Graph Visualization and Navigation in Information Visualization: A Survey

Ivan Herman, *Member*, *IEEE Computer Society*, Guy Melançon, and M. Scott Marshall

Abstract—This is a survey on graph visualization and navigation techniques, as used in information visualization. Graphs appear in numerous applications such as web browsing, state-transition diagrams, and data structures. The ability to visualize and to navigate in these potentially large, abstract graphs is often a crucial part of an application. Information visualization has specific requirements, which means that this survey approaches the results of traditional graph drawing from a different perspective.

Index Terms—Information visualization, graph visualization, graph drawing, navigation, focus+context, fish-eye, clustering.

1 Introduction

Although the visualization of graphs is the subject of this survey, it is *not* about graph drawing in general. Excellent bibliographic surveys [4], [34], books [5], or even on-line tutorials [26] exist for graph drawing. Instead, the handling of graphs is considered with respect to information visualization.

Information visualization has become a large field and "subfields" are beginning to emerge (see, for example, Card et al. [16] for a recent collection of papers from the last decade). A simple way to determine the applicability of graph visualization is to consider the following question: *Is there an inherent relation among the data elements to be visualized*? If the answer to the question is "no," then data elements are "unstructured" and the goal of the information visualization system might be to help discover relations among data through visual means. If, however, the answer to the question is "yes," then the data can be represented by the nodes of a graph, with the edges representing the relations.

Information visualization research dealing with unstructured data has a distinct flavor. However, such research is *not* the subject of this survey. Instead, our discussion focuses on representations of structured data, i.e., *where graphs are the fundamental structural representation of the data*. Information visualization has specific requirements, which means that we will approach the results of traditional graph drawing from a different perspective than other surveys.

1.1 Typical Application Areas

Graph visualization has many areas of application. Most people have encountered a file hierarchy on a computer system. A file hierarchy can be represented as a tree (a special type of graph). It is often necessary to navigate through the file hierarchy in order to find a particular file. Anyone who has done this has probably experienced a few of the problems involved in graph visualization: "Where am I?" "Where is the file that I'm looking for?" Other familiar types of graphs include the hierarchy illustrated in an organizational chart and taxonomies that portray the relations between species. Web site maps are another application of graphs, as well as browsing history. In biology and chemistry, graphs are applied to evolutionary trees, phylogenetic trees, molecular maps, genetic maps, biochemical pathways, and protein functions. Other areas of application include object-oriented systems (class browsers), data structures (compiler data structures in particular), real-time systems (state-transition diagrams, Petri nets), data flow diagrams, subroutine-call graphs, entity relationship diagrams (e.g., UML and database structures), semantic networks and knowledge-representation diagrams, project management (PERT diagrams), logic programming (SLD-trees), VLSI (circuit schematics), virtual reality (scene graphs), and document management systems. Note that the information isn't always guaranteed to be in a purely hierarchical format—this necessitates techniques which can deal with more general graphs than trees.

1.2 Key Issues in Graph Visualization

The size of the graph to view is a key issue in graph visualization. Large graphs pose several difficult problems. If the number of elements is large, it can compromise performance or even reach the limits of the viewing platform. Even if it is possible to layout and display all the elements, the issue of viewability or usability arises because it will become impossible to discern between nodes and edges (see Fig. 1, although this tree is by no means a very complex one). In fact, usability becomes an issue even before the problem of discernability is reached. It is well-known that comprehension and detailed analysis of data in graph structures is easiest when the size of the displayed graph is small. In general, displaying an entire large graph may give an indication of the overall structure or a location within it, but makes it difficult to comprehend. These issues form the context for most of this survey.

Other than the usual reference to information overload and the occasional reference to the gestalt principle, papers in information visualization rarely apply cognitive science

• *The authors are with the Centre for Mathematics and Computer Sciences, CWI, Kruislaan 413, PO Box 94079, 1090 GB Amsterdam, The Netherlands. E-mail: {I.Herman, G.Melancon, M.S. Marshall}@cwi.nl.*

For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 111225.

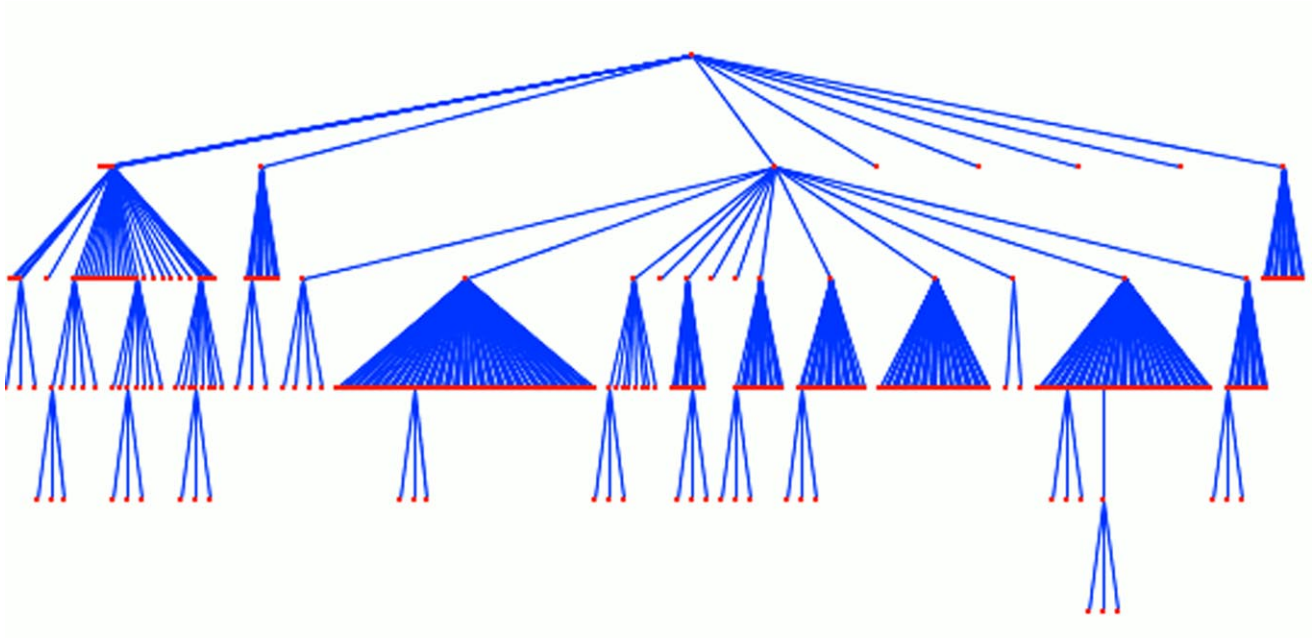


Fig. 1. A tree layout for a moderately large graph.

and human factors. This is for no lack of trying; very few of the findings in cognitive science have practical applications at this time and very few usability studies have been done. Cognitive aspects are undoubtedly a subject for future research. For this reason, an objective evaluation of the merits of a given approach is difficult. The reader has to bear this limitation in mind when various techniques are presented.[1]

The rest of this survey is organized as follows: In Section 2, we try to give an impression of graph layout issues and limitations with regard to scalability. Then, we discuss several approaches to navigation of large graphs (Section 3), followed by methods of reducing visual complexity through reorganization of the data (Section 4). Afterwards, we discuss a few application systems that implement many of the techniques described in this survey (Section 5). To help the reader pursue further research and development, we have listed the various sources of information that we found particularly important for graph visualization (Section 6) and provided an extensive list of references.

2 Graph Layout

This section looks at the current results in graph drawing and layout algorithms, but from the point of view of graph visualization in information visualization. As we will see, this point of view differs, in many respects, from the traditional view of the Graph Drawing community. We will give an account of the available results and discuss their relevance for graph visualization, although, in general, we will not go too far into the technical details. For those desiring more information, we recommend the excellent book from Battista et al. [5] as one of the best starting points.

2.1 Background of Graph Drawing

The Graph Drawing community[2] grew around the yearly Symposia on Graph Drawing (GD 'XX conferences), which were initiated in 1992 in Rome. Springer-Verlag publishes the proceedings of the conference in the LNCS series, which contains new layout algorithms, theoretical results on their efficiency or limitations, and systems demonstrations. The recent electronic *Journal of Graph Algorithms and Applications* is dedicated to papers concerned with design and analysis of graph algorithms, as well as with experiences and applications.

The basic graph drawing problem can be put simply: Given a set of nodes with a set of edges (relations), calculate the position of the nodes and the curve to be drawn for each edge. Of course, this problem has always existed for the simple reason that a graph is often defined by its drawing. Indeed, Euler himself relied on a drawing to solve the "Königsberger Brückenproblem" in his 1736 paper (see the recent book of Jungnickel [74]). The annotated bibliography by Battista et al. [4] gathers hundreds of papers studying what a *good* drawing of a graph is. That is, where the problem becomes more intricate: It requires the definition of properties and a classification of layouts according to the type of graphs to which they can be applied. For example, a familiar property is *planarity*—whether it is possible to draw a graph on the plane with no edge crossing. Layout algorithms may be categorized with respect to the type of layout they generate. For example, grid layouts position nodes of a graph at points with integer coordinates. Other categories of layouts are defined by the methodology on which they are based. For example, nondeterministic

1. Ware's new book [123] may become an important source of information in this area.

2. http://www.cs.brown.edu/people/rt/gd.html.

approaches form a category that uses algorithms such as force-directed models or simulated annealing. Each class of graphs and layouts thus generates its own set of problems. Planarity, for example, raises problems such as:

- Planarity tests for graphs: Is it possible to draw a graph without edge-crossings?
- Planar layout algorithms according to various constraints: Given that a graph is planar, find a layout satisfying a group of constraints.

Many constraints in use are also expressed in terms of *aesthetic rules* imposed on the final layout. Nodes and edges must be evenly distributed, edges should all have the same length, edges must be straight lines, isomorphic substructures should be displayed in the same manner, edge-crossings should be kept to a minimum, etc.[3] Trees have received the most attention in the literature. Consequently, additional aesthetics rules have also been formulated for them. For example, nodes with equal depth should be placed on a same horizontal line, distance between sibling nodes is usually fixed, etc. See again the book of Battista et al. [5] for further examples.

The Reingold and Tilford algorithm for trees [103], [121] (see Fig. 1) is a good example of a layout algorithm achieving these aesthetic goals. Isomorphic subtrees are laid out in exactly the same way and distance between nodes is a parameter of the algorithm. On the other hand, the more straightforward and naive algorithm for displaying a tree, consisting of distributing the available horizontal space to subtrees according to their number of leaves, actually fails to achieve some of the aesthetic rules listed above.

Although the adjective "aesthetic" is used, some rules were originally motivated by more practical issues. For instance, minimization of the full graph area might be an important criterion in applications. Some of the rules clearly apply to a certain category of graphs or layouts only, others have a more "absolute" character. Furthermore, each of the rules defines an associated optimization problem, used in a number of nondeterministic layout algorithms.

There has been some work lately which questions the absolute character of those rules, however. Usability studies were conducted in order to evaluate the relevance of these aesthetics for the end-user. Purchase [100] demonstrates that "reducing the crossings is by far the most important aesthetic, while minimizing the number of bends and maximizing symmetry have a lesser effect." Her work concludes by prioritizing these aesthetics; see also Purchase et al. [101], [102] for more details. Other authors [10], [29], [86] report differences in the perception of a graph depending on its layout. Unfortunately, usability studies necessitate a great effort, both to realize the experimentation itself and to analyze its results properly, but we regard this line of work as essential for information visualization. Usability studies have recently gained credibility in the graph visualization community as well, recognizing their contribution to help focus on important issues in the area.

3. Actually, some aesthetics are quite arbitrary and are not seen as absolute rules any more [100], [101]. Ware's book [123] is also an interesting source of information for this topic.

A wide variety of tasks related to graph drawing have been studied: layering a graph, turning it into an acyclic directed graph, planarization of a graph, minimizing the area occupied by a layout, minimizing the number of bends in edges, etc. Unfortunately, many of the associated algorithms are too complex to be practical for applications. On the positive side, this has motivated the development of effective heuristics to overcome the complexity of some of these problems [5], [34].

In graph visualization, a major problem that needs to be addressed is the *size* of the graph. Few systems can claim to deal effectively with thousands of nodes, although graphs with this order of magnitude appear in a wide variety of applications. NicheWorks [126], GVF [64], and H3Viewer [94] are among the few systems that claim to handle data sets with thousands of elements. The size of a graph can make a normally good layout algorithm completely unusable. In fact, a layout algorithm may produce good layouts for graphs of several hundred nodes, but this does not guarantee that it will scale up to several thousand nodes. For example, Fig. 1 illustrates a tree with a few hundred nodes laid out using the classical Reingold and Tilford algorithm. The high density of the layout comes as no surprise and changing particular parameters of the algorithm will not improve the picture for the graph. Other 2D layout techniques could be used, but most layout algorithms suffer from the same problem. Because the layout is so dense, interaction with the graph becomes difficult. Occlusions in the picture make it impossible to navigate and query about particular nodes. The use of 3D or of non-Euclidean geometry have also been proposed to alleviate these problems. Sections 2.4 and 2.5 provide more details about these techniques. However, beyond a certain limit, no algorithm will guarantee a proper layout of large graphs. There is simply not enough space on the screen. In fact, from a cognitive perspective, it does not even make sense to display a very large amount of data. Consequently, a first step in the visualization process is often to reduce the size of the graph to display. Classical layout algorithms remain usable tools for visualization, but only when combined with these techniques.

Other properties of a layout algorithm can be critical when navigating through a graph. The concept of *predictability* has been identified as an important and necessary aspect of layout algorithms [61], [99]. What is meant by predictability is that two different runs of the algorithm, involving the same or similar graphs should not lead to radically different visual representations. This property is also referred to in the literature as "preserving the mental map" of the user [90]. Predictability is often ignored during analysis of classical layout algorithms, which are usually used to produce a static view of a graph.

Another important issue is *time complexity*. Any visualization system needs to provide near real-time interaction, where updates must be done in very short time intervals in order to escape the notice of the user. Having an accurate estimate of the time complexity of an algorithm can be of great help for the implementation of large systems when planning which algorithm to apply.

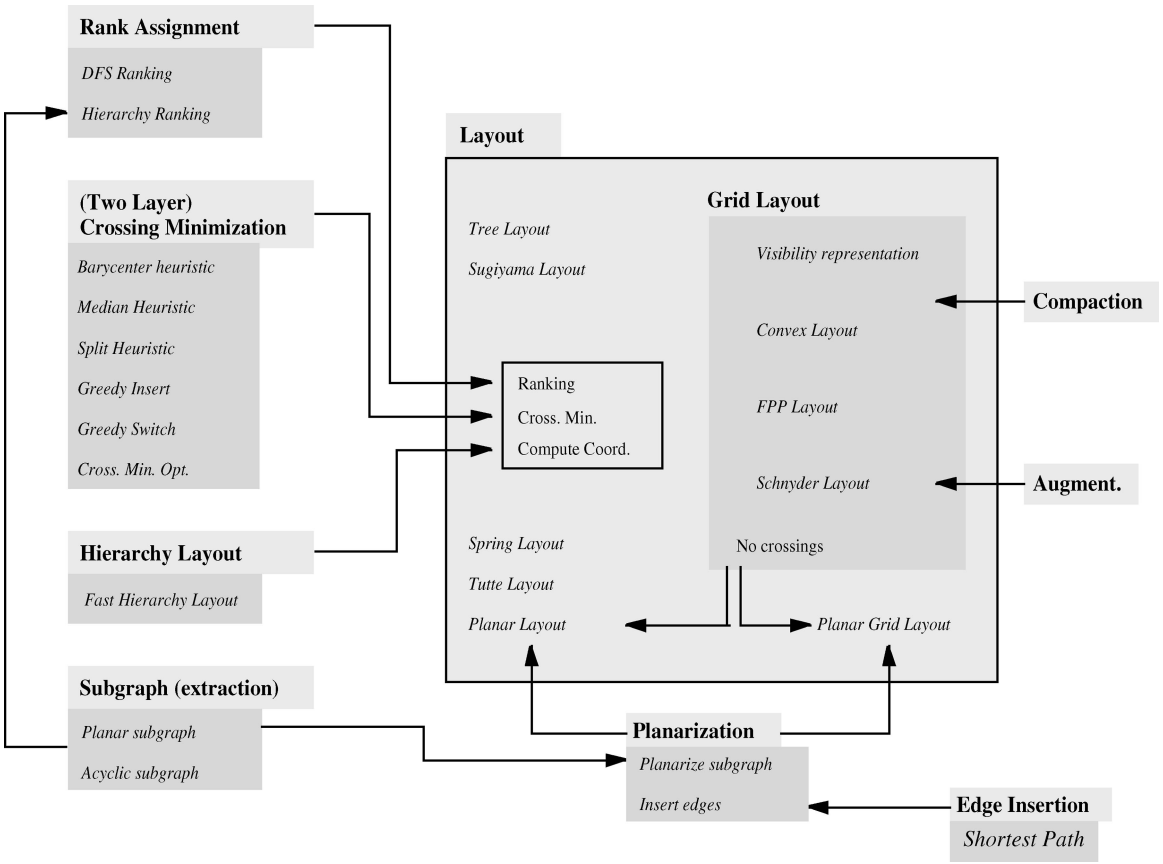


Fig. 2. Overview of graph layout algorithms. (Reproduced from Mutzel et al. [96], courtesy of T. Mutzel, Max-Planck-Institut, Saarbrücken, Germany.)

2.2 Traditional Layout—An Overview

We will briefly review existing layout techniques in graph drawing, keeping the issues of predictability and time complexity in mind. Fig. 2 gives a classification of existing layout techniques. This classification is the work of Mutzel et al. [96]. Most of the algorithms are described in the book by Battista et al. [5]. We will focus on the *Layout* box containing a list of possible layout types.

A classical *Tree Layout* will position children nodes "below" their common ancestor. The algorithm by Reingold and Tilford [103], [121] is probably the best known layout technique in the tree layout category (see Fig. 1). It can be adapted to produce top-down, as well as left-to-right tree layout, and can also be set to output grid-like positioning.

H-tree layouts are also classical representations for binary trees [113] which only perform well on balanced trees. Eades [35] suggests a variation of the algorithm that behaves well in general (see Fig. 3). The radial positioning by Eades [35] places nodes on concentric circles according to their depth in the tree (see Fig. 4). A subtree is then laid out over a sector of the circle and the algorithm ensures that two adjacent sectors do not overlap (although this condition can be ignored to obtain relatively good drawings on average [63], [126]). The cone tree [20], [106] algorithm can be used to obtain a "balloon view" of the tree by projecting it onto the plane [20], [71], where sibling subtrees are included in circles attached to the father node. It is also possible to compute the node position directly, without using cone trees[87] (see Fig. 5; Section 2.4 describes cone trees in more detail).

The Reingold and Tilford algorithm produces a more classical drawing in the sense that the drawing clearly reflects the intrinsic hierarchy of the data. The radial and H-tree positioning are different in this respect because it is less clear where the root of the tree is and, thus, one might explore the graph in a less hierarchical fashion. The Reingold and Tilford, H-tree, radial, and balloon layouts are all predictable. Tree layout problems usually have the lowest complexity, which is linear in the number of nodes. As we can see, although the *Tree Layout* box occupies only a small area of Fig. 2, it contains a variety of layouts. Chapter 3.1 of the book by Battista et al. [5] is a good starting point for a further overview of these tree layout techniques. Two tree layout algorithms, which are not part of the "traditional" arsenal, are also worth mentioning here: tree-maps [72] (see Fig. 6) and onion graphs [115], which represent trees by sequences of nested boxes. It is important to note that, in tree-maps, the size of the individual rectangles is significant. For example, if the tree represents a file system hierarchy, this size may be proportional to the size of the respective file. This is why tree-maps enjoy popularity in information visualization in spite of the fact that it is difficult to perceive the structure in this representation.[4] An attempt to overcome this problem has

4. The value of the tree-map is demonstrated in an interactive java applet at http://smartmoney.com/marketmap/.

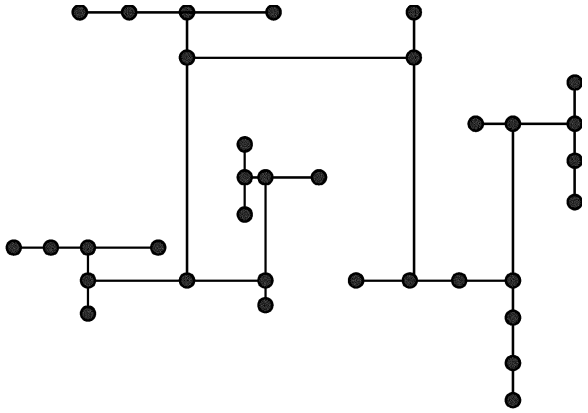


Fig. 3. H-tree layout.

been recently presented by Wijk and Wetering [125] in the form of cushion tree-maps.

A separate box at the bottom of Fig. 2 is devoted to *Planarity*. This is a critical issue in graph drawing because the planarity of a graph may be an important constraint imposed by practical applications (such as graphs representing printed circuit boards). The complexity for testing planarity for undirected graphs can be linear [67]. (See Chapter 3.3 in Battista et al. [5]. See also Mehlhorn and Mutzel [88] for a discussion on implementation issues.) However, many applications impose the additional requirement that edges are all in the same direction (planar drawings often make use of edges going around some nodes to avoid crossings). This condition, called *upward planarity*, turns the original problem into an NP problem. (See Garg and Tamassia [54]. See also Chapter 6 in Battista et al. [5]). In information visualization applications, it only makes sense to check for planarity when dealing with a small and sparse graph [3], [30], such as a subgraph obtained by clustering a larger graph (see Section 4.). In general, we can safely say that planarity is not a central issue in information visualization.

The *Sugiyama Layout* box included in Fig. 2 is named after the seminal work by Sugiyama et al. on layout for

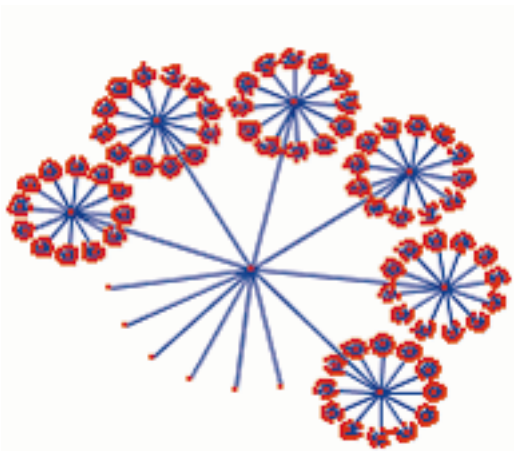


Fig. 5. Balloon view.

general directed graphs [117]. The basic approach to laying out a directed graph is to first decide on a *layering* of its nodes; that is, assign a layer number to each node and place nodes of a given layer in a certain order. Several layering techniques exist, the majority of which rely on the extraction of an acyclic subgraph. In this process, a subgraph containing all nodes of the original graph is extracted in such a way that, when nodes have been placed in their respective layers, edges will all point in the same direction (usually downward). Another solution is not to extract a subgraph, but to turn the original graph into an acyclic one by reversing the direction of a subset of the edges.

Once the nodes have been assigned to layers, one must position the nodes within the same layer following an imposed order. A major effort has been invested in edge-crossing minimization [5], [34] since the crossing of edges has been recognized as an obstacle to the readability of graphs [100], [101]. This is usually done by minimizing the number of edge-crossings between two consecutive layers. This minimization step is at the core of complexity for the whole algorithm. Note that these strategies do not address the problem of minimizing the number of crossings in the

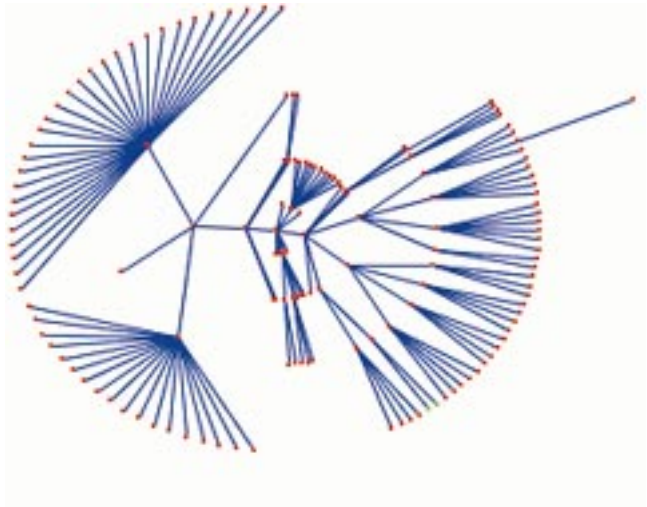


Fig. 4. Radial view.

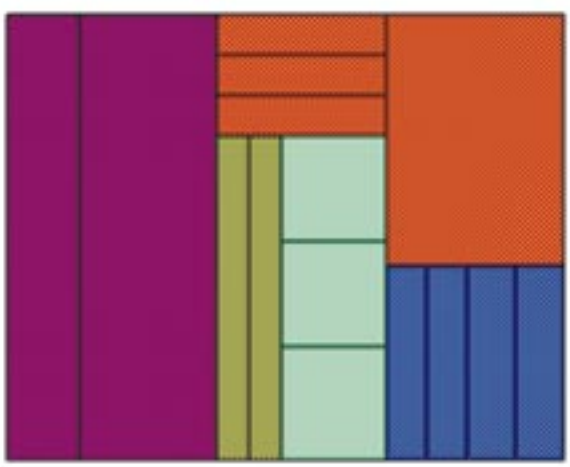


Fig. 6. Tree-map: rectangles with color belong to the same level of the (tree) hierarchy. (Adapted from Johnson and Schneiderman [72]).

whole graph: Even with the restriction of looking at consecutive layers only, minimization of edge-crossings is difficult and complex. In fact, Garey and Johnson proved the problem to be NP-hard [53] and Eades and Whitesides proved the corresponding decision problem to be NP-complete [36].

The complexity of a proper minimization has motivated the development of various heuristics for computing a good order for the nodes on a layer. Tutte [119] was the first to propose a heuristic: Starting from an order on the top and bottom layers, the coordinates of a node are defined to be the barycenter of those of its neighbors. This corresponds to the intuitive idea that a node should be kept "close" to its neighbors. The solution is then obtained by solving a system of linear equations. One variation to this scheme is to compute barycentric coordinates by performing a layer-by-layer descent in the graph. More generally, the four boxes on the left of the figure correspond to various preprocessing possibilities for the algorithm in the *Sugiyama Layout* category. New improvements and perspectives on the problem were published recently [73], [79], which include a detailed report on existing techniques [80] and a comparison of existing heuristics [81].

The critical element of the general scheme for directed graphs is its high complexity, although it might be kept within reasonable bounds if the size of the graph—or, should we say, subgraph—to be drawn is kept small. The ranking process in itself has a low cost. Indeed, a breadth first search of the graph returns an acyclic subgraph that can be used for layering. However, the choice of this subgraph can determine the quality of the final layout. We will return to that issue later. It is also not clear whether any algorithm in this class will be predictable. Some approaches can certainly be made predictable, but then the price to pay will be a greater complexity due to the loss in flexibility in reordering the nodes on a layer. Battista et al. give a detailed account of edge-crossing minimization in Chapter 9 of their book [5].

The *Spring Layout* box stands for all nondeterministic layout techniques, also called *Force-Directed Methods*. Eades [33] was the first to propose this approach in graph drawing, modeling nodes and edges of a graph as physical bodies tied with springs. Using Hooke's law describing forces between the bodies, he was able to produce layouts for (undirected) graphs. Since then, his method was revisited and improved [28], [47], [49], [75]. Mathematically, the methods are based on an optimization problem. Different physical models lead to algorithms of different complexities and they produce layouts of varying quality. Spring layouts have been used successfully to produce well-balanced layout for graphs. In some cases, their output can even behave well with respect to edge-crossing minimization without any supplementary efforts [47]. Bertault has recently developed a force-directed model preserving edge-crossings, turning it into a more predictable approach [9].

In general, however, force-directed methods can be rather slow. Each iteration involves a visit of all pairs of nodes in the graph and the quality of the layout depends on the number of full iterations: each step improves the positions following the underlying mathematical model. Even one of the best variants [47] is still estimated to work with a complexity of $O(N^3)$, where N is the number of nodes in the graph. Moreover, two different runs of the algorithm on almost identical graphs might produce radically different layouts. In other words, the methods may be highly unpredictable. This makes them less interesting for information visualization since unpredictability can be a major problem for interaction. However, in some cases, the lack of predictability can be compensated for if the graph is small or sparse, by animating changes in the layout to help the user in adapting to the new drawing [69]. For further information on force-directed methods, the reader should refer to the comparison of nondeterministic techniques of Brandenburg et al. [12] or Chapter 10 in the book of Battista et al. [5].

We will not discuss layouts on grids here. We refer to Battista et al. [5] for details on that, as well as for learning more about the additional techniques included in the boxes "Compaction" and "Augmentation" on the right side of Fig. 2. None of these techniques play a central role in graph visualization.

The classification of algorithms in Fig. 2 assumes that layout is determined only by the nodes and edges, without additional constraints. However, some work has been done with applications where the nodes of the graph have preassigned positions in the plane, such as geographical positions. The challenge is then to find a way to draw edges, for example, by using polylines or spline curves [6], [13], [97].

2.3 Spanning Trees

A general problem with the majority of the available techniques is that they are only applicable for relatively small graphs.[5] The "traditional" concerns of Graph Drawing become much less relevant in graph visualization, which typically deals with relatively large graphs. In general, it makes no sense to test a graph of several hundred nodes for planarity or to try to minimize edge-crossings. Often, the most obvious and practical solution is simply to layout a spanning tree for the graph. As we have already seen, tree layout algorithms [20], [35], [103], [121] have the lowest complexity and are simpler to implement. The problem is then transformed into one of finding a spanning tree. That option involves laying out a graph based on the positioning of a tree containing all nodes of the graph which had been previously extracted from the graph. Additional edges are then added to the tree. The literature in graph theory proposes a long list of algorithms to compute spanning trees for graphs, both for the directed and undirected cases (see, for example, Jungnickel [74]). Incidentally, using a spanning tree to layout a graph can also be a solution to gain predictability of the layout. Although spanning trees are obviously not the only layout approach in graph visualization, they certainly do and will play an important role.

Extracting a spanning tree with no particular property can be done easily. One approach is to visit the nodes of the

5. This is clearly shown by the size of the graphs submitted each year to the Graph Drawing Contest, although bigger graphs—and, also, graphs coming from real-world life situations applications—have also been included in recent years.

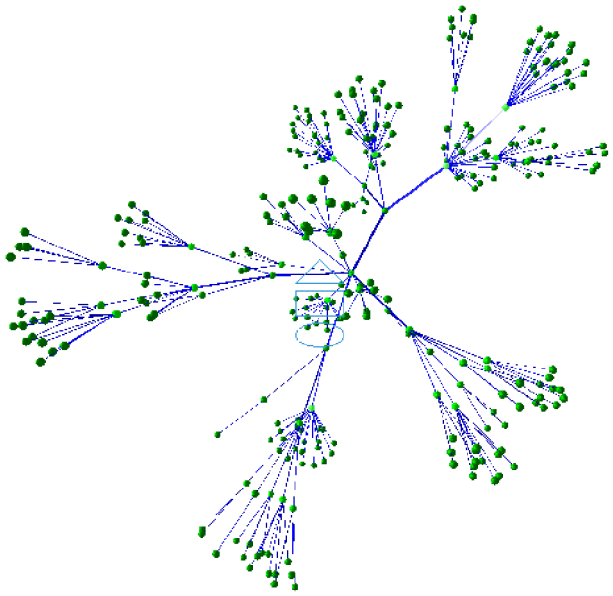


Fig. 7. 3D version of a radial algorithm. (Courtesy of S. Benford, University of Nottingham, U.K.)

graph through a breadth first search and collect edges to form a tree. The search can start from a node that is more likely to "act" as the root of the extracted tree. A node whose distance to all other nodes is minimal is a good candidate [11]. More sophisticated algorithms have been designed to satisfy various optimization goals. If a weight function exists for the graph, algorithms exist to compute spanning trees minimizing (or maximizing) the total weight of the tree. One solution is to iteratively build a tree by adding edges adjacent to the set of already selected nodes, each time selecting an edge with minimal (maximal) weight. Different choices for the weight function will yield different solutions and will also affect the complexity of the extracting process (see, for example, Chapters 4 and 5 of Jungnickel [74]). The complexity of this task varies according to the variant used. The naive solution has a complexity of $O(N^2)$; better solutions exist which bring the complexity down to $O(N \log N)$ or to $O(E \log N)$ (where N and E denote the number of nodes and edges of the graph, respectively).

A weight function can be used to extract different spanning trees and, consequently, to obtain different possible layouts for the same graph (although the implementor must be aware of the fact that a spanning tree realizing an optimization goal for a given weight function does not necessarily produce a good view of the graph). Use of weight functions can also be applied to directed acyclic graphs to avoid going through the task of edge-crossing minimization. For large and dense acyclic directed graphs, the use of layers as a weight function (the weight of a node or edge is its layer number) has proven to give good results (see, for instance, Herman et al. [63]).

2.4 3D Layout

One popular technique is to display graphs in 3D instead of 2D. The hope is that the extra dimension will give, literally, more "space" and that this will ease the problem of displaying large structures. Furthermore, the user can navigate to find a view without occlusions. The simplest approach is to generalize classical 2D layout algorithms for 3D. Fig. 7, for example, shows a 3D version of a radial tree algorithm, while Fig. 8 is a generalization [104] of the two-dimensional approach using nested boxes [115]. Most force-directed methods are also described in dimension independent terms, which allows them to be generalized to 3D (such as the approaches based on simulated annealing by Davidson and Harel [28] and, also, from Cruz and Twarog [27]). The reader may find further examples in the overview by Young [128] or in the new book by Ware [123].

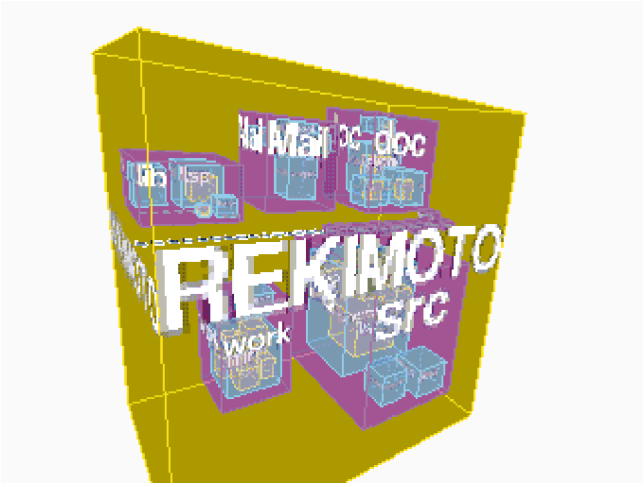


Fig. 8. Information Cube. (Courtesy of J. Rekimoto, Sony Computer Science Laboratory, Inc., Japan [104].)

In spite of their apparent simplicity, Figs. 7 and 8 show that displaying graphs in 3D can also introduce new problems. Objects in 3D can occlude one another and it is also difficult to choose the best "view" in space [38]. As a consequence, virtually all 3D displays of graphs include additional visual cues, like transparency, depth queuing, etc. They also allow the user to interactively change the view by "moving around" in space. But, the ability to change perspective adds another difficulty. Common practices, such as the minimization of edge-crossings, are less rewarding if the user can change the perspective and see edge-crossings from another angle. However, it is the job of the application to provide the best possible view of the information in the perspective initially provided to the user, so aesthetics cannot be dismissed.

The cone tree , [107] (see Fig. 9) is one of the best known 3D graph (in this case, tree) layout techniques in information visualization.[6] In contrast to the previous examples, cone trees have been developed directly for 3D, instead of generalizing another 2D algorithm.

Mathematically, the layout is quite simple. A node is placed at the apex of a cone with its children placed evenly along its base. In the original implementation, each layer has cones of the same height and the cone base diameters for each level are reduced in a progression so that the bottom layer fits into the width of what the authors called

6. The term "cam tree" is also used. Strictly speaking, cam trees are horizontal arrangements, whereas cone trees are vertical. We will not differentiate between them.

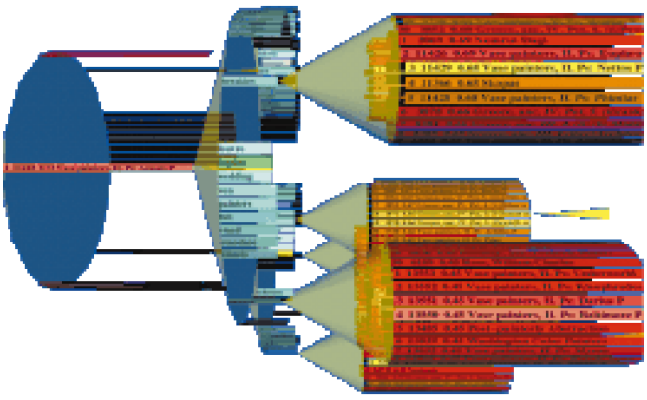


Fig. 9. A cone tree. (Courtesy of M. Hemmje, GMD, Germany [59].)

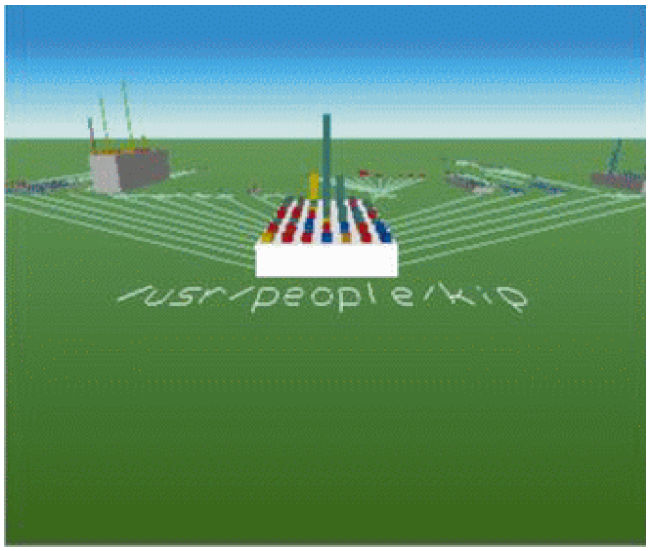


Fig. 10. The SGI File System Navigator.

the "room," i.e., the box containing the full cone tree. The original idea of cone trees has been reimplemented by others [20], [59], [71] with, in some cases, a refined layout algorithm. Carrière and Kazman [20], for example, calculate an approximation of the diameter for each cone base by traversing the tree bottom-up and by taking the number of descendents into account at each step to make better use of the available space. Jeong and Pang [71] replace the cones with discs to reduce occlusion.

The interactive and visual aspects of cone trees are essential to make them usable. Not only are some of the labels at the nodes transparent, but the user can pick any node and rotate the cone tree so that the chosen node is brought to the front. This can either be done automatically, by the system, or as a result of further user interaction. For horizontal cone trees, the effect somewhat resembles stepping through Rolodex cards arranged in multiple levels.

Gaining more "space" is not the only possible advantage of using 3D. Because of general human familiarity with 3D in the physical world, 3D lends itself to the creation of real-world metaphors that should help in perceiving complex structures. One of the earliest widespread applications is the File System Navigator (see Fig. 10), which came with earlier SGI Workstations until version 5 of their operating system. The layout of the graph (a tree representing the user's file space) is a simple planar layout. The 3D aspect consists, on the one hand, of adding blocks on the plane whose sizes are proportional to the file sizes and, on the other hand, of the ability to "fly" over the virtual landscape created by those blocks. This cityscape approach has been implemented in various other systems, see, for example, SDM [24], or, more recently, the system presented by Chen and Carr [22]. More complex 3D metaphors include the Perspective Wall [107], which represents the data as posters on a big wall in virtual space. VizNet [43] and Vitesse [98] both use an idea similar to the perspective wall by mapping objects onto the surface of a sphere with highly related objects placed close to a selected object of interest. The Web Book [15] displays an animated book in 3D with Web page contents, etc. Here again, we refer to the overview of Young [128] for further examples.

In spite of all the technical development in the area, and their undeniably attractive features, 3D graph visualization techniques have significant difficulties. In our view, the main reason lies with the inherent cognitive difficulties of 3D navigation in our current systems. Perceptual and navigational conflicts are caused by the discrepancy of using 2D screens and 2D input devices to interact with a 3D world, combined with missing motion and stereo cues (see the overview of Ware and Franck [122] for how important these cues are). Limited 3D interaction, such as the ability to rotate an object for inspection without getting closer to it, may provide 3D interaction that doesn't cause disorientation. If advanced VR-like systems, such as a Workbench, CAVE, or large tiled displays are used, some of these difficulties may be solved. However, such facilities are not widely available and are still too expensive to serve as a basis for most information visualization applications. When more advanced display and interactive facilities (e.g., haptic displays and interaction, stereo views, etc.) become more widely available, 3D techniques may have a profound effect in graph visualization.

2.5 Hyperbolic Layout

The hyperbolic layout of graphs (mainly trees) is one of the new forms of graph layout which has been developed with graph visualization and interaction in mind. The first papers in this area are from Lamping et al. [82], [83], followed by a series of papers by Munzner [92], [93], [94]. Both developed, for example, Web content viewers based on these techniques. The technique has been since used by other systems, too, see, for example, Robinson [108] or Wilson and Bergeron [127]. Hyperbolic views, which can be implemented in either 2D or 3D, provide a distorted view of a tree (see Fig. 11). It resembles the effect of using fish-eye lenses on traditional tree layouts. This distorted view makes it possible to interact with potentially large trees, making it suitable for real-life applications. We will come back to this

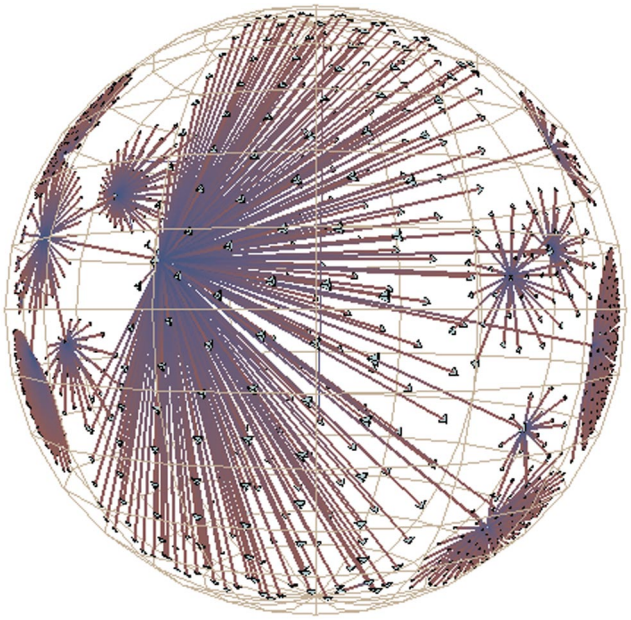


Fig. 11. Hyperbolic view of a tree in 3D. (Courtesy of T. Munzner, Stanford University.)

distortion effect later in this survey (see Section 3.2) when we will focus on navigation rather than layout.

Hyperbolic views represent a radically different direction in layout when compared to the algorithms described so far, due to their different geometrical background. In fact, some of the classical layout algorithms can be reused in a hyperbolic setting, yielding sometimes quite different results, as demonstrated later in this section. Hyperbolic views are also surrounded by a sort of mystery because few people in the information visualization community really understand the mathematics of hyperbolic visualization. Furthermore, it is quite difficult to reproduce the results. Unfortunately, none of the papers are didactic enough to reveal the mystery. We will discuss the main elements of these layout methods further with the hope that the reader will gain a better understanding of the technique.

Hyperbolic geometry is based on an axiomatic system almost identical to the traditional Euclidean axioms with the exception of one, the so-called fifth postulate. Whereas the Euclidean postulate states that if a line does not intersect a point, then there is *only one* line intersecting the point and parallel to the original line (i.e., nonintersecting and coplanar), in hyperbolic geometry there exists *more than one* such parallel line. This alternative set of axioms results in a perfectly consistent form of geometry, albeit different in flavor: The traditional trigonometric equations are no longer valid, the sum of the internal angles of a triangle is no longer 180 degrees, etc.[7] (These differences, by the way, represent significant difficulties for implementors using hyperbolic geometry.)

It is also possible to define a consistent *model* for the hyperbolic plane (or space) within the Euclidean space, thereby making a logical link between the two worlds. A model in this respect means defining a subset of the

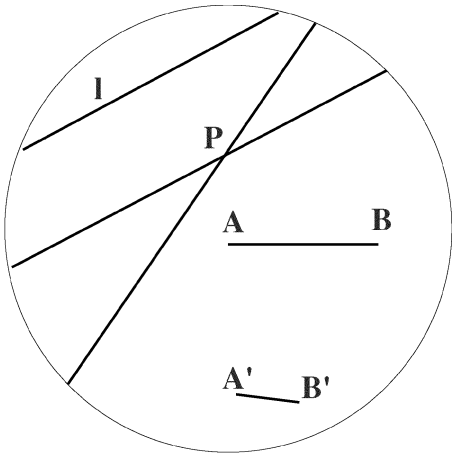


Fig. 12. The Klein model for the hyperbolic plane. The line segment **AB** and **A'B'** have an equal length in the hyperbolic sense.

Euclidean space and the notions of "points," "lines," "intersections," "length" within this subset so that the axioms of hyperbolic geometry would be valid locally. Several different models were developed. The best known are the Klein and the Poincaré models. The Klein model (see Fig. 12) uses an open disc (or sphere for 3D) as a subset, i.e., the hyperbolic plane in this model consists of the points within the perimeters of the disc. Hyperbolic lines are represented by chords of the disc. Intersection is just the Euclidean intersection. The only major difference is the *length* of a line segment. We will not give a detailed definition here. Suffice it to say that this length is defined as a function of the position of the points vis-à-vis the perimeter of the disc: Segments which are congruent in a hyperbolic sense are exponentially smaller in the Euclidean sense when approaching the perimeter. To prove the local validity of *all* the axioms of hyperbolic geometry requires some nontrivial work. The validity of the negation of Euclid's fifth postulate is quite obvious, though, just consider the line l and the point P on the figure. The Poincaré model is quite similar, although hyperbolic lines are represented by arcs which intersect ortogonally the perimeter of the disc.

It is now possible to give a more exact description of what the hyperbolic graph layouts do orthogonally: They perform a layout algorithm in the hyperbolic plane or space and, then, display the results in the familiar Euclidean plane or space *using one of the models of hyperbolic geometry*. That is, what we see is *not* hyperbolic geometry per se, but its representation in Euclidean geometry. The original paper of Lamping et al. used the Poincaré model, whereas Munzner primarily uses the Klein model. In Fig. 11, for example, the Klein model for hyperbolic 3D space is used to display the tree. The distortion effect referred to earlier is the result of the exponential shrinking of congruent line segments closer to the disc perimeter when viewed in the Euclidean space.

The different spatial nature of hyperbolic geometry makes some rather simple layout algorithms suddenly viable. As an example, consider the outline of the following tree placement algorithm (see Fig. 13).[8] The algorithm starts

7. The interested reader might want to refer to Coxeter [25] for further details. Also, look at the papers of Gunn [56] or Hausman et al. [57].

8. This algorithm is essentially the same as the one used in the paper of Lamping and Rao [83].

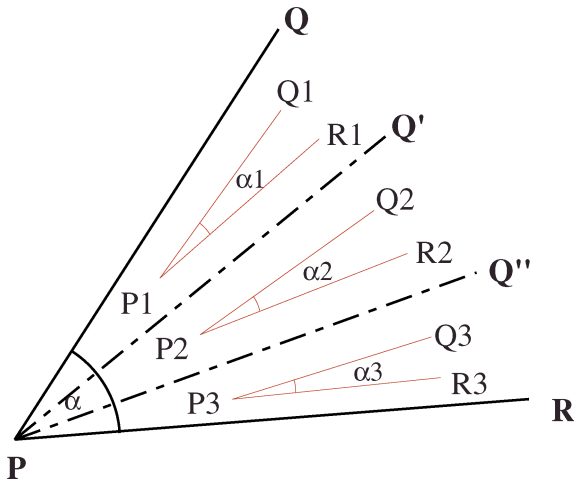


Fig. 13. A simple tree positioning algorithm on the Euclidean plane.

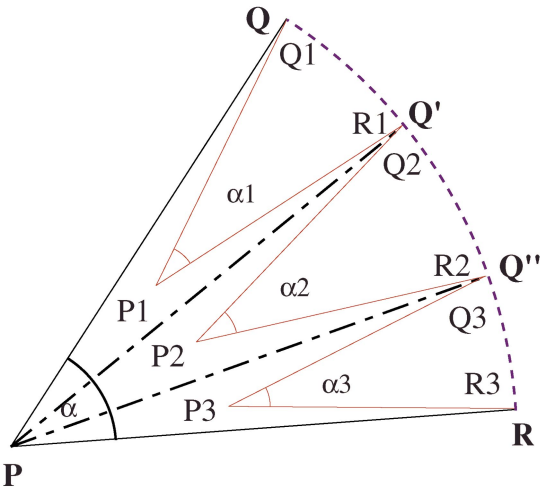


Fig. 14. The same tree positioning algorithm on the hyperbolic plane, using the Klein model to visualize the results.

from the root of the tree, positioning the subtrees recursively in a circular fashion. In each step, the algorithm determines a wedge to place a subtree. The goal is to find wedges in such a way that no crossing would occur between edges of different subtrees. If the point P on the figure refers to a node, and the wedge QPR with angle α is the one assigned to the subtree starting at P, the main step of the algorithm is to define subwedges for the subtrees of P (starting at P_1, P_2, and P_3). The angle α is divided into (for the sake of simplicity, equal) subangles, one for each subtree. The subdivision of the original wedge results in the radii PQ', PQ'', etc. (see the figure). The points P_1, P_2, P_3 are positioned in the middle of these subwedges at some suitable distance from P. The next step is to determine the constraining wedges for these subtrees. This can be done by establishing parallel lines with PQ, PQ', PQ'', starting at the points P_1, P_2, P_3, etc. These lines will determine the new wedges with angles α_1, α_2, α_3, etc., and the recursion step can continue for each of the corresponding subtrees. Obviously, because parallel lines are used, the children's wedges will not overlap.

The algorithm is very naive and would lead to quite unusable figures on the Euclidean plane. Indeed, the wedge angles become very small after a few steps, which shrinks the space available for the next subtree. However, if the same algorithm is used on a hyperbolic plane, the situation is quite different. Fig. 14 shows the same algorithm in the Klein model. The major difference is the way the parallel lines to PQ', PQ'', etc., are calculated: The (hyperbolic) parallel lines are the lines intersecting *on* the perimeter of the disc of our model. The effect will be to "open" the angles α_1, α_2, α_3. To cite Lamping and Rao [83]: "Each child will typically get a wedge that spans about as big an angle as does its parent's wedge." Of course, although visible on the Klein model, this statement has to be substantiated through explicit formulae using the hyperbolic trigonometric calculations, which is quite possible. The result is a perfectly feasible layout algorithm. It should be noted that Munzner uses different layouts. More details on her spherical placement can be found in one of her papers [93], which is actually a generalization of the cone tree algorithm described in Section 2.4. However, here again, the placement algorithm is used in terms of hyperbolic geometry, taking advantage of the "large space" available in hyperbolic space.

3 Navigation and Interaction

Navigation and interaction facilities are essential in information visualization. No layout algorithm alone can overcome the problems raised by the large sizes of the graphs occurring in the visualization applications. Furthermore, the task of revealing the *structure* of the graph calls for innovative approaches.

3.1 Zoom and Pan

Zoom and pan are traditional tools in visualization. They are quite indispensable when large graph structures are explored. Zoom is particularly well-suited for graphs because the graphics used to display them are usually fairly simple (lines and simple geometric forms). This means that zoom can, in most cases, be performed by simply adjusting screen transformations and redraw the screen's contents from an internal representation, rather than zooming into the pixel image. In other words, no aliasing problems occur.

Zooming can take on two forms. *Geometric zooming* simply provides a blow up of the graph content. *Semantic-zooming* means that the information content changes and more details are shown when approaching a particular area of the graph. The technical difficulty in this case is not with the zooming operation itself, but rather with assigning an appropriate level of detail, i.e., a sort of clustering, to subgraphs. The more general problem of clustering is addressed in Section 4.

Although conceptually simple, zoom and pan does create problems when used in interactive environments. Let us imagine, for example, the following setting: The graph being displayed is the road network of Europe and the user has zoomed into the area around Amsterdam. The user then wants to change the view of the area around Milano. Doing this without changing the zoom factor, at

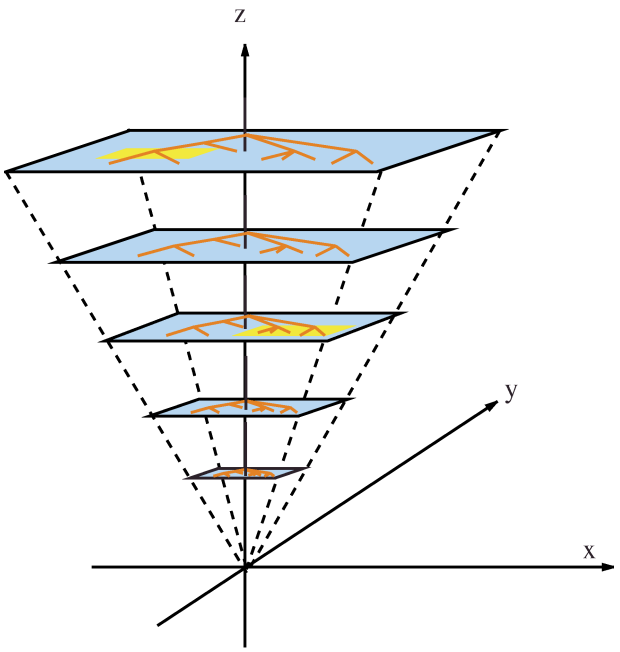


Fig. 15. A space-scale diagram. The yellow rectangles represent possible window positions in space-scale, yielding different zoom factors and pan positions. (Adapted from Furnas and Bederson [51].)

least temporarily, might be too slow because the user has to first zoom out, pan to Milano, and zoom in again. Furthermore, the user wants the system to make the necessary moves smoothly. A naive implementation might calculate the necessary changes for the pan and the zoom independently and perform the changes in parallel. The problem is that, when zooming in, the world view expands exponentially fast and the target point moves away faster than the pan can keep up with. The net result is that the target is approached nonmonotonically: It first moves away as the zoom dominates and only later comes back to the center of the view, which can be quite disturbing.

The zoom and pan problem is not restricted to graphs nor is the elegant solution proposed by Furnas and Bederson [51] to alleviate it. Nevertheless, graph visualization systems can greatly benefit from their approach, so we will provide a short description here. Furnas and Bederson introduce the concept of space-scale diagrams (see Fig. 15). The basic idea is to define an abstract space "by creating many copies of the original 2D picture, one at each possible magnification, and stacking them up to form an inverted pyramid." Points in the original image can be represented by rays that contain information both about the point and its magnification. Various combinations of (continuous) zoom and pan actions can then be described as paths in this space by describing the central position of a window parallel to the *x*–*y* plane. A cost, or "length," can also be associated to each path and, if the length is judiciously chosen, a minimum length path can represent an optimal combination of zoom and pan movements. Furnas and Bederson not only give a solution to the problem outlined above; space-scale diagrams can also be used to describe semantic zooming (instead of stacking the same picture in the pyramid, the content of the picture may depend on the magnification level), which also allows for the development of a specialized authoring system for semantic zooming [52].

3.2 Focus+Context Techniques

A well-known problem with zooming is that if one zooms on a focus, all contextual information is lost.[9] Such a loss of context can become a considerable usability obstacle. A set of techniques that allow the user to focus on some detail *without* losing the context can alleviate this problem. The term *focus+context* has been used to describe these techniques. They do not replace zoom and pan, but rather complement them. The complexity of the underlying data might make zoom an absolute necessity. However, focus+context techniques are a good alternative and full-blown applications systems often implement both.[10]

3.2.1 Fisheye Distortion

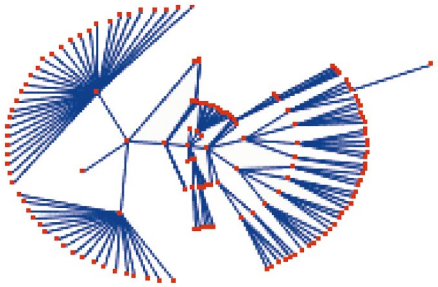
Graphical fisheye views are popular techniques for focus+context. Fisheye views imitate the well-known fisheye lens effect by enlarging an area of interest and showing other portions of the image with successively less detail (see Fig. 16).

We will describe some of the mathematics involved in the fisheye technique. Conceptually, the graph is mapped onto the plane and a "focus" point is defined (usually by the user). The distance from the focus to each node of the tree is then distorted by a function $h(x)$ and the distorted points, and connecting edges, are displayed. The function $h(x)$ should be concave, mapping monotonically the [0, 1] interval onto [0, 1] (see Fig. 17). The distortion created by the fisheye view is the consequence of the form of the function, which has a faster increment around 0 (hence affecting the nodes around the focus), with the increment slowing down when closing up to 1. The exact definition of the function may yield a lesser or stronger distorting effect. A simple distortion function, for example, used by Sarkar and Brown [110], [111] is: $h(x) = (d+1)/(d+1/x)$ (this is the function plotted in Fig. 17). The factor d is the so-called distortion factor, which can be set interactively by the user. It should be positive; the larger it is, the stronger the fisheye distortion. Fig. 18 shows the effect of this function (with $d = 4$) on the regular grid around the origin.

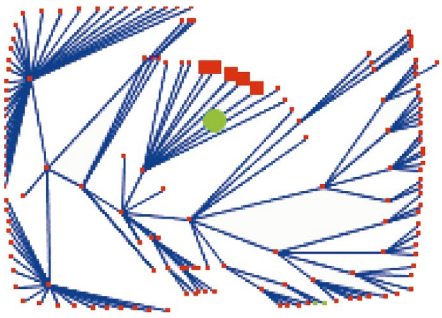
There are some variations to this basic scheme. What we have just described is usually referred to as a "polar" distortion, in the sense that it applies to the nodes radially in all directions starting from the focus point. An alternative is to use a "Cartesian" fisheye: The distance distortion is applied independently on the x and y directions before establishing the final position of the node (see again Fig. 16). Other variations are possible. Consult the overview of Carpendale et al. [18] or Keahey and Robertson [77] for further examples and for their visual effects. The final

9. Unless a separate window, for example, keeps the context visible, which is done by several systems. But, this solution is not fully satisfactory either.

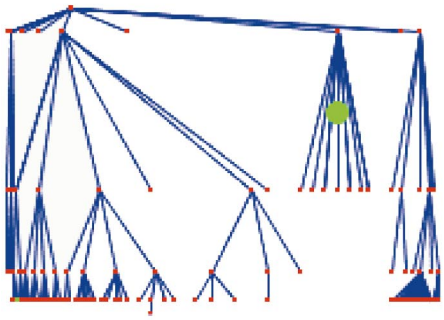
10. All techniques described in this section are *geometric*, i.e., they operate on the geometric representation of the underlying graphs. This is in contrast with a *logical* focus+context view described in an often-cited paper of Furnas [50]. In our view, the work of Furnas is more related to what we call "metrics," rather than to graphical focus+context. See Section 4.2 for further details.



(a)



(b)



(c)

Fig. 16. Fisheye distortion. (a) Represents the graph without the fisheye. (b) Uses polar fisheye, whereas (c) uses Cartesian fisheye with a different layout of the same graph. The green dots on (b) and (c) denote the focal points of the fisheye distortion. Note the extra edge-crossing on (b).

choice should depend on the style of the graph to be explored, as well as the layout algorithm in use.

This simple but powerful technique is an important form of navigation that complements zoom and pan. However, implementors should be aware of one of the pitfalls. The essence of a fisheye view is to distort the position of each node. If the distortion is faithfully applied, the edges connecting the nodes will also be distorted. Mathematically, the result of this distortion is a general curve. Standard graphics systems (e.g., X11, Java2D, OpenGL) do not offer the necessary facilities to transform lines into these curves easily (the curves can be rather complex). The implementer's only choice is, therefore, to approximate the original line segments with a high number of points, transform

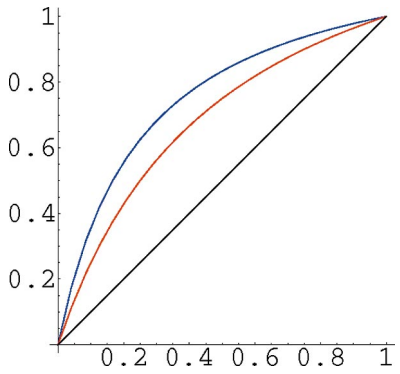


Fig. 17. The Sarkar-Brown distortion function with a distortion factor 2 (red curve) and 4 (blue curve).

those points, and display a polyline to approximate the ideal, transformed curve. The problem is that the number of approximating points must be relatively high if a smooth impression is desired (on average, 60 points per edge), which leads to a prohibitively large number of calculations and may make the responsiveness of the system sink to an unacceptably low level. The only viable solution is to apply the fisheye distortion on the node coordinates only and to connect the transformed nodes by straight-line edges. The consequence of this inexact solution is that unintended edge-crossings might occur (see, for example, the upper left quadrant of Fig. 16b). This is one of those typical situations when the pragmatism required by information visualization should prevail. If large graphs are explored, these extra intersection points do not really matter much and it is more important to keep the exploration tool fast.

3.2.2 *Focus+Context Layout Techniques*

The fisheye technique is independent of the layout algorithm and is defined as a separate processing step on the graphical layout of the graph. Interacting with fisheye means changing the position of the focus point and/or modifying the distortion value. This independence has positive and negative aspects. On the positive side, it allows for a modular organization of software in which fisheye is a separate step in the graph rendering pipeline somewhere between the layout module and the actual display. Fisheye can also be significantly faster than the layout algorithm, which is an important issue for interaction.

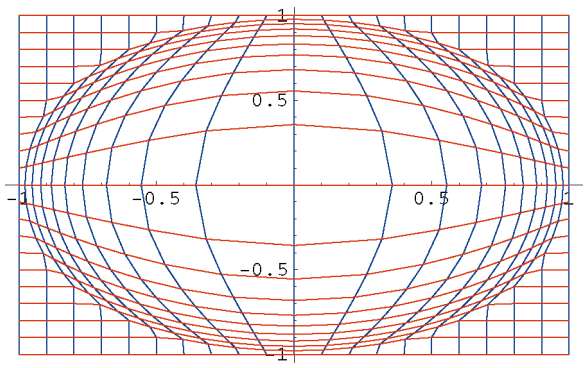


Fig. 18. Fisheye distortion of a regular grid of the plane. The distortion factor is 4.

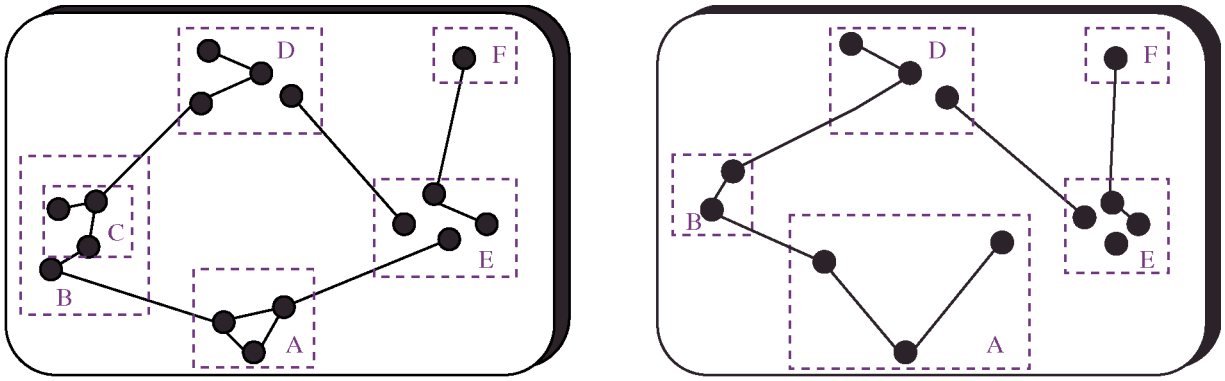


Fig. 19. Multifocal fisheye/zoom in a hierarchically clustered graph. The dotted rectangles denote the (logical) clusters. Note the disappearance of cluster C on the righthand side. (Adapted from Schaffer et al. [112].)

However, the fisheye distortion may destroy the aesthetics governing the layout algorithm. For example, as we have seen in the previous section, it can add new and unwanted edge-crossings.

An alternative is to build appropriate distortion possibilities into the layout algorithm itself, thereby merging the focus+context effects and the layout proper. Interacting with the distortion would mean interacting with (some) parameters governing the layout algorithm. The hyperbolic layout (see Section 2.5) does just that. The hyperbolic view of a graph, whether in 2D or 3D, produces a distorted view, not unlike the fisheye view (see Fig. 11). The equivalent of the focal point of the graphical fisheye view is the center of the Euclidean circle (or sphere) which is used to "map" the hyperbolic view onto the Euclidean space through either the Klein or the Poincaré model. Interacting with the view means changing the position of this center point within the graph.

Similar effects can be achieved by using 3D techniques (see also Section 2.4). By putting objects on 3D surfaces, for example, the view created by the perspective or parallel projections create a natural distortion on the 2D screen. In the Vitesse system [98], for example, the user has only limited 3D navigation facilities. The main goal of mapping objects onto a sphere or an ellipsoid is indeed to achieve a focus+context distortion. More complex surfaces (such as 3D surfaces of blended Gaussian curves) have also been used to achieve focus+context effects (see Carpendale et al. [17], [18]). Other 3D visualization techniques, already cited in Section 2.4 (such as the Perspective Wall [107]), apply this principle as well.

The hyperbolic layout is special because it is a graph layout algorithm that was developed with the focus+context distortion in mind. In fact, we do not know of any systematic research conducted on the existing, and more traditional, layout algorithms to decide whether such layout dependent distortions are possible or not, and, if yes, to exploit this feature in real systems. This is in spite of the fact that, at least in some cases, the possibility of applying such distortion control is clearly available. For example, Fig. 5 shows a balanced view of a tree, using a balloon layout algorithm [87]. This algorithm defines the radii of the circles by taking the number of descendents into account. The algorithm can be easily directed to give one of the circles a larger "share" of the display space by shrinking all the others, thereby creating a focus+context effect on that circle [63]. We think that such research would provide valuable input for the implementors of graph visualization systems.

3.2.3 Further Issues in Focus+Context Techniques

There are further issues in the area of focus+context that can be of interest, some of which could be the basis for future research as well (a general characterization and taxonomy of distortion techniques is also presented in Leung and Apperly [84]). For example, fisheye is based on the choice of a distortion function, but we presented only a simple version here, used by Sarkar and Brown. This function can be replaced by others with different distortion features (arctan or tanh functions, piecewise linear approximations to speed up processing, etc.) [44], [77], [111]. The techniques can also be extended to 3D [19]. Also, just as we could speak about "semantic zoom," one could also refer to "semantic focus+context," meaning that, when the distortion becomes too "extreme," in some sense, nodes might disappear after all. Sarkar and Brown describe this technique in their paper [110], but finer control over this facility might lead to new insights as well. Note that the space-scale diagrams [51] (see Section 3.1) can also be used to model fisheye distortions, which may lead to interesting results in combining (semantic) fisheye with zoom and pan. Finally, multifocal focus+context methods can also be applied [18], [76], [77], allowing the user to simultaneously concentrate on several important areas of the graph or to use the system in a cooperative environment [98].

An interesting example that combines various techniques, including multifocal zoom and focus+context, is provided by Schaffer et al. [112]. Their system also shows the fundamental importance of clustering, which we address in Section 4. They consider graphs that already have a hierarchical clustering. The lefthand side of Fig. 19 shows a drawing of the initial graph. The dotted rectangles denote the logical clusters (they appear on the figure only for the sake of the explanation; they would not necessarily appear on a real screen). The righthand side of the same figure shows the same graph after a multifocal zoom/fisheye action on clusters A and D. These clusters are now bigger, while the other clusters have shrunk. Moreover, cluster C has disappeared as a result of a sort of a "semantic fisheye" action on the graph. Schaffer et al. describe the

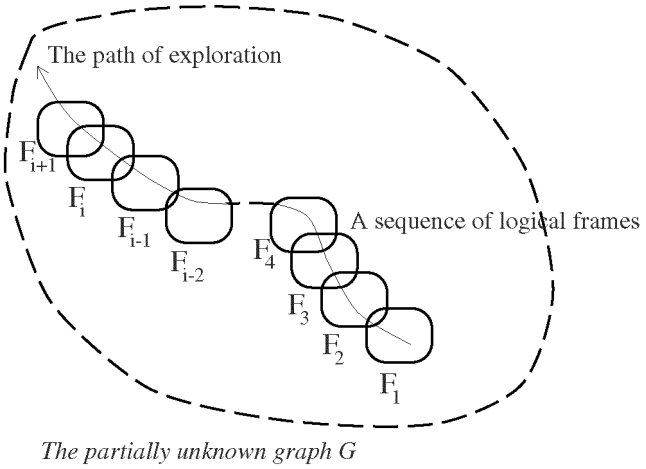


Fig. 20. Exploration of a huge graph. (Adapted from Huang et al. [69].)

mathematics of distortion and shrinking used to achieve these results. Similar ideas can also be found in the DA–TU system of Huang and Eades [70]. However, much remains to be done in combining these different approaches to achieve a coherent set of navigation techniques.

3.3 Incremental Exploration and Navigation

We have emphasized several times that the size of the graph is a major problem in graph visualization applications. There are cases when this size is so huge that it becomes impossible to handle the full graph at any time; the World Wide Web is an obvious example. *Incremental exploration techniques* are good candidates for such situations. The system displays only a small portion of the full graph and other parts of the graph are displayed as needed. The advantage of such an incremental approach is that, at any given time, the subgraph to be shown on the screen may be limited in size, hence, the layout and interaction times may not be critical any more. This approach to graph exploration is still relatively new, but interesting results in the area are already available, see, for example [14], [40], [68], [69], [99], [130].

Incremental exploration means that the system places a visible "window" on the graph, somewhat similar to what pan does. Exploration means to move this window (also referred to as *logical frames* by Huang et al. [68]) along some trajectory (see Fig. 20). Implementation of such incremental exploration has essentially two aspects, namely:

- decide on a strategy to generate new logical frames
- reposition the content of the logical frame after each change.

Generating new logical frames is always under the control of the user. In some cases, the logical frame simply contains the nodes visited so far. This is the case, for example, in the NESTOR tool, implemented by Zeiliger [130], which uses incremental exploration to record a history of the user's surfing the World Wide Web: Newly accessed web pages are simply added to the logical frame to generate a new one. Huang et al. [68] (who also implemented a tool along the same lines to explore the World Wide Web [69]) anticipate the user's future interaction by adding not only a new node to a frame, but also its immediate neighbors. Huang et al. [68] or North [99] also include a control over throwing away some part of the logical frame to avoid saturation on the screen.

As far as the repositioning is concerned, the simplest solution is to use the same layout algorithm for each logical frame. This is done, for example, by Huang et al. [68]. (Note that the latter use a modified spring algorithm. This is one case where the relatively small graph on the screen makes the use of a force-directed method perfectly feasible in graph visualization.) North [99] and Brandes and Wagner [14] go further by providing dynamic control over the parameters that direct the layout algorithms.

As said above, this line of visual graph management is still quite new, but we think that it will gain in importance in the years to come and that it will complement the navigation and exploration methods described elsewhere in this survey.

4 Clustering

As mentioned earlier, it is often advantageous to reduce the number of visible elements being viewed. Limiting the number of visual elements to be displayed both improves the clarity and simultaneously increases performance of layout and rendering [78]. Various "abstraction" and "reduction" techniques have been applied by researchers in order to reduce the visual complexity of a graph. One approach is to perform clustering.

Clustering is the process of discovering groupings or classes in data based on a chosen semantics. Clustering techniques have been referred to in the literature as *cluster analysis, grouping, clumping, classification, and unsupervised pattern recognition* [41], [89]. We will refer to clustering that uses only structural information about the graph as *structure-based* clustering (also referred to as identifying *natural clusters* [109]). The use of the semantic data associated with the graph elements to perform clustering could be termed *content-based* clustering.

Although content-based clustering can yield groupings which are most appropriate for a particular application and can even be combined with structure-based clustering, most mentions of clustering in graph visualization are references to purely structure-based clustering, with a few notable exceptions [91], [105]. This is probably due to the fact that content-based clustering requires application-specific data and knowledge. Any application which implements content-based clustering is likely to be so specialized to a problem domain that it is no longer general enough for use in other application areas. Furthermore, an advantage of using structure-based clustering is that natural clusters often retain the structure of the original graph, which can be useful for user orientation in the graph itself.

It is important to note that clustering can be used to accomplish functions such as filtering and search. In visualization terms, filtering usually refers to the deemphasis or removal of elements from the view, while search usually refers to the emphasis of an element or group of elements. Both filtering and search can be accomplished by partitioning elements into two or more groups and, then, emphasizing one of the groups.

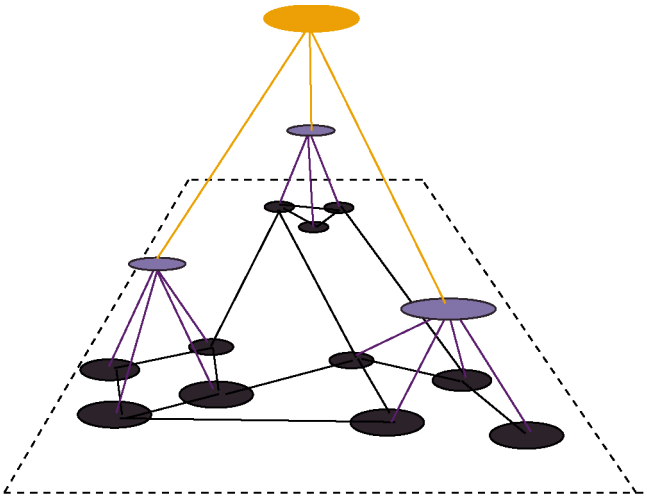


Fig. 21. A structure induced by hierarchical clustering. (Adapted from Eades and Feng [37].)

By far the most common clustering approach in graph visualization is to find clusters which are disjoint or mutually exclusive, as opposed to clusters that overlap (found by a process called *clumping*). Disjoint clusters are simpler to navigate than overlapping clusters because a visit of the clusters only visits the members once. It should be noted, however, that it is not always possible to find disjoint clusters, such as in the case of language-oriented or semantic topologies.

A common technique for finding natural clusters is to choose the clustering with the least number of edges between members. This technique is described by Mirkin [89]. It is also known as the Ratio Cut technique in VLSI design [124]. This technique extends to the case when edges have a weight. The task is then to minimize the total weight of the edges connecting members [109]. Natural clusters can also be obtained by applying a spring model (see below).

4.1 Layout of a Clustered Graph

After discovering clusters within the data, we can reduce the number of elements to display by restricting our view to the clusters themselves. This provides an overview of the structure and allows us to retain a context while reducing visual complexity. Looking at the simpler and smaller clustered graph, the user should be better able to grasp the overall structure of the graph. Most algorithms look for a balance between the number of clusters and the number of nodes within clusters [1], [31]. A small number of clusters allows for fast processing and navigation. However, this number should not be too small because, otherwise, the visible information content is too low.

A common technique is to represent the clusters with glyphs and treat them as super-nodes in a higher-level or *compound graph*, which we can now navigate instead of the original graph. Some approaches have already been proposed [37], [112]. Huang and Eades [70] also give a precise definition of how edges between super-nodes can be induced (they refer to this idea as *abridgement*). This technique has also been implicitly implemented in many other visualization systems. One original solution is to omit the edges and position the nodes in a way that indicates their connectivity [126]. This solution eliminates the problem of edge-crossings and reduces visual clutter.

If clustering is performed by successively applying the same clustering process to groups discovered by a previous clustering operation, the process is referred to as *hierarchical clustering* [89]. A containment hierarchy will result from hierarchical clustering and this may be navigated as a tree, with each cluster represented as a node in the tree (see Fig. 21). Hierarchical clustering can therefore be used to induce a hierarchy in a graph structure that might not otherwise have a hierarchical structure.

The approaches discussed until now involve first finding logical clusters, then laying out the graph of clusters. A completely different approach to clustering is based on force-directed layout. It lets forces between nodes influence the position of the node in the layout. All nodes in the system exert repulsive force on the others and related nodes are attracted to each other. After several iterations in which the positions are adjusted according to the calculated force, the system stabilizes, yielding clusters which are visually apparent. In a case study of Narcissus [60], the authors report that this technique can produce useful clusters in a relatively small number of iterations. As with other N-body problems, the complexity is $O(N^3)$. Another example of clustering by layout is described for the SemNet system [42], where clustering is accomplished by using semantic information to determine the positioning of nodes.

4.2 Node Metrics for Clustering

In order to cluster a graph, we must use numerical measures associated with the nodes. A node metric can be used to measure or to quantify an abstract feature associated with a node in order to compare it with others of the same type and acquire a ranking. A metric can be implemented as a numeric computable function. Clustering can be accomplished by assigning elements to groups according to their metric value. Metrics can also be used to implement search or filtering in which elements with a certain metric value or a value above a threshold are highlighted.

The term *metric*, or *node metric*, has been used in many different ways in graph visualization. In this survey, we will use the term to refer to *a measure that is associated with a node in the graph*. We have identified the concept of node metrics in several places in the literature [11], [50], [61], [78]. Of course, similar concepts can be applied to metrics associated with edges.

A metric is structure-based if it only uses information about the structure of the graph. A metric is content-based if it uses information or data associated with the node such as text. The advantage of a structural metric is that no domain knowledge is required. This makes a structural metric useful for all applications. It is possible, of course, to combine structural and content-based metrics for more powerful effects. A simple approach is to allow the user to add an application-specific "weight" to the nodes, which is then combined with the structural metric [50], [61], [62].

An example of a structural metric is the degree of a node (i.e., the number of edges connected to the node). With such a metric, the application could exclusively display the nodes with a degree higher than or equal to a

threshold value. This would give a view of data which shows the nodes that have the largest number of relations with other nodes. A metric more specific to trees (called the Strahler metric [120]) has been applied in Fig. 22, in which nodes with the highest Strahler metric values generate a *skeleton* or backbone which is then emphasized (see Herman et al. [61], [62]).

Metrics can also be composed due to their numeric nature [62]. By choosing, for example, the weighted average of metrics, the user can choose how much influence a particular feature has on the resulting composed metric and thereby influence the resulting clustering. The *Degree of Interest (DOI)* function of Furnas [50] is also an example of a metric that is composed of two other metrics (in this case, a metric based on distance and a level of detail).

Node metrics can be used for many different purposes and, in our view, all the possible applications have not yet been fully explored. For instance, metrics can also be used to govern a spanning tree extraction procedure (see Section 2.3). Furnas's DOI function has been used to generate a focus+context view of the graph.[11] In another application, metrics are used to influence layout [127].

Once a subset of nodes has been selected, as with a skeleton, a method of representing the unselected nodes must be chosen. In the case of clustering, the selected set of nodes is the set of super-nodes or the groups themselves. Kimelman et al. name three possible approaches [78] (see Fig. 22):

- *Ghosting*: deemphasizing nodes, or relegating nodes to the background.
- *Hiding*: simply not displaying the unselected nodes. This is also referred to as folding or eliding.
- *Grouping*: grouping nodes under a new super-node representation.

These approaches may be combined, for example, with clusters represented by transparent super-nodes used by Sprenger et al. [116] in the IVORY system. Fig. 22c demonstrates an alternative where the size and the shape of the glyph representing the grouping is used to indicate the structure of the underlying subgraph. The resulting graph, technically a compound graph, is a sort of high-level map or *schematic view* [23], [62] of the original graph, which is useful for navigation of the original graph.

Clustering is full of challenges and is applied in many different fields, which has the unfortunate consequence that results about clustering are disseminated in journals and conferences addressing very different topics. This makes it difficult to gather the results into a unified theory or into a structured set of methodologies. Surprisingly, the book by Battista et al. [5] does not include a chapter on clustering, although the Graph Drawing Symposia welcomes papers on the topic every year. Our feeling is that this issue should receive more attention in the future, especially from the information visualization community.

11. As mentioned earlier, although Furnas referred to this technique as "fish-eye," his technique is not limited to fish-eye in the geometric sense, as described in Section 3.2.1.

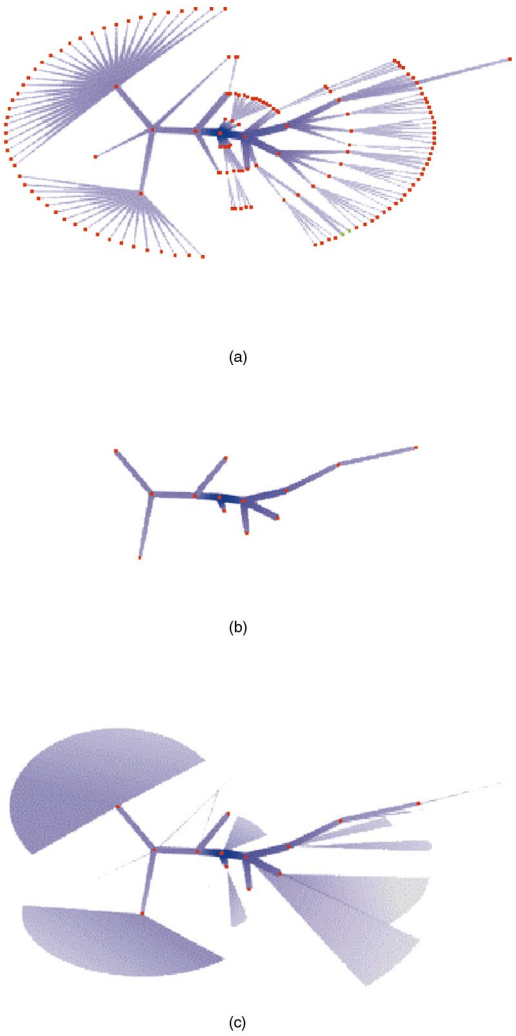


Fig. 22. Different schematic views of a tree: (a) ghosting, (b) hiding, and (c) grouping.

5 Systems

The area of graph visualization has reached a level of maturity in which large applications and application frameworks are being developed. However, it is difficult to enumerate all the systems because of the sheer quantity. Furthermore, some of them have a short lifetime because they are research tools and others are embedded in specialized applications. An overview of all graph visualization systems would go beyond the scope of this survey. However, we have already referred to a number of systems in earlier sections, based on features that we found interesting or important. Some other systems also caught our attention. Without any claim to completeness, we briefly describe a few additional systems below.

Efforts to develop software libraries and frameworks have been underway in several places. Some libraries are directed at mathematicians and include large libraries of algorithms, while others are meant for more general application. Some of the libraries and frameworks that are available are GTL [45], LINK [8], GFC [21], GDT [55], and GVF [64]. Although there is no widely used standard for

graph description formats, GML [66] and GraphXML [65] are available.

SemNet [42] is one of the few systems to provide graph editing while still providing a comprehensive set of tools to visualize large graphs. It is also one of the earliest complete systems that we know about.

Clustering has been applied by many older systems such as SemNet [42], Narcissus [60], SKETCH [118], and the Navigational View Builder [91]. Some newer systems that cluster graphs are NicheWorks [126], DA–TU [70], STARLIGHT [105], and a new system used by Bell Laboratories [58] for network visualization.

NicheWorks is an example of a complete system implementation that can be adapted for very specific applications. As an example, it has been used to visualize Y2K related problems [39]. The *fsviz* system of Carrière and Kazman [20], the da Vinci system of the University of Bremen [48], or the Latour system developed at CWI [63] fall into the same category. We should also mention the company called Tom Sawyer Software,[12] which offers a number of products based on various graph drawing techniques.

A few systems stand out because of unique features. The STARLIGHT [105] system performs content-based clustering and allows multiple mappings and layouts. It is one of the few systems that allows a 3D graph to be mapped to locations on a plane (for associating nodes or entire graphs with geographical positions). Shiozawa et al. [114] use a similar type of 3D to 2D mapping in order to view cell dependencies in a spreadsheet application. Another system, SDM [24] is unique because of a method of filtering in which nodes of interest are selected from a cityscape view by a plane above them. A similar cityscape view of nodes is used by Chen and Carr [22]. A system called WebPath [46] uses a fog effect in a 3D rendering of web history to limit the window of viewing. Graphs have also been used in an attempt to understand images and the transformations on them, where edges represent operations [85]. A system for viewing Bayesian Belief Networks [129] is one of a unique few (including [8], [63]) to employ animation for informative purposes. A highly interactive system called Constellation [95] has sophisticated zooming and highlighting features that facilitate the analysis of linguistic networks.

The World Wide Web is one of the typical application areas where graph visualization may play an important role in the future. H3View [93], based on hyperbolic viewing (see Section 2.5), is part of a Web site management tool of SGI, whereas the similar ideas of Lamping et al. [82], [83] are also exploited by a commercial spin-off of Xerox, called Inxight.[13] Earlier in this survey, we referred to NESTOR [1] or WebOFDAV [69], which can be used as web navigation tools. Other examples in this category are the Harmony Browser [1], Mapa [32], or Fetuccino [7] (the latter also combines the results of a web search engine with graph visualization).

12. http://www.tomsawyer.com.
13. http://www.inxight.com.

6 Journals and Conferences

This survey is based on an extensive literature overview drawn from various conferences and journals. One of the difficulties of the field is that results are spread over a large number of different publications. To help the reader in pursuing research in the area, we list here some of the main publications which may be of interest:

- The *Graph Drawing Symposia* are organized yearly at various locations in the World. The proceedings are published by Springer-Verlag. These symposia have evolved into the traditional meeting places of the graph drawing community.
- The new *Journal of Graph Algorithms and Applications (JGAA)* is an on-line journal which gathers a similar community as the graph drawing symposia. The home page of the journal is at Brown University,[14] but Oxford University Press will also publish the collected papers in book formats.
- Graph drawing has strong relationships with computational geometry and algorithms. As a consequence, specialized journals like *Computational Geometry: Theory and Applications* or *Algorithmica* might also be a valuable source, although the papers in these journals tend to be much more "mathematical", hence, more difficult to read for the computer graphics and information visualization communities.
- As said before, the yearly *CHI'XX* and *UIST'XX* conferences, both sponsored by ACM SIGCHI, often contain important papers for information visualization due to the importance of the user interface issue. Similarly, the *ACM Transactions on Human Computer Interaction* can be a valuable source of information.
- The yearly *InfoViz'XX* symposia form a separate track within the well-known *IEEE Visualization* conference. These symposia, as well as the Visualization conference itself, have become one of the leading events in the area by now.
- Somewhat confusingly, there is also a yearly *IEEE Conference on Information Visualization* which, however, has no real connection to the *InfoViz'XX* symposia (besides being sponsored by IEEE, too). Our own experience is that the academic level of *InfoViz'XX* is somewhat better.
- What was known before as the series of *Eurographics Workshop on Scientific Computing'XX* has recently changed its name to *Data Visualization'XX*, with information visualization as a separate track. The workshops have become joint Eurographics IEEE TCVG symposia and are considered as the European "sister" conference to IEEE Visualization.
- Some traditional computer graphics journals, like the *IEEE Transactions on Visualization and Computer Graphics* or the *Computer Graphics Forum* (which include the proceedings of the Eurographics conferences, too), have an increasing number of papers in information visualization.

14. http://www.cs.brown.edu/publications/jgaa.

- Finally, application-oriented journals or conference proceedings may also include papers on information visualization related to their respective application area. Examples include the proceedings of the yearly *XXth World Wide Web* or the *Digital Library'XX* conferences.

Obviously, the list is not exhaustive, but, hopefully, it is still useful for the reader as a starting point.

REFERENCES

[1] C.J. Alpert and A.B. Kahng, "Recent Developments in Netlist Partitioning: A Survey," *Integration: The VLSI J.,* vol. 19, pp. 1-81, 1995.

[2] K. Andrews, "Visualizing Cyberspace: Information Visualization in the Harmony Internet Browser," *Proc. IEEE Symp. Information Visualization (InfoViz '95),* pp. 97-105, 1995.

[3] P.K. Argawal, B. Aronov, J. Pach, R. Pollack, and M. Sharir, "Quasi-Planar Graphs Have a Linear Number of Edges," *Proc. Symp. Graph Drawing, GD '95,* pp. 1-7, 1995.

[4] G. di Battista, P. Eades, R. Tamassia, and I.G. Tollis, "Algorithms for Drawing Graphs: An Annotated Bibliography," *Computational Geometry: Theory and Applications,* vol. 4, no. 5, pp. 235-282, 1994.

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

[6] R.A. Becker, S.G. Eick, and A.R. Wilks, "Visualizing Network Data," *IEEE Trans. Visualization and Computer Graphics,* vol. 1, no. 1, pp. 16-28, 1995.

[7] I. Ben-Shaul, M. Herscovici, M. Jacovi, Y.S. Maarek, D. Pelleg, M. Shtalhaim, V. Soroka, and S. Ur, "Adding Support for Dynamic and Focused Search with Fetuccino," *Proc. Eighth Int'l World Wide Web Conf.,* pp. 575-587, 1999.

[8] J. Berry, N. Dean, M. Goldberg, G. Shannon, and S. Skiena, "Graph Drawing and Manipulation with LINK," *Proc. Symp. Graph Drawing GD '97,* pp. 425-437, 1999.

[9] F. Bertault, "A Force-Directed Algorithm that Preserves Edge Crossing Properties," *Proc. Symp. Graph Drawing, GD '99,* pp. 351-358, 1999.

[10] J. Blythe, C. McGrah, and D. Krackhardt, "The Effect of Graph Layout on Inference from Social Network Data," *Proc. Symp. Graph Drawing, GD '95,* pp. 40-51, 1995.

[11] R.A. Botafogo, E. Rivlin, and B. Schneiderman, "Structural Analysis of Hypertexts: Identifying Hierarchies and Useful Metrics," *ACM Trans. Information Systems,* vol. 10, no. 2, 1992.

[12] F.J. Brandenburg, M. Himsolt, and C. Rohrer, "An Experimental Comparison of Force-Directed and Randomized Graph Drawing Algorithms," *Proc. Symp. Graph Drawing GD '95,* 1996.

[13] U. Brandes, G. Shubina, and R. Tamassia, "Improving Angular Resolution in Visualizations of Geographic Networks," *Data Visualization '2000, Proc. Joint Eurographics and IEEE TCVG Symp. Visualization,* to appear.

[14] U. Brandes and D. Wagner, "A Bayesian Paradigm for Dynamic Graph Layout," *Proc. Symp. Graph Drawing GD '97,* pp. 236-247, 1997.

[15] S.K. Card, G.G. Robertson, and W. York, "The WebBook and the Web Forager: An Information Workspace for the World Wide Web." *Human Factors in Computer Systems, CHI '96 Conf. Proc.,* pp. 111-117, 1996.

[16] *Readings in Information Visualization,* S.K. Card, J.D. Mackinlay, and B. Shneiderman, eds. Morgan Kaufmann, 1999.

[17] M.S.T. Carpendale, D.J. Cowperthwaite, and F.D. Fracchia, "3D Pliable Surfaces," *Proc. UIST '95 Symp.,* pp. 217-266, 1995.

[18] M.S.T. Carpendale, D.J. Cowperthwaite, F.D. Fracchica, and T. Shermer, "Graph Folding: Extending Detail and Context Viewing into a Tool for Subgraph Comparisons," *Proc. Symp. Graph Drawing GD '95,* pp. 127-139, 1996.

[19] M.S.T. Carpendale, D.J. Cowperthwaite, and F.D. Fracchia, "Extending Distortion Viewing from 2D to 3D," *IEEE Computer Graphics and Applications,* vol. 17, no. 4, pp. 42-51, 1997.

[20] J. Carrière and R. Kazman, "Research Report: Interacting with Huge Hierarchies: Beyond Cone Trees," *Proc. IEEE Conf. Information Visualization '95,* pp. 74-81, 1995.

[21] C.L. Cesar, *Graph Foundation Classes for Java.* IBM, http://www.alphaWorks.ibm.com/tech/gfc, 1999.

[22] C. Chen and L. Carr, "Visualizing the Evolution of a Subject Domain: A Case Study," *Proc. IEEE Visualization '99 Conf.,* pp. 449-452, 1999.

[23] M.C. Chuah, "Dynamic Aggregation with Circular Visual Designs," *Proc. IEEE Symp. Information Visualization (InfoViz '98),* pp. 30-37, 1998.

[24] M.C. Chuah, S.F. Roth, J. Mattis, and J. Kolojejchick, "SDM: Malleable Information Graphics," *Proc. IEEE Symp. Information Visualization,* pp. 36-42, 1995.

[25] H.S.M. Coxeter, *Introduction to Geometry.* John Wiley & Sons, 1973.

[26] I.F. Cruz and R. Tamassia, "Online Tutorial on Graph Drawing," http://www.cs.brown.edu/people/rt/papers/gd-tutorial/gd-constraints.pdf. year?

[27] I.F. Cruz and J.P. Twarog, "3D Graph Drawing with Simulated Annealing," *Proc. Symp. Graph Drawing GD '95,* pp. 162-165, 1995.

[28] R. Davidson and D. Harel, "Drawing Graphs Nicely Using Simulated Annealing," *ACM Trans. Graphics,* vol. 15, no. 4, pp. 301-331, 1996.

[29] E. Dengler and W. Cowan, "Human Perception of Laid-Out Graphs," *Proc. Symp. Graph Drawing GD '98,* pp. 441-444, 1998.

[30] A. Denise, M. Vasconcellos, and D.J.A. Welsh, "The Random Planar Graph," *Congressus Numerantium,* vol. 113, pp. 61-79, 1996.

[31] C.A. Duncan, M.T. Goodrich, and S.G. Kobourov, "Balanced Aspect Trees and Their Use for Drawing Very Large Graphs," *Proc. Symp. Graph Drawing GD '98,* pp. 111-124, 1998.

[32] D. Durand and P. Kahn, "MAPA," *Proc. Ninth ACM Conf. Hypertext and Hypermedia (Hypertext '98),* 1998.

[33] P. Eades, "A Heuristic for Graph Drawing," *Congressus Numerantium,* vol. 42, pp. 149-160, 1984.

[34] P. Eades and K. Sugiyama, "How to Draw a Directed Graph," *J. Information Processing,* vol. 13, no. 4, pp. 424-434, 1990.

[35] P. Eades, "Drawing Free Trees," *Bulletin of the Inst. for Combinatorics and Its Applications,* pp. 10-36, 1992.

[36] P. Eades and S.H. Whitesides, "Drawing Graphs in Two Layers," *Theoretical Computer Science,* vol. 131, no. 2, pp. 361-374, 1994.

[37] P. Eades and Q.-W. Feng, "Multilevel Visualization of Clustered Graphs," *Proc. Symp. Graph Drawing GD '96,* pp. 101-112, 1997.

[38] P. Eades, M.E. Houle, and R. Webber, "Finding the Best Viewpoints for Three-Dimensional Graph Drawings," *Proc. Symp. Graph Drawing GD '97,* pp. 87-98, 1998.

[39] S.G. Eick, "A Visualization Tool for Y2K," *Computer,* vol. 31, no. 10, pp. 63-69, 1998.

[40] J. Eklund, J. Sawers, and R. Zeiliger, "NESTOR Navigator: A Tool for the Collaborative Construction of Knowledge through Constructive Navigation," *Proc. Ausweb '99, Fifth Australian World Wide Web Conf.,* 1999.

[41] B. Everitt, *Cluster Analysis,* first ed. Heinemann Educational Books Ltd., 1974.

[42] K.M. Fairchild, S.E. Poltrock, G.W. Furnas, "SemNet: Three-Dimensional Representation of Large Knowledge Bases," *Cognitive Science and Its Applications for Human-Computer Interaction,* pp. 201-233, Lawrence Erlbaum Assoc., 1988.

[43] K.M. Fairchild, "Information Management Using Virtual Reality-Based Visualisations," *Virtual Reality: Application and Explorations,* Academic Press, 1993.

[44] A. Formella and J. Keller, "Generalized Fisheye Views of Graphs," *Proc. Symp. Graph Drawing GD '95,* pp. 242-253, 1995.

[45] M. Forster, A. Pick, and M. Raitner, *Graph Template Library,* Univ. of Passau, http://infosun.fmi.uni-passau.de/GTL/, 1999.

[46] E. Frécon and G. Smith, "WebPath—A Three Dimensional Web History," *Proc. IEEE Symp. Information Visualization (InfoViz '98),* 1998.

[47] A. Frick, A. Ludwig, and H. Mehldau, "A Fast Adaptive Layout Algorithm for Undirected Graphs," *Proc. Symp. Graph Drawing GD '93,* pp. 389-403, 1994.

[48] M. Fröhlich and M. Werner, "Demonstration of the Interactive Graph Visualization System da Vinci," *Proc. DIMACS Workshop Graph Drawing '94,* 1995.

[49] T.M.J. Fruchterman and E.M. Reingold, "Graph Drawing by Force-Directed Placement," *Software—Practice & Experience,* vol. 21, pp. 1,129-1,164, 1991.

[50] G.W. Furnas, "Generalized Fisheye Views," *Human Factors in Computing Systems, CHI '86 Conf. Proc.,* pp. 16-23, 1986.

[51] G.W. Furnas and B.B. Bederson, "Space-Scale Diagrams: Understanding Multiscale Interfaces," *Human Factors in Computing Systems, CHI '95 Conf. Proc.,* pp. 234-241, 1995.
[52] G.W. Furnas and X. Zhang, "MuSE: A Multi-Scale Editor," *Proc. UIST '98 Symp.,* 1998.
[53] M.R. Garey and D.S. Johnson, "Crossing Number is NP-Complete," *SIAM J. Algebraic and Discrete Methods,* vol. 4, no. 3, pp. 312-316, 1983.
[54] A. Garg and R. Tamassia, "On the Computational Complexity of Upward and Rectilinear Planarity Testing," *Proc. Symp. Graph Drawing, GD '95,* pp. 286-297, 1995.
[55] *Graph Drawing Toolkit.* Third Univ. of Rome, http://www.dia.uniroma3.it/~gdt/, 1999.
[56] C. Gunn, "Visualizing Hyperbolic Space," *Proc. Eurographics Workshop Computer Graphics and Math.,* pp. 299-313, 1992.
[57] B. Hausmann, B. Slopianka, and H.-P. Seidel, "Exploring Plane Hyperbolic Geometry," *Proc. Workshop Visualization and Math.,* pp. 21-36, 1998.
[58] T. He, "Internet-Based Front-End to Network Simulator," *Data Visualization '99, Proc. Joint Eurographics and IEEE TCVG Symp. Visualization,* pp. 247-252, 1999.
[59] M. Hemmje, C. Kunkel, and A. Willet, "LyberWorld—A Visualization User Interface Supporting Fulltext Retrieval," *Proc. ACM SIGIR '94,* 1994.
[60] R.J. Hendley, N.S. Drew, A.M. Wood, and R. Beale, "Narcissus: Visualising Information," *Proc. IEEE Symp. Information Visualization,* pp. 90-96, 1995.
[61] I. Herman, M. Delest, and G. Melançon, "Tree Visualization and Navigation Clues for Information Visualization," *Computer Graphics Forum,* vol. 17, no. 2, pp. 153-165, 1998.
[62] I. Herman, M.S. Marshall, G. Melançon, D.J. Duke, M. Delest, and J.-P. Domenger, "Skeletal Images as Visual Cues in Graphs Visualization," *Data Visualization '99, Proc. Joint Eurographics and IEEE TCVG Symp. Visualization,* pp. 13-22, 1999.
[63] I. Herman, G. Melançon, M.M. de Ruiter, and M. Delest, "Latour—A Tree Visualization System," *Proc. Symp. Graph Drawing GD '99,* pp. 392-399, 1999. A more detailed version in: *Reports of the Centre for Math. and Computer Sciences,* Report number INS-R9904, available at: http://www.cwi.nl/InfoVisu/papers/LatourOverview.pdf, 1999.
[64] I. Herman, M.S. Marshall, and G. Melançon, "An Object-Oriented Design for Graph Visualization," *Reports of the Centre for Math. and Computer Sciences,* Report no. INS-R0001, available at: http://www.cwi.nl/InfoVisu/GVF/GVF.pdf, 2000.
[65] I. Herman and M.S. Marshall, "GraphXML," *Reports of the Centre for Math. and Computer Sciences,* available at: http://www.cwi.nl/InfoVisu/GVF/GraphXML/GraphXML.pdf, 1999.
[66] M. Himsolt, *GML—Graph Modelling Language,* Univ. of Passau, http://infosun.fmi.uni-passau.de/Graphlet/GML/, 1997.
[67] J. Hopcroft and R.E. Tarjan, "Efficient Planarity Testing," *J. ACM,* vol. 21, no. 4, pp. 549-568, 1974.
[68] M.L. Huang, P. Eades, and J. Wang, "Online Animated Graph Drawing Using a Modified Spring Algorithm," *J. Visual Languages and Computing,* vol. 9, no. 6, 1998.
[69] M.L. Huang, P. Eades, and R.F. Cohen, "WebOFDAV—Navigating and Visualizing the Web On-Line with Animated Context Swapping," *Proc. Seventh World Wide Web Conf.,* pp. 636-638, 1998.
[70] M.L. Huang and P. Eades, "A Fully Animated Interactive System for Clustering and Navigating Huge Graphs," *Proc. Symp. Graph Drawing GD '98,* pp. 374-383, 1998.
[71] C.-S. Jeong and A. Pang, "Reconfigurable Disc Trees for Visualizing Large Hierarchical Information Space" *Proc. IEEE Symp. Information Visualization (InfoViz '98),* 1998.
[72] B. Johnson and B. Schneiderman, "Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures," *Proc. IEEE Visualization '91,* pp. 275-282, 1991.
[73] M. Juenger and P. Mutzel, "2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms," *J. Graph Algorithms and Applications,* vol. 1, pp. 33-59, 1997.
[74] D. Jungnickel, *Graphs, Networks and Algorithms.* Springer Verlag, 1999.
[75] T. Kamada and S. Kawai, "An Algorithm for Drawing General Undirected Graphs," *Information Processing Letters,* vol. 31, pp. 7-15, 1989.
[76] K. Kaugars, J. Reinfelds, and A. Brazma, "A Simple Algorithm for Drawing Large Graphs on Small Screens," *Proc. Symp. Graph Drawing GD '94,* pp. 278-281, 1995.
[77] T.A. Keahey and E.L. Robertson, "Techniques for Non-Linear Magnification Transformations," *Proc. IEEE Symp. Information Visualization (InfoViz '97),* pp. 38-45, 1997.
[78] D. Kimelman, B. Leban, T. Roth, and D. Zernik, "Reduction of Visual Complexity in Dynamic Graphs," *Proc. Symp. Graph Drawing GD '93,* 1994.
[79] M.R. Laguna, R. Martí, and V. Vals, "Arc Crossing Minimization in Hierarchical Digraphs with Tabu Search," *Computers and Operations Research,* vol. 24, no. 12, pp. 1,165-1,186, 1997.
[80] M. Laguna and R. Martí, "GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization," *INFORMS J. Computing,* vol. 11, pp. 44-52, 1999.
[81] M. Laguna and R. Martí, "Heuristics and Meta-Heuristics for 2-Layer Straight Line Crossing Minimization," URL: http://www-bus.colorado.edu/Faculty/Laguna/, 1999.
[82] J. Lamping, R. Rao, and P. Pirolli, "A Focus+context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies," *Human Factors in Computing Systems, CHI '95 Conf. Proc.,* 1995.
[83] J. Lamping and R. Rao, "The Hyperbolic Browser: A Focus+context Technique for Visualizing Large Hierarchies," *J. Visual Languages and Computing,* vol. 7, no. 1, pp. 33-55, 1996.
[84] Y.K. Leung and M.D. Apperly, "A Review and Taxonomy of Distortion-Oriented Presentation Techniques," *ACM Trans. Computer-Human Interaction,* vol. 1, no. 2, pp. 126-160, 1994.
[85] K.L. Ma, "Image Graphs—A Novel Approach to Visual Data Exploration," *Proc. IEEE Visualization '99,* pp. 81-88, 1999.
[86] M. McGrath, J. Blythe, and D. Krackhardt, "The Effect of Spatial Arrangement on Judgments and Errors in Interpreting Graphs," *Social Networks,* vol. 19, no. 3, pp. 223-242, 1997.
[87] G. Melançon and I. Herman, "Circular Drawings of Rooted Trees," *Reports of the Centre for Math. and Computer Sciences,* report number INS–9817, available at: http://www.cwi.nl/InfoVisu/papers/circular.pdf, 1998.
[88] K. Mehlhorn and P. Mutzel, "On the Embedding Phase of the Hopcroft and Tarjan Planarity Testing Algorithm," *Algorithmica,* vol. 16, pp. 233-242, 1996.
[89] B. Mirkin, *Mathematical Classification and Clustering.* Kluwer Academic, 1996.
[90] K. Misue, P. Eades, W. Lai, and K. Sugiyama, "Layout Adjustment and the Mental Map," *J. Visual Languages and Computing,* vol. 6, pp. 183-210, 1995.
[91] S. Mukherjea, J.D. Foley, and S. Hudson, "Visualizing Complex Hypermedia Networks through Multiple Hierarchical Views," *Human Factors in Computing Systems, CHI '95 Conf. Proc.,* pp. 331-337, 1995.
[92] T. Munzner and P. Burchard, "Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space," *Proc. VRML '95 Symp.,* 1995.
[93] T. Munzner, "H3: Laying Out Large Directed Graphs in 3D Hyperbolic Space," *Proc. 1997 IEEE Symp. Information Visualization (InfoViz '97),* pp. 2-10, 1997.
[94] T. Munzner, "Drawing Large Graphs with H3Viewer and Site Manager," *Proc. Symp. Graph Drawing GD '98,* pp. 384-393, 1998.
[95] T. Munzner, F. Guimbretière, and G. Robertson, "Constellation: A Visualization Tool for Linguistic Queries from MindNet," *Proc. IEEE Symp. Information, InfoVis '99,* pp. 132-135, 1999.
[96] P. Mutzel, C. Gutwengwer, R. Brockenauer, S. Fialko, G. Klau, M. Kruger, T. Ziegler, S. Naher, D. Alberts, D. Ambras, G. Koch, M. Junger, C. Bucheim, and S. Leipert, "A Library of Algorithms for Graph Drawing," *Proc. Symp. Graph Drawing GD '97 Symp.,* pp. 456-457, 1998.
[97] T. Munzner, E. Hoffman, K. Claffy, and B. Fenner, "Visualizing the Global Topology of the MBone," *Proc. IEEE Symp. Information Visualization,* 1996.
[98] L. Nigay and F. Vernier, "Design Method of Interaction Techniques for Large Information Space," *Proc. Advanced Visual Interfaces (AVI '98),* 1998.
[99] S. North, "Incremental Layout in DynaDAG," *Proc. Symp. Graph Drawing GD '95,* pp. 409-418, 1995.
[100] H.C. Purchase, "Which Aesthetic Has the Greatest Effect on Human Understanding?" *Proc. Symp. Graph Drawing GD '97,* pp. 248-261, 1998.
[101] H.C. Purchase, R.F. Cohen, and M. James, "Validating Graph Drawing Aesthetics," *Proc. Symp. Graph Drawing GD '95,* pp. 435-446, 1995.

[102] H.C. Purchase, R.F. Cohen, and M. James, "An Experimental Study of the Basis for Graph Drawing Algorithms," *ACM J. Experimental Algorithmics,* vol. 2, no. 4, 1997.
[103] E.M. Reingold and J.S. Tilford, "Tidier Drawing of Trees," *IEEE Trans. Software Eng.,* vol. 7, no. 2, pp. 223-228, 1981.
[104] J. Rekimoto and M. Green, "The Information Cube: Using Transparