Visualizing Graphs - A Generalized View

Hans-Jörg Schulz, Heidrun Schumann
Department of Computer Science, University of Rostock, 18051 Rostock, Germany
{hjschulz, schumann}@informatik.uni-rostock.de

Abstract

The visualization of graphs has proven to be very useful for exploring structures in different application domains. However, in certain fields of computer science, graph visualization is understood and focused quite differently. While "graph drawing" focuses on optimized layouts for node-link-representations of networks, "information visualization" prefers to work on hierarchies focusing on very large structures, different views and interactivity.

This paper gives a systematic view of the problem of graph visualization by combining both approaches. We will introduce a general view of different visualization methods as well as describe occurring problems and discuss basic constraints. These will be used to propose a visualization framework for graphs, whose development motivated this paper.

Keywords— **Network Visualization, Graph Drawing, Exploratory Graph Analysis**

1 Introduction

For a long time, visualizing structures is a hot topic in the domain of information visualization. Main objectives are the handling of very large data sets as well as providing functionality to interactively exploring them. A number of customized methods for visualizing structures have been developed. Generally, we distinguish between methods presenting hierarchical structures and methods for more general classes of graphs. However, most of the recent work is concentrating on presenting hierarchies. Here, the ideas of the *information seeking mantra* "Overview first, Zoom and Filter, then Detail on Demand" [23] can be effectively realized by using adaptive techniques and information hiding. Thus, we may start with an overview image, showing the first levels of the hierarchy only, and refine it for regions of interest, e.g. by an interactive folding and unfolding of subtrees.

On the other side, in the domain of graph drawing, there exists a wide variety of different layout techniques for graphs. Here, the focus of many applications and libraries like JUNG (*http://jung.sourceforge.net*) or JGraph (*http://www.jgraph.com*) lies upon optimized layouts for static, medium-sized, sparse node-link-representations, rather than upon interactivity or providing multiple views at different levels of detail.

Combining both strategies would significantly increase the importance of visual techniques to solve problems in many application domains, where complex structures have to be analyzed, e.g. in systems biology, electrical circuits or social networks.

A first step on this way could be a systematic view of the variety of techniques for visualizing structures from both domains, information visualization and graph drawing. For visualizing hierarchies there exist classification schemes [13] as well as studies to analyze and compare the different approaches [5, 25].

In this paper we want to give a taxonomy of network presentation techniques (see section 2). Moreover, we want to discuss the different constraints to be considered, when visualizing complex structures (see section 3). Finally, we will show, how the different approaches can be combined in a framework for exploring complex graphs (see section 4).

2 Taxonomy

Over the last years a great number of powerful methods for visualizing structures have been developed. Since different methods focus on different aspects, choosing the right one can be a difficult task. To support the selection of appropriate visualization techniques, classification schemes have been proven to be very helpful, since they clarify the fundamental features of the different approaches. In doing so, they allow comparisons and evaluations of diverse techniques.

In general, an important criterion in classifying visualization methods is, whether the layouts are realized in 2- or 3-dimensional presentation space. 2D-techniques are intuitive and widely used, which is true for graph visualization techniques as well. However, with the growing amount of data on one side, and with the growing capabilities of modern graphic cards on the other side, even 3D-techniques become popular. In 3D presentation space we have an addi-

tional axis for information encoding. This extra dimension can either be used to encode attributes of nodes/edges in a so called "$2\frac{1}{2}$D" network representation, as it is discussed in [7], or to lay out large data sets within three dimensions. However, there are also some problems with these 3D-layouts like occlusions, perspective distortions, difficulties in the interpretation of location and orientation of lines and so on. Therefore, the choice between 2D- or 3D-representations has to be made with care. Figure 1 shows examples for two 3D- and four 2D-graph-layouts.

In addition to this general criterion of dimensionality, we can classify visualization techniques for structures by more specific aspects. This will be discussed in the next sections. First in section 2.1, we will briefly describe a taxonomy of visualization methods for hierarchies. Afterwards in section 2.2 we will introduce our taxonomy for network visualization techniques.

