/− buttons within the icicle diagrams allow subtrees of each matrix to be expanded or collapsed. This kind of icicle diagram is also shown along the left edge of the matrix in Lattix (Figure 4).

In each matrix, the weight of each edge is shown in three ways: as a numeric value in the appropriate cell, by the background color of that cell (using a continuous grayscale), and by the color of the corresponding 180-degree arc. Between matrices, the colors of the edges again reflect their weight. The direction of edges is shown with small arrow heads. Edges drawn between matrices are drawn as cubic bezier curves. The start and end point of each of these bezier curves may be along any of the four sides (top, bottom, left or right edge) of the relevant matrix node; the side used is chosen to minimize curve length. When drawing aggregated edges, start and end points are made to coincide to bundle the edges together, reducing clutter. The user may interactively control such aggregation of edges, by collapsing matrices (e.g., the edges from the "Sys1", "Data", "Util", and "org" nodes in Figure 1 are aggregated because those nodes have been collapsed), allowing the user to reduce clutter and focus only on the nodes of interest. To further reduce clutter from edges, a mode can be activated such that only the node under the mouse cursor has its 180-degree arcs drawn, rather than all such arcs.

When the mouse hovers over a node, that node is highlighted. The user may select individual nodes by clicking, or a contiguous set of nodes in the matrix by click-dragging and using lasso gestures.

Once selected, nodes can be manipulated using actions within the radial menu [CHWS88, KB93] shown in Figure 1. Placing these actions in a radial menu rather than a toolbar has several advantages. Radial popup menus require no screen space when not in use. Also, they eliminate the need to travel back and forth between a work area and a toolbar. Finally, they can be invoked with rapid directional drags.

In our prototype, the user may also interactively collapse or expand subtrees, as well as change the location of nodes or highlight them, all to change the graphical presentation of the compound graph. Every time a new matrix is instantiated, the rows and columns can be automatically re-ordered using the barycentric algorithm [STT81, MS00], to make highly-linked groups of nodes more evident within the matrix (see section 4.2 of [Hen08] for an overview of matrix ordering algorithms). To ensure that the ordering does not violate the hierarchical grouping of nodes within the matrix, the barycentric ordering algorithm is only applied to the leaf nodes of the lowest levels of the tree of clusters. The user may manually reorder the rows and columns. All these operations allow the user to manage available screen space and represent the data in whatever way best for their task, however these operations only change the presentation of the data and do not change the *structure* of the compound graph.

In addition, the user may interactively change the *structure* of the compound graph, by selecting any subtree and moving it under a new parent node, by selecting a set of nodes and moving them under a new subtree, or by splitting or merging nodes (Figures 10, 11). This allows the user to change the way the compound graph is clustered. Such operations are useful in software engineering when performing software design discovery.

Finally, a text field in the upper right corner of the main window (Figure 1) allows the user to search for nodes by name. Matching nodes are highlighted, unless they are hidden because a parent or ancestor has been collapsed, in which case the nearest visible ancestor node is highlighted. The right margin of the main window (Figure 1) also contains an overview of the entire compound graph, which is useful for navigating a zoomed-in view in the main window (in other words, the user has a a focus + context interface).

4.1. Software Design Discovery

While any compound graph can be visualized using our prototype, we were interested in evaluating our approach in the software design domain. The data shown in Figures 1, 2, 10, 11 is the source code structure of the ApexText text editor program, available at http://sf.net/projects/apextext/. Each node is a Java class, and each directed edge is a *coupling* between the software elements, computed with a source code analysis tool. Edge weights are computed from the number of couplings between elements.

Initially, the hierarchical clustering of nodes can be automatically deduced from the organization of packages (subdirectories) of source code classes. The user can then browse the compound graph to see if the couplings (edges) either respect or violate the package architecture of the software. As described in the previous section, the user can also dynamically (re-)define the hierarchical clustering of nodes at runtime, which is useful for software design discovery where the user interprets the couplings and chooses the most logical clustering of nodes, and also useful if the user sees that the package architecture is violated by many couplings and should be reorganized.

We note that in software design discovery, the typical workflow is to progress one layer of abstraction at a time, and examine the links *between* clusters (rather than those within multiple clusters) to determine how to organize the next layer. This workflow is supported through our prototype's ability to collapse any subtree, hiding the details inside that subtree and aggregating (and bundling) edges emanating from that subtree, thereby reducing clutter and allowing the user to focus on links between that cluster and others. The ability to visualize any subtree as a matrix also reduces clutter by eliminating edge crossings within that subtree.

5. Comparison with Lattix

Lattix [SJSJ05] is a commercial software package for visualizing a compound graph representing the couplings between source code modules and their hierarchical clustering (Figure 4). Lattix is useful for software design recovery, and allows the user to dynamically change the hierarchical clustering of nodes, as does our TreeMatrix prototype. However, as mentioned earlier, Lattix visualizes the compound graph as one large adjacency matrix, called the "Design Structure Matrix" (DSM).

Several DSM analysis and visualization tools are listed at http://www.dsmweb.org. To our knowledge, Lattix is the only one of these that supports software design recovery tasks (such as manually changing the hierarchical clustering of modules), and is also the most widely-used software analysis tool that uses a matrix-based visualization. Thus, it is a reasonable candidate for comparison with TreeMatrix. In addition, to our knowledge, there are no DSM tools that allow the adjacency matrix to be split apart and visualized as multiple small matrices, as does TreeMatrix. The ability to split the matrix into multiple matrices in TreeMatrix means that users may position two or more matrices of interest close to each other to see the edges connecting them, whereas in Lattix it can be very inconvenient or impossible to see the edges connecting multiple groups of nodes, hence users of Lattix may sometimes need to do much scrolling, resulting in loss of context. The increased flexibility afforded by TreeMatrix's hybrid visualization is the primary difference between our prototype and Lattix that we are interested in evaluating.

There are several secondary differences between the TreeMatrix prototype and Lattix. For example, TreeMatrix has a popup radial menu, whereas Lattix uses traditional pull-down menus and toolbars to access functionality. Also, both software tools display the numerical weight of each edge (i.e., the number of couplings) within the appropriate matrix cell, however TreeMatrix *also* shows this weight by coloring the cell, causing interesting weight values to "pop out" visually. This means that groups of interesting edges (and their associated nodes) can be perceived faster in the TreeMatrix prototype for subsequent selection and manipulation. Finally, TreeMatrix allows for both pan and zoom to be done continuously, whereas Lattix only allows continous panning (or scrolling).

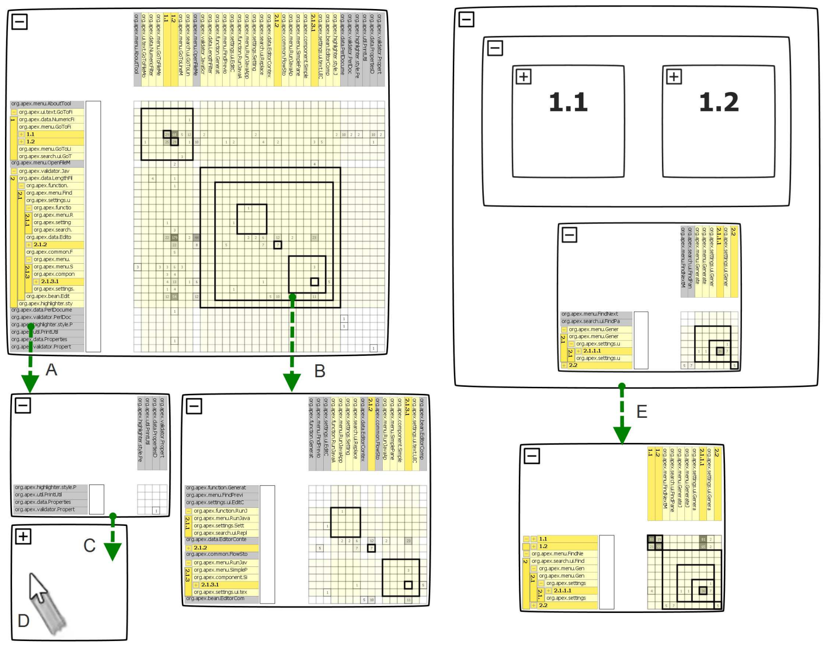

Figure 10: *Moving nodes and changing their representations.* A: *the user may select a subset of nodes (in this case, the last four nodes), and move them into a destination rounded rectangle by selecting "Move Here" in the radial menu (in this case, creating a 4×4 matrix).* B: *the user selects one subtree within the matrix (by moving the cursor over the corresponding square) and selects "To Matrix" in the radial menu, causing the subtree to be extracted as a new, smaller matrix.* C: *dragging an existing matrix or rounded rectangle inside another rounded rectangle causes the dragged cluster to be re-parented under the dropped location.* D: *dragging an existing matrix or rounded rectangle toward the border of its parent rounded rectangle has one of two effects: if dragged towards its parent, the parent increases in size and continues to contain the child; if dragged towards another parent, the child is allowed to leave the parent and is re-parented under that other rounded rectangle.* E: *the user can also transform a cluster of nodes back into a matrix.*

6. Qualitative User Study

We compared our TreeMatrix prototype with Lattix version 6.2.6 in a qualitative user study involving tasks related to software design discovery. Ten participants who were students in a master's-level course on software design received three hours of instruction on Design Structure Matrix (DSM) theory and layered architectural style for software design. The students were separated into two groups (A and B) of five students each. Two professional software engineering researchers also participated in the study bringing the total number of participants to 12. The study involved three design discovery tasks, performed on the source code of two open-source software projects (orion-ssh2 and sdedit).

6.1. Choice of Source Code Projects

We selected the data sets among the open-source projects available on SourceForge. Specifically, we sought two projects that were (1) in development for more than a year, (2) developed in Java, (3) used in different application domains, and (4) contain between 100 and 200 classes. The first project is OrionSSH2 (build 213), a library for the SSH2 protocol, and is available at http://sf.net/projects/orion-ssh2/. The second project is an UML editor program, Quick Sequence Diagram Editor 3.0.5, available at http://sf.net/projects/sdedit/.

6.2. Tasks

We chose three tasks to be done using each tool, which were performed by participants in each group during sessions that lasted 3.5 hours. In each session, the participants received 30 minutes of training (5 minutes per task, per tool) and had 1.5 hours per tool to perform the tasks and fill out questionnaires (approximately 30 minutes per task).

In the first task, the participants had to categorize the relationships :

1. Select all the nodes in the matrix and cluster them ;
2. Find a node with very little incoming links (type 1) ;
3. Find a node with very little outgoing links (type 2) ;
4. Find a cycle or type 3 category (i.e., nodes with a high density of edges along the matrix's diagonal).

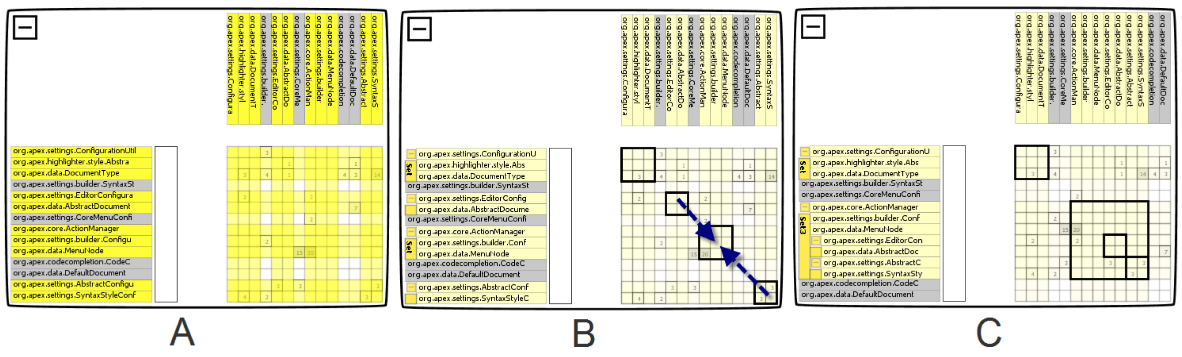

Figure 11: *Creating and editing of subtrees within a matrix.* A: *initially, this matrix contains no hierarchy. The user selects 4 different subsets of nodes and creates a subtree for each one, shown in B.* B: *tabs are visible along the left edge of the matrix. These form the icicle diagram showing the hierarchy, and indicate there are now 4 subgroups. To edit the hierarchy, the user may drag groups within their parent or into each other, resulting in C.* C: *the icicle diagram along the left edge of the matrix indicates the new hierarchical organization of groups within the matrix.*

The second task aimed at restructuring the design in a layered fashion. In a software architecture, the direction of the links in a design should be from a top layer to a bottom layer [GS93]. The participants had to :

1. Check which of the following scenarios apply :
   a) If the node of type 1 identified previously uses other elements, it is an upper layer. Move it above the first used element.
   b) If the node of type 2 identified previously is used by other elements, it is a lower layer. Move it below the last element which uses it.
   c) Group the nodes inside the cycle (type 3).
2. Find a new cycle or type 3 and group these nodes ;
3. Find a node with a strong link with the group created in the previous step and move that element in the group.

In the third task, they had to explore high level and low level links between nodes and restructure the design. They were asked to :

1. Identify the three top layers with the highest number of internal links.
2. Indicate the links between the three top layers and group the two more related layers as "TL".
3. Find the low level nodes which are responsible for the previous relationship. Group and place them under "TL".
4. Find the nodes which are linked to the "TL" group and move them into it.

7. Results

In this study, we collected user feedback, user ratings (Table 1), and task completion times (Table 2) . Users rated the usefulness of the hybrid of matrices in TreeMatrix at 2.2, on average, on a scale of 0 to 3 (unused, not very useful, useful, very useful). Some participants explained that our visual approach is advantageous in that the ability to split matrices helps to have a better view of the design elements and their relationships (P1-P6, P9). For example, participants mentioned that having multiple matrices facilitates the visualization of links between software layers (P6) and the identification of low level elements that are responsible for higher level couplings (P1).

The 180-degree arcs in TreeMatrix were found useful by 75% of the participants (P1-P5, P8, P10, P11, P12) and they noted that "Arcs help to find relationships between elements and evaluate the link strength" (P1, P2, P8, P10 all made similar statements) and that "it is a huge plus" (P1, P4).

Some users also expressed that they preferred the arrow heads showing edge direction in TreeMatrix, rather than having to deduce the direction of edges based on their location in Lattix's matrix. For instance, P4 and P9 commented that arrows are easier to follow than reading a matrix cell. Recall that, in TreeMatrix, the weight of an edge in a matrix cell is shown both numerically and with a color, whereas in Lattix it is only shown numerically. One user (P12) suggested that the TreeMatrix prototype should also show numerical weight values beside the 180-degree arcs and bezier curves that are outside each matrix, to reveal detailed information. Two users (P6, P7) asserted that the tooltips, supported in both tools, helped them to read the direction of an edge. All participants were able to learn how to use the radial menu although they had never used one before. Participants also appreciated the ability to collapse subtrees and aggregate edges to reduce the visual complexity.

7.1. Tasks Ratings and completion times

The task completion results show that tasks were, on average, faster using TreeMatrix. However, there were no statistically significant differences for tasks 1 and 3. The first two task components focus on a matrice perspective and users generally preferred Lattix for these (Table 1, Task Components 1 and 2). Participants explained that they rated TreeMatrix less positively because its non-traditional user interface made them feel less confident. Reasons given included that the prototype uses a novel radial menu, in contrast to the traditional menu bars of Lattix; and TreeMatrix also lacks keyboard shortcuts, undo, and traditional scrollbars for scrolling, which are features of Lattix. In addition, some participants (P1, P7, P10) pointed out that Lattix makes it easy to see the diagonal of the matrix at all times, which can help to find cycles. We find interesting that the more classical approach of Lattix in tasks 1 and 2 was not faster, despite the unfamiliar aspects of TreeMatrix that the users had to learn. The second task was significatively faster using TreeMatrix and we believe this is due to the highlighting and the 180-degree arcs that were appreciated.

The ratings for the 3rd task component show that TreeMatrix was preferred for analyzing links at a higher level of abstraction. This is backed up by some participants commenting that TreeMatrix is better for qualitative analysis (P2, P10). The TreeMatrix prototype was also appreciated with respect to highlighting of edges and nodes (the 6th component in Table 1). Participants explained that "the colored edges and the inter-matrix links really help" (P1, P6) and commented that the TreeMatrix prototype is more visual, helping to perform the tasks (P3, P7, P9). The task completion results show that TreeMatrix was not significantely faster for the third task. The comments suggest that some users have spent more time in visually exploring the data set and playing with the interface.

We believe that certain modifications could be easily made to the TreeMatrix prototype to increase its usefulness to users: implementing undo, keyboard shortcuts, optional traditional scrollbars and pull-down menus, making the diagonal of each matrix always easier to see, and displaying the edge weights of bezier curves and arcs with numerical labels (possibly displayed in a tooltip).

7.2. Threats to validity

We first consider the internal validity (i.e., the confidence level of the results) of the user study. We tried to prevent the effects of confounding variables, between a tool (independent variable) and a dependent variable (e.g., user ratings, time). Participants received the same training for both approaches. None of the ten students had any prior experience with TreeMatrix or Lattix, while the two researchers had experience using Lattix. The participants had no prior exposure of the open-source projects. To counterbalance the conditions, the participants of each group were randomly chosen, with group A performing the three tasks using Lattix on orion-ssh2 and then using TreeMatrix on sdedit. Group B performed the tasks on the same data sets, but started with TreeMatrix and then with Lattix.

	Task Component	Lattix	TreeMatrix
1	Perception of relationships within matrices (Tasks 1, 2)	8.50	5.75
		p = 0.0051* (n=12)	
2	Ability to reorganize matrices (Task 2)	8.50	6.17
		p = 0.0054* (n=12)	
3	Identification of high-level links (links between clusters) (Task 3)	6.75	8.17
		p = 0.0381* (n=12)	
4	Identification of low-level links (links between classes) (Task 3)	7.42	7.42
		p = 0.9682 (n=12)	
5	Ability to reorganize entire clusters at a high level (Task 3)	7.58	7.58
		p = 0.9683 (n=12)	
6	Highlighting of nodes and edges (Tasks 1-3)	6.80	7.80
		p = 0.1649 (n=10)	

Table 1: *Average ratings by participants of how each tool performed for components of the tasks, on a scale of 1 (Poor) to 10 (Excellent), along with the p-values of the Wilcoxon signed rank test.*

	Task	Lattix	TreeMatrix
1	Find clusters in a matrix	14.25	11.63
		p = 0.4349 (n=8)	
2	Reorganize nodes inside a matrix	25.11	18.22
		p = 0.0116* (n=9)	
3	Restructure the design	20.92	19.70
		p = 0.1381 (n=10)	

Table 2: *Average timings by participants of how each tool performed with respect to tasks (in minutes), along with the p-values of the Wilcoxon signed rank test.*

We statistically verified the differences in the dependent variables. We verified that we could perform the tasks for all projects with a similar effort and time. Given the exploratory nature of the tasks and the real-world complexity in the data, we focused on collecting the impressions of the participants after performing useful exploratory tasks and did not evaluate the error rates. However, the user feedback that we collected suggest that there were no significant differences among the tools. The participants had to write down their timings at each substep and we excluded any uncertain data.

We now turn to the external validity (the generalizability of the study). Real and complete projects, from different domains, were evaluated by the participants in all their complexity, which should ensure that the approach has a practical value for other projects. Since a small number of partipants were involved, it should be repeated with more participants. The prototype, demonstrated in the supplementary video[†], has been tested with software designs of a few thousands of nodes and links on a Sager NP8150 laptop. We valided one hybrid approach, but further research is needed to evaluate the other possible hybrids that can be derived from our taxonomy againsts a larger number of tools and domains.

[†] The video is available online at http://ref.rufiange.com/cgf2012.

8. Conclusions and Future Directions

We have presented new taxonomies of hybrid visualizations of trees, graphs, and compound graphs, and shown how these taxonomies lead to a novel hybrid visualization of compound graphs called TreeMatrix. TreeMatrix combines advantages of NodeTrix and MatLink (which, in their original form, are not designed for compound graphs) with the matrix visualization of Lattix. Our hybrid visualization allows users to represent selected clusters of the compound graph as matrices, with each matrix providing a visual summary of its contents, free of edge crossings, and containing a double encoding of both the tree and graph structure within the matrix.

We have also presented a prototype implementation of TreeMatrix, and demonstrated how it can be useful for software design discovery tasks, such as visualizing the structure of source code. Our prototype displays both low-level and high-level abstractions (within and above the level of matrices, respectively) in one integrated view, and allows the user to dynamically edit tree clusters (inside a matrix or at an upper level). We reported the results of a comparative evaluation of our prototype with Lattix, a commercial-grade, matrix-based software tool. Several users found our prototype's ability to split matrices into multiple submatrices advantageous, and found it either useful or preferable that our prototype displays edges as arcs or curves with arrowheads. User ratings indicate that they preferred our prototype in particular for interpreting high-level links between clusters. This confirms the rationale behind using a hybrid visualization: TreeMatrix has the flexibility of enabling low-level details to be shown within matrices (eliminating edge crossings) while allowing high-level edges to be shown in node-link form (which are easier to interpret, if they are not too dense or numerous).

Based on some of the comments we collected, one possible future direction would be to allow the user to switch between novice or advanced modes to access a different set of features. There are also improvements that could be made to the computed layout and bundling of edges, to make the TreeMatrix visualization even less cluttered. Future user studies could also examine in more detail the kinds of tasks [LPP*06] that benefit from hybrid visualizations, e.g., comparing hybrid and non-hybrid visualizations in a follow-up to Archambault et al. [APP10]. Another future direction would be to implement an interactive prototype flexible enough to allow the user to explore all possible hybrid combinations in Figure 8, with automatic generation of appropriate layouts and even smoothly animated transitions during changes to the visualization. Although many of the hybrids generated this way might not be very useful, there is the possibility that new useful forms could be discovered this way.

9. Acknowledgements

We thank the participants in our study and the anonymous reviewers for their valuable comments. This work was funded by NSERC, and doctoral scholarships from Fonds de recherche du Québec and ÉTS.

References

[ACJM03] AUBER D., CHIRICOTA Y., JOURDAN F., MELANÇON G.: Multiscale visualization of small world networks. In *Proceedings of the Ninth annual IEEE conference on Information visualization* (Seattle, Washington, 2003), INFOVIS'03, IEEE Computer Society, pp. 75–81. 3

[AMA09] ARCHAMBAULT D., MUNZNER T., AUBER D.: TugGraph: Path-preserving hierarchies for browsing proximity and paths in graphs. In *Proceedings of IEEE Pacific Visualization* (Beijing, China, 2009), pp. 113–120. 3

[APP10] ARCHAMBAULT D., PURCHASE H. C., PINAUD B.: The readability of path-preserving clusterings of graphs. *Computer Graphics Forum 29*, 3 (2010), 1173–1182. 11

[AvHK06] ABELLO J., VAN HAM F., KRISHNAN N.: ASK-GraphView: A large scale graph visualization system. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 12*, 5 (September/October 2006), 669–676. 3

[BD07] BALZER M., DEUSSEN O.: Level-of-detail visualization of clustered graph layouts. In *Asia-Pacific Symposium on Visualisation (APVIS)* (Sydney, Australia, 2007). 2, 3

[Ber83] BERTIN J.: *Semiology of graphics: diagrams, networks, maps*. University of Wisconsin Press, Madison, Wisconsin, 1983. 2, 3

[BHvW00] BRULS M., HUIZING K., VAN WIJK J. J.: Squarified treemaps. In *Proceedings of the joint Eurographics and IEEE TCVG Symposium on Visualization* (Amsterdam, The Netherlands, 2000), pp. 33–42. 3

[BM99] BERTAULT F., MILLER M.: An algorithm for drawing compound graphs. In *Proceedings of Symposium on Graph Drawing* (Stirín Castle, Czech Republic, 1999), pp. 197–204. 3

[CHWS88] CALLAHAN J., HOPKINS D., WEISER M., SHNEIDERMAN B.: An empirical comparison of pie vs. linear menus. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)* (Washington, DC, 1988), pp. 95–100. 6

[DBETT99] DI BATTISTA G., EADES P., TAMASSIA R., TOLLIS I. G.: *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, Upper Saddle River, N.J., 1999. 3

[DBF09] DI BATTISTA G., FRATI F.: Efficient c-planarity testing for embedded flat clustered graphs with small faces. *Journal of Graph Algorithms and Applications 13*, 3 (Nov 2009), 349–378. Special Issue on Selected Papers from GD '07. 2

[EF10] ELMQVIST N., FEKETE J.-D.: Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. *IEEE Transactions on Visualization and Computer Graphics 16*, 3 (2010), 439–454. 3

[FWD*03] FEKETE J.-D., WANG D., DANG N., ARIS A., PLAISANT C.: Overlaying graph links on treemaps. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis) Poster Compendium* (Seattle, Washington, 2003), pp. 82–83. 5

[GBD09] GREILICH M., BURCH M., DIEHL S.: Visualizing the evolution of compound digraphs with TimeArcTrees. *Computer Graphics Forum 28*, 3 (June 2009), 975–982. 3, 5

[Gen00] GENE ONTOLOGY CONSORTIUM: Gene ontology: tool for the unification of biology. *Nature Genetics 25*, 1 (May 2000), 25–29. http://www.geneontology.org (Accessed January 18, 2012). 2

[GFC05] GHONIEM M., FEKETE J.-D., CASTAGLIOLA P.: On the readability of graphs using node-link and matrix-based representations: Controlled experiment and statistical analysis. *Information Visualization 4*, 2 (2005), 114–135. 2, 3

[GKN05] GANSNER E. R., KOREN Y., NORTH S. C.: Topological fisheye views for visualizing large graphs. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 11*, 4 (2005), 457–468. 2, 3

[GS93] GARLAN D., SHAW M.: An introduction to software architecture. *Advances in Software Engineering and Knowledge Engineering 2* (1993), 1–39. 9

[Har88] HAREL D.: On visual formalisms. *Communications of the ACM (CACM) 31*, 5 (May 1988), 514–530. 5

[HCL05] HEER J., CARD S. K., LANDAY J. A.: Prefuse: A toolkit for interactive information visualization. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)* (Portland, Oregon, 2005), pp. 421–430. 6

[Hen08] HENRY N.: *Exploring Social Networks with Matrix-based Representations*. PhD thesis, Université Paris Sud, France, and University of Sydney, Australia, 2008. 6

[HF07] HENRY N., FEKETE J.-D.: MatLink: Enhanced matrix visualization for analyzing social networks. In *Proceedings of IFIP TC13 International Conference on Human-Computer Interaction (INTERACT)* (Rio de Janeiro, Brazil, 2007), pp. 288–302. 2, 3

[HFM07] HENRY N., FEKETE J.-D., MCGUFFIN M. J.: NodeTrix: A hybrid visualization of social networks. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 13*, 6 (2007), 1302–1309. 2, 3, 5

[HMM00] HERMAN I., MELANÇON G., MARSHALL M. S.: Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 6*, 1 (January 2000), 24–43. 3

[Hol06] HOLTEN D.: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 12*, 5 (2006), 741–748. 3, 5

[JS91] JOHNSON B., SHNEIDERMAN B.: Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Proceedings of IEEE Visualization (VIS)* (San Diego, CA, 1991), pp. 284–291. 3

[JS10] JÜRGENSMANN S., SCHULZ H.-J.: A visual survey of tree visualization. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis) Poster Compendium* (Salt Lake City, Utah, 2010). http://treevis.net/ (Accessed January 18, 2012). 2

[KB93] KURTENBACH G., BUXTON W.: The limits of expert performance using hierarchic marking menus. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI)* (Amsterdam, The Netherlands, 1993), pp. 482–487. 6

[KW01] KAUFMANN M., WAGNER D. (Eds.): *Drawing Graphs: Methods and Models*. Springer, Berlin, New York, 2001. 3

[LPP*06] LEE B., PLAISANT C., PARR C. S., FEKETE J.-D., HENRY N.: Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization* (Venice, Italy, 2006), BELIV '06, ACM, pp. 1–5. 11

[MR10] MCGUFFIN M. J., ROBERT J.-M.: Quantifying the space-efficiency of 2D graphical representations of trees. *Information Visualization 9*, 2 (2010), 115–140. 3, 5

[MS00] MÄKINEN E., SIIRTOLA H.: Reordering the reorderable matrix as an algorithmic problem. In *International Conference on the Theory and Application of Diagrams* (Edinburgh, UK, 2000), pp. 453–468. 6

[PvW06] PRETORIUS A. J., VAN WIJK J. J.: Visual analysis of multivariate state transition graphs. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 685–692. 3, 5

[San96] SANDER G.: *Layout of Compound Directed Graphs*. Tech. Rep. A/03/96, Universität des Saarlandes, Saarbrücken, Germany, June 1996. 3

[SGJ93] SINDRE G., GULLA B., JOKSTAD H. G.: Onion graphs: Aesthetics and layout. In *Proceedings of IEEE Symposium on Visual Languages (VL)* (Bergen , Norway, 1993), pp. 287–291. 5

[SJSJ05] SANGAL N., JORDAN E., SINHA V., JACKSON D.: Using dependency models to manage complex software architecture. In *Proceedings of ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)* (San Diego, CA, 2005), pp. 167–176. 2, 3, 5, 7

[SM91] SUGIYAMA K., MISUE K.: Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man and Cybernetics 21*, 4 (July/August 1991), 876–892. 3

[SM07] SHEN Z., MA K.-L.: Path visualization for adjacency matrices. In *Proceedings of Eurographics/IEEE-VGTC Symposium on Visualization (EuroVis)* (Norrköping, Sweden, 2007), pp. 83–90. 3

[STT81] SUGIYAMA K., TAGAWA S., TODA M.: Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics SMC-11*, 2 (February 1981), 109–125. 6

[vLKS*10] VON LANDESBERGER T., KUIJPER A., SCHRECK T., KOHLHAMMER J., VAN WIJK J. J., FEKETE J.-D., FELLNER D. W.: Visual analysis of large graphs. In *EuroGraphics: State of the Art Report* (Norrköping, Sweden, 2010). 3, 4

[Wat02] WATTENBERG M.: Arc diagrams: Visualizing structure in strings. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)* (Boston, Massachusetts, 2002), pp. 110–116. 3

[Wat05] WATTENBERG M.: A note on space-filling visualizations and space-filling curves. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)* (Minneapolis, Minnesota, 2005), pp. 181–186. 3

[ZMC05] ZHAO S., MCGUFFIN M. J., CHIGNELL M. H.: Elastic hierarchies: Combining treemaps and node-link diagrams. In *Proceedings of IEEE Symposium on Information Visualization (InfoVis)* (Minneapolis, Minnesota, 2005), pp. 57–64. 2, 3, 4