2.1 Hierarchy Representations

There are two principal alternatives to classify visualization techniques for hierarchies:

- explicit vs. implicit
- axes-oriented vs. radial

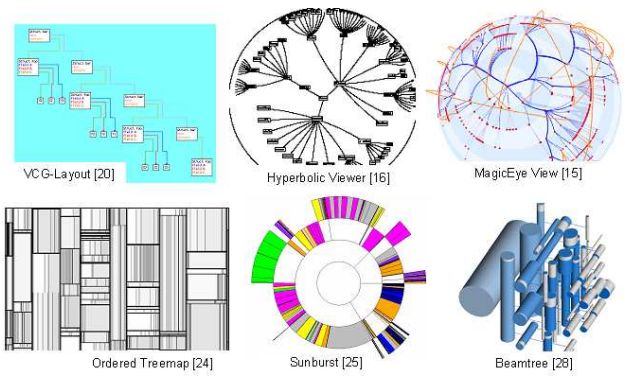

Figure 1: Examples of hierarchical graph representations — derived from [22]

Explicit vs. Implicit: Explicit methods display the edges between the elements of the hierarchy in the traditional node-link-representation (see upper row of figure 1). Implicit techniques are space-filling methods. They use a wide variety of abstract node representations (lines, boxes, circles, etc.) and indicate their relation by subtle arrangements of these elements (inclusion, intersection, adjacency, etc.). For examples see the lower row of figure 1.

A user study comparing explicit and implicit representation techniques according to their usability for different tasks was presented in [5]. As a representative of explicit techniques, the Organization Chart was chosen. Moreover, 3 implicit techniques (Icicle Plot, Treemap, and Tree Ring) were considered. The comparison was based on such aspects as: ease of interpretation, node size and user preferences. During the tests, the Icicle Plot achieved nearly the same results as the Organization Chart. Tree Ring (see Sunburst in figure 1 to get an impression) and especially Treemap (see figure 1) required more effort. However, it was stated that none of the views was clearly superior to the other views. The achieved results were related to a combination of the given tasks and the view. Explicit techniques are advantageous, when solving node description tasks. Implicit techniques like Tree Ring are effective, when performing node size tasks.

It is of course also possible to combine both approaches and thus try to make use of the advantages of both. A recent technique of this kind combining Treemaps and Node-Link-Representations are the Elastic Hierarchies [30].

Axes-oriented vs. Radial: Axes-oriented layouts arrange elements line by line. One line depicts the nodes of one level of the hierarchy. Typically, the lines are arranged parallel to the axes. Radial layouts present concentric circles with the root of the hierarchy in the midpoint. The main advantage of radial layouts is the automatically increasing display space for each layer of the hierarchy. This advantage is reflected in the outstanding rating the radial technique Sunburst got at the comparative user study in [25].

While the most widely used tree visualization techniques belong to either one of these two categories, there are a few exotic tree visualization techniques using more unrestrained layouts that do not fit into this binary classification scheme (i.e. Vornoi Treemaps [4] or Botanical Trees [14]). Yet due to their small number and their limited practical impact, a third layout category containing such free layouts is usually not considered.

2.2 Network Representations

While the organization of data in a hierarchical manner is a powerful pattern used frequently throughout many fields of application, hierarchies and especially trees are just a tiny subclass compared to the huge graph class of networks. That is why adequately representing networks is in most cases a much more difficult task than depicting hierarchies. Networks for example do not have the distinct top-to-bottom-ordering of trees that is exploited heavily by tree visualization techniques to generate a suitable layout. Yet in many application domains, it is common to direct the network's edges and thus impose an ordering upon it. Therefore, we distinguish between visualization methods for plain networks and such suitable to visualize directed graphs. Additionally, network visualizations can be differentiated between explicit and implicit, or according to the

degree of predetermination of their node layout. The latter is of particular interest, since it generalizes the notion of axis-oriented and radial layouts discussed in section 2.1, which are not applicable to network visualizations. Thus, we propose three possible categorizations for visualization techniques for networks:

- directed vs. undirected
- explicit vs. implicit
- free, styled or fixed

Directed vs. Undirected: Attributing the edges of a network with directional information is common in many fields of application, where it is mostly used to model notions of flow or dependency. In contrast to the layout of undirected networks, directed graphs pose an additional challenge to the layout process, as the directions of the edges should be integrated in the visualization. For node-link-representations, this is usually done by adding arrowheads to the edges indicating their direction. Another alternative to encode the directionality of edges is to arrange the graph in a way, that allows to deduce the directions easily from the layout itself, as it is done e.g. by the DIG-COLA method [8].

Explicit vs. Implicit: The distinction between explicit and implicit graphing techniques for networks runs along the same line, as shown in section 2.1. Yet, while implicit tree visualizations usually exploit the characteristic hierarchical, acyclic topology of trees, implicit network visualizations cannot take advantage of such structural restrictions. There are essentially three ways to cope with this additional difficulty:

- use a matrix-based implicit layout, since matrices are well know to admit every kind of graph – examples are introduced in [1, 19],
- use an implicit tree layout to depict a spanning tree of the network and augment it with the remaining edges, as it can be done using techniques presented in [11, 18],
- check the network for certain structural characteristics and exploit them, i.e. a planar network can be depicted using a tesselation representation [26] or an interval graph can be represented by segments of a straight line [17].

Figure 2 respectively shows an example for each one of the possibilities given above. However, because it has proven to be quite difficult to implicitly depict networks, up to now node-link representations are the most common way to display graphs (for a comprehensive overview see [6]) and implicit techniques are far from being as widespread and accepted as for tree visualizations. That might be the reason that user studies comparing the usefulness of implicit and explicit network layouts for different tasks are very rare. Thus far, there exists only one comparison between matrix-based and explicit layout techniques [12].

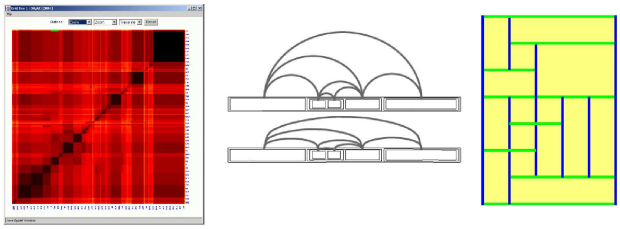

Figure 2: Examples for implicit network representations (from left to right): the matrix-based technique Graph Sketch [1], two ArcTrees [18] and the tesselation representation of a planar graph [6]

Free, Styled and Fixed: Yet another way to classify network visualization techniques is to use the degree of freedom for node layouts within the presentation space. In most cases, the node layout is not restricted (*free layout*) and can thus be generated with a wide variety of methods (i.e. force-directed placement [6]) On the other hand, there are cases where the node layout is entirely *fixed* and only the run of the edges can be determined freely by the technique. This occurs mostly for geospatial visualization techniques, i.e. as it is known from airway route maps. But there are also a few implicit techniques for which an alteration of the fixed node layout would result in discrepancies to the originally represented graph topology. An example for such a case is the permutation diagram of a permutation graph [10]. Apart from free and fixed layouts, there are less restricted, yet not entirely free layouts called *styled layouts*. That means the node layout confines to a certain predefined scheme, i.e. a grid. Using a styled layout has the advantage that the overall style of the graph visualization is known even before the actual layout is computed. This knowledge can be used to estimate the screen space needed to display the graph and to preselect fitting interaction methods that are known to work well on a certain style. Figure 3 shows a graph in two common styled node-link-layouts:

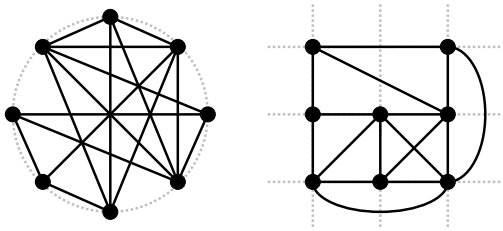

Figure 3: Two examples for styled network layouts: circular (left) and grid (right).

Since trees can be conceived as a certain type of network, it is not surprising that this layout-based differentiation of graph visualization techniques works just as well for tree visualizations. There the only limitation is that merely two styled layouts are used in practice: axis-oriented and radial.

3 Constraints on Network Visualization

As the above sections have shown, there is an abundance of visualization techniques available for all kinds of graph topologies, application domains and visualization goals. So today's question is not whether there exists a suitable visualization technique, but which of them to choose. The following sections discuss a number of constraints that influence the decision for a certain visualization method on different levels. All of the constraints mentioned in the following sections must be considered in order to yield a meaningful and usable visualization.

3.1 User Constraints

The following examples are constraints that can be derived from assumptions about interactive tasks that the user will be likely to carry out on the final visual representation. That means, the *information seeking mantra* has to be supported, providing different views on the given graph at different levels of granularity to solve different tasks.

Compatibility of visualization and user: In some fields of application, certain visualization methods are more common and better understood than others. Since at the end, the user with the scientific background from her domain of research has to work with the chosen visualization, this should be an important consideration when choosing a visualization technique. Hence, when evaluating visual representations user preferences are a main aspect, e.g. [5].

Well known examples are the standards for technical drawing, that require i.e. to depict conducting paths in a circuit diagram as *orthogonal edges of a node-link-representation*.

Compatibility of visualization and interaction techniques: Typically the user wants to explore the structure of a graph in depth, and thus we have to provide different views at interactive frame rates. Therefore, the graphical representation of a graph needs to be generated with respect to the limited time-resources, rather than focusing on optimized layouts. Moreover, the desired techniques of interaction have to be easily combinable with the underlying graph visualization to ensure a high degree of usability. That means, e.g. searching for details of interest has to be performed without losing the orientation within the general network. This is one of the main problems, when exploring large structures. Therefore, often different representations for overview and detail are linked together in such a way that manipulating one view automatically updates the others. One commonly used manipulation technique in this context is the identification of certain substructures (i.e. shortest paths or cliques). That means, the user searches for some structures of interest in one view, and these structures are also accentuated in the other views e.g. by highlighting or color-coding. Working with different views at once, the user can effectively change the representations to choose the best for the task at hand.

This means for example, that a *node-link-representation* of a road network would be well suited to navigate through it, because it is highly compatible with the interaction techniques used for navigation (highlighting of edges to indicate paths, etc.) Yet, a civil engineer who has to plan and to maintain the road network, might be served better by a compact *matrix-based representation* that can easily be rearranged, grouped and sorted.

Compatibility of different visualization techniques: Sometimes, the focus of the graph analysis does not lie upon the structure of the graph, but on certain attributes associated with the nodes or edges. To reflect this focus, the visualization has to provide extra space to visualize the attributes with different visualization methods – each customized to the nature of one attribute of interest. An example for such a combination of graph and attribute visualization is the so called Needle Grid [2].

Generally, *explicit techniques* are advantageous in case of edge attributes to be visualized, since they explicitly draw the edges and leave enough possibilities for graphically encoding the attributes (color coding, dashing, etc.). On the other side, *implicit techniques* are superior to explicit ones when it comes to annotating nodes with certain attributes, because no extra space is wasted for drawing edges and all display space is available to depict the nodes and their attributes. This is valid up to the point, when more information about each attribute needs to be displayed than can be represented within the space a node occupies. Above that level, again *explicit node-link-representations* are a better choice, since they leave a lot of space in between the nodes that can be used for annotation.

3.2 Data Constraints

While the user constraints already rule out a number of inappropriate visualization methods, they do not take any properties of the network data itself into account. Among these are:

Static Data vs. Dynamic Updates: While most sets of data can be acquired and stored completely before the analysis starts, some fields of application work with highly

volatile data that changes with every timestep. When for example analyzing a peer-to-peer net, every now and then a peer quits its connection (a node and its incident edges are deleted), while others join the net (a new node is added).

If the user requires these dynamic updates of the structure to be taken into account, then it would force the visualization method to be computed relatively fast. Furthermore, it would clearly be a poor choice to visualize a dynamically changing graph using a *topology-dependent layout*, since the graph's planarity or acyclicity is subject to change with every update operation. However, a rather static, maybe *styled* arrangement of the nodes is needed, which changes only locally and maintains the overall layout as minor changes on the underlying data occur. This allows the user to preserve her mental image of the general graph structure and to orientate herself during the process of analysis [9].

Figure 4 shows two *radial layouts* of a dynamically changing hierarchy before and after an additional subtree gets attached to it. The left image shows the application of the Walker algorithm [29], adapted to a radial layout. After dynamically attaching the subtree, the original layout is almost completely lost, and thus the mental map is hard to maintain. The right image shows an even further adapted version, the Radial Walker algorithm, that abandons the optimized space-filling strategy. However, here the original structure is recognizable after dynamically attaching the subtree. Thus, the orientation of the user is preserved.

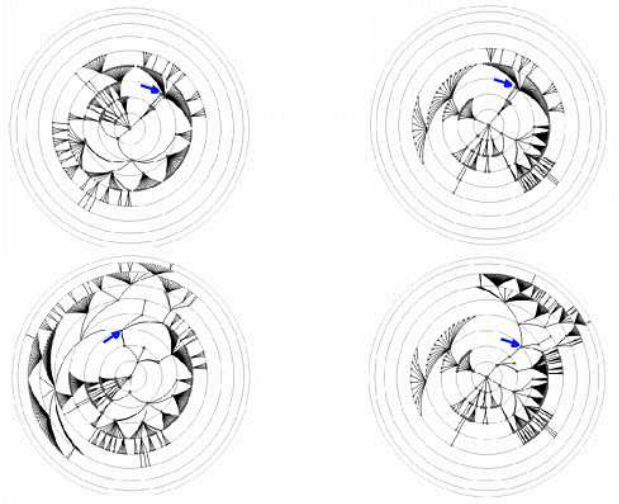

Figure 4: Two examples for unfolding hierarchies – on the left side with an optimal space-filling strategy, on the right side with preserving the orientation

Similar considerations can be made for interactive techniques that allow the folding of subtrees of a hierarchy or to aggregate clustered nodes in a network, since the effect is the same: nodes are constantly disappearing and being reintroduced to the layout.

Size and Density of the Graph: A main problem of graph visualization is the concealment of structural information, which cannot be avoided especially for large and dense graphs. This problem is commonly known as the so called "screen bottleneck", and it occurs at the latest when the number of nodes to display at once exceeds the number of available pixels on the screen. Hence, techniques to circumvent the lack of adequate display space need to be applied. Among others, the user could utilize overview+detail, clustering or information hiding techniques. However, as already discussed in section 3.1 not every visualization technique is suitable to be combined with every possible technique from this list, which results in additional constraints for the selection of a suitable graph visualization.

A variety of visualization methods for large graphs already assume some kind of preprocessing like a hierarchical clustering of the graph as additional input. A powerful example of this kind is the *matrix-based network visualization* technique Graph Sketches [1] which supposedly can handle networks with up to 250 million vertices. In general, *implicit representations* have the advantage that more nodes can be depicted, since they are screen-filling techniques, that do not use up screen space to draw edges.

Graph Topology: As already discussed in the above sections, some visualization techniques (especially *implicit* or *hierarchical methods*) admit only graphs of a certain topology. These visualizations have the advantage that they emphasize the special topology of a graph by means of a well adapted layout technique.

This restriction is often not tight, which means, that i.e. a network with only a few so called "cross edges" can still be depicted with a wide variety of hierarchical layout techniques. To do so, its spanning tree is laid out hierarchically and the cross edges that form induced cycles within the network and thus destroy the hierarchical topology of the graph are drawn on top of the hierarchical layout. This way of representing *treelike* networks emphasizes the underlying topology and eases the identification of cross edges by coloring them differently. An example for this technique using a Treemap as underlying hierarchical layout is discussed in [11] and is depicted on the left of figure 5.

Another way to adequately depict the topological structure of a network is to divide the network in subgraphs and to visualize these subgraphs according to their special topology. A recent technique following this approach has been presented in [3] and is shown on the right side in figure 5.

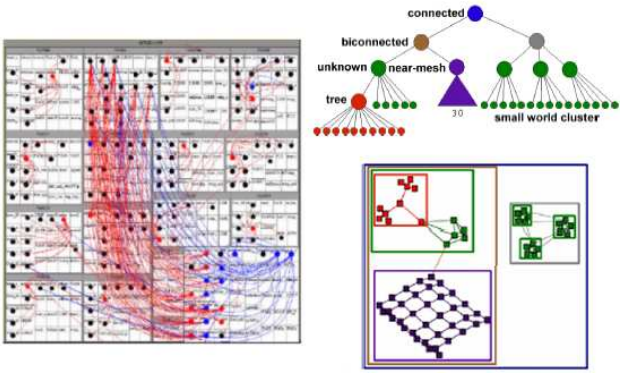

Figure 5: Two examples for topological visualization. Left: Treemap with cross edges [11], Right: TopoLayout [3]

3.3 Aesthetic Constraints

Finally the hardest problem when visualizing graphs is to find an appropriate layout in the 2D- or 3D-space. Some of the most prominent aesthetical criteria for a desirable graph layout are:

- small number of edge crossings
- small area of drawing
- small number of bends along the edges
- small but uniform edge lengths

Thus, the selection and parametrization of an appropriate algorithm for the graph layout is influenced by several constraints, i.e. the minimization of edge crossings or nearby layout for adjacent nodes to preserve structural proximity. As shown in figure 6, these constraints often even contradict each other, so that the user needs to prioritize or weigh the constraints according to her aesthetic preference or an informally stated visualization goal.

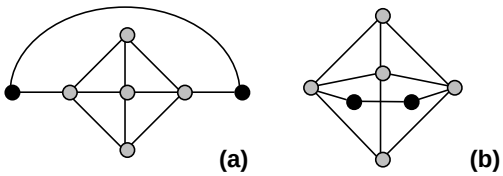

Figure 6: While layout (a) minimizes the number of edge crossings, it distorts the structural proximity between the two black nodes. Layout (b) graphically preserves this structural proximity, but trades an additional edge crossing to do so.

Further problems arise, when additional node- or edge-labels need to be integrated in the layout. All of this contributes to make the computation of network layouts a complicated and algorithmically complex task.

4 Assembling the Pieces

Many of the ideas mentioned in the above sections actually stem from a domain-independent framework for graph visualization that we developed [21]. In fact, this framework serves as a perfect example which finally puts together the presented ideas and proves their usefulness when combined with each other. Since graph-like structures occur in many fields of application, the computer aided visualization and interactive analysis of large graph structures plays an increasingly important role for modern scientific research.

A framework to assist the user with visualizing hierarchical or network data from any application background needs to cover the entire process from acquiring and cleaning the data through different stages of algorithmic computation to finally visualizing the results and interacting on them. The layout of the developed framework directly reflects this process, as it is shown in figure 7.

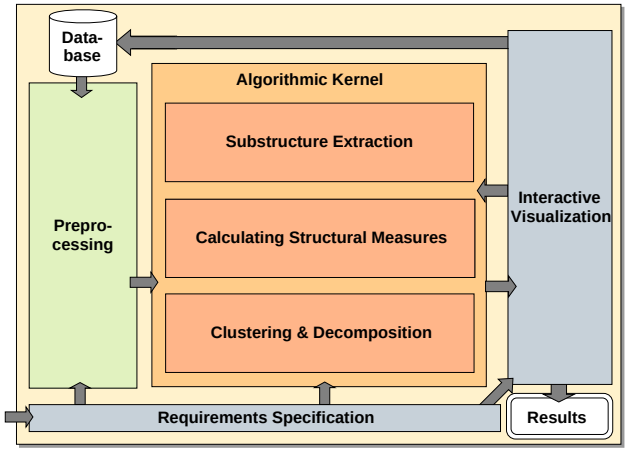

Figure 7: Overview of the graph visualization framework.

For its modular design, the shown components of the framework can be extended with different visualization techniques and algorithmic modules to realize their functioning according to the actual domain of application. In detail, the components provide the following functionality:

Requirements Specification: During this initial phase, the user can roughly parametrize the upcoming algorithmic computations, e.g. by setting upper runtime bounds, choosing preferred visualization techniques or manually weighing the different layout constraints available.

Preprocessing: This step can be used for cleansing and filtering the data, i.e. deleting dangling edges or normalizing node/edge attributes.

Algorithmic Kernel: This component does the actual work of calculating additional data, which is one of the

crucial features an integrated framework for graph visualization must fulfill: extracting/classifying substructures as it is needed i.e. for the TopoLayout technique [3], computing measures like density or diameter of a graph that can be used to automatically determine fitting visualization techniques, clustering/decomposition of the graph i.e. to prepare for later display as a Graph Sketch [1].

Interactive Visualization: Besides laying out and rendering the desired graphical representation, this component can also be used to interactively extract a refined data set (selection/filtering) or to trigger additional calculations, e.g. of measures of interest.

For each of these computational steps, user defined modules can be plugged into the framework and executed in the given order. Of particular interest for the user interaction are the two arrows pointing back and forth between the algorithmic and the visualization component. They tightly couple non-visual procedures with various visual representations at different levels of granularity. This integration enables us, to apply the Shneiderman's *information seeking mantra* to both, non-visual and visual methods allowing a smooth and flexible transition between both – just as the task at hand requires it (see figure 8).

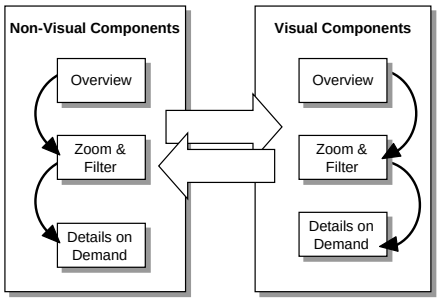

Figure 8: How the framework splits the mantra in techniques on data and on representation level.

How to effectively combine these two views depends on different aspects, first of all on the interests of the user, but also on the size of the given structure and the computed features. A likely example would be, that a user wants to browse a large network via its cluster hierarchy generated by a graph clustering algorithm. If the dendrogram of the cluster hierarchy does not exceed the restrictions of the available display space, it will be possible to show the whole dendrogram at once. But, if the display space is not enough, the user has to start her exploration with an overview image that presents only the first couple of hierarchy levels of the dendrogram but allows further refinement by unfolding subtrees of interest. This shows that a certain non-visual processing step does not automatically lead to its visual counterpart. Instead it can result in different visual outcomes, which leaves the transition from the visual to the non-visual components (as shown in figure 8) influenced by environmental constraints (e.g. screen resolution) or the user's will.

A second example would be, when the user discovers a heavy accumulation of nodes in a certain area of the visualization that she suspects to be a complete subgraph. To prove this, the user could either run an appropriate clustering or decomposition method that would isolate the dense subgraph into a separate cluster if it thus proves to be one. Or, she could use a graph matching approach to search for the largest complete configuration within the subgraph in question. This demonstrates the very same flexible nature of the proposed twofold approach in the other direction. Depending on the user's will or the available computation time, various non-visual components are accessible from visual ones.

Thus, a framework allowing a flexible combination of different computational and visual methods has a great power in supporting the complex visualization and exploration process. Effective exploration utilizes a tight coupling between the different stages as well as between visual and non-visual methods. This allows a natural, iterative exploration process that pursues its goal by stepping back and forth through the available techniques as it is needed. Exactly this is the aim of the new topic *visual analytics* [27].

We successfully applied our framework to a variety of data sets. This included a web link graph of our institute internet pages (51497 nodes, 425247 edges), a citation network (509 nodes, 1551 edges), peer-to-peer-networks with a few hundred nodes and genomic data sets. In [21], we demonstrated the usefulness of our framework design and presented interesting insights for the medium sized Edinburgh Associative Thesaurus data set.

5 Conclusion

In this paper we present a generalized view of graph visualization. This includes a taxonomy scheme for general graph representations, an introductory discussion of constraints that need to be considered in order to gain a meaningful and useful visual graph representation, and a proposal for a visualization and exploration framework for graphs.

The presented ideas point towards current and future challenges to further investigate and understand the numerous interrelated aspects in depicting and exploring graphs. In detail, the next steps of our work will be to identify suitable constraints for hypergraph visualization and to include them into our implementation of the proposed framework.

References

[1] J. Abello, I. Fionocchi, and J. Korn. Graph Sketches. In *Proc. of InfoVis'01*, pages 67–70, 2001.

[2] J. Abello and J. Korn. MGV: A System for Visualizing Massive Multidigraphs. *IEEE Trans. on Visualization and Computer Graphics*, 8(1):21–38, 2002.

[3] D. Archambault, T. Munzner, and D. Auber. Interactive Poster: TopoLayout - Graph Layout by Topological Features. In *Proc. of InfoVis'05*, 2005.

[4] M. Balzer and O. Deussen. Vornoi Treemaps. In *Proc. of InfoVis'05*, pages 49–56, 2005.

[5] T. Barlow and P. Neville. A Comparison of 2-D Visualizations of Hierarchies. In *Proc. of InfoVis'01*, pages 131–138, 2001.

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

[7] T. Dwyer. *Two-and-a-Half-Dimensional Visualisation of Relational Networks*. PhD thesis, 2004.

[8] T. Dwyer and Y. Koren. Dig-CoLa: Directed Graph Layout through Constrained Energy Minimization. In *Proc. of InfoVis'05*, pages 65–72, 2005.

[9] P. Eades, W. Lai, K. Misue, and K. Sugiyama. Preserving the mental map of a diagram. In *Proc. of Compugraphics*, pages 24–33, 1991.

[10] S. Even, A. Pnueli, and A. Lempel. Permutation Graphs and Transitive Graphs. *Journal of the ACM*, 19(3):400–410, 1972.

[11] J.-D. Fekete, D. Wang, N. Dang, A. Aris, and C. Plaisant. Interactive Poster: Overlaying Graph Links on Treemaps. In *Proc. of InfoVis'03*, 2003.

[12] M. Ghoniem, J.-D. Fekete, and P. Castagliola. On the Readability of Graphs Using Node-Link and Matrix-Based Representations. *Information Visualization*, 4:114–135, 2005.

[13] D. Keim, W. Müller, and H. Schumann. Information Visualization and Visual Data Mining. *State of the Art Report, Eurographics 2002*, 2002.

[14] E. Kleiberg, H. van de Wetering, and J. J. van Wijk. Botanical Visualization of Huge Trees. In *Proc. of InfoVis'01*, pages 87–94, 2001.

[15] M. Kreuseler and H. Schumann. A flexible approach for visual data mining. *IEEE Trans. on Visualization and Computer Graphics*, 8(1):39–51, 2002.

[16] J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for viewing large hierarchies. In *ACM Proc. of CHI'95*, pages 401–408, 1995.

[17] C. G. Lekkerkerker and J. C. Boland. Representation of a Finite Graph by a Set of Intervals on the Real Line. *Fundamenta Mathematicae*, 51:45–64, 1962.

[18] P. Neumann, S. Schlechtweg, and S. Carpendale. ArcTrees: Visualizing Relations in Hierarchical Data. In *Proc. of Eurovis'05*, pages 53–61, 2005.

[19] B. Otjacqes and F. Feltz. Representation of Graphs on a Matrix Layout. In *Proc. of IV'05*, pages 339–344, 2005.

[20] G. Sander. Graph Layout through the VCG Tool. In *Proc. of Graph Drawing '94*, pages 194–205, 1995.

[21] H.-J. Schulz, T. Nocke, and H. Schumann. A Framework for Visual Data Mining of Structures. In *Proc. of 29th Australasian Computer Science Conference*, pages 157–166, 2006.

[22] H. Schumann and W. Müller. Informationsvisualisierung: Methoden und Perspektiven. *it - Information Technology*, 3:135–141, 2004.

[23] B. Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualization. Technical report, 1996.

[24] B. Shneiderman and M. Wattenberg. Ordered Treemap Layouts. In *Proc. of InfoVis'01*, pages 73–78, 2001.

[25] J. Stasko, R. Catrambone, M. Guzdial, and K. McDonald. An Evaluation of Space-Filling Information Visualizations for Depicting Hierarchical Structures. *Journal of Human-Computer Studies*, 53:663–694, 2000.

[26] R. Tamassia and I. G. Tollis. Tessellation Representations of Planar Graphs. In *Proc. of 27th Allerton Conference on Communication, Control and Computing*, pages 48–57, 1989.

[27] J. Thomas. *Visual Analytics: a Grand Challenge in Science*. Keynote Talk, InfoVis'05, 2005.

[28] F. van Ham and J. J. van Wijk. Beamtrees: Compact Visualization of Large Hierarchies. In *Proc. of InfoVis'02*, pages 93–100, 2002.

[29] J. Q. Walker. A node-positioning algorithm for general trees. *Software – Practice and Experience*, 20(7):685–705, 1990.

[30] S. Zhao, M. J. McGuffin, and M. H. Chignell. Elastic Hierarchies: Combining Treemaps and Node-Link-Diagrams. In *Proc. of InfoVis'05*, pages 57–64, 2005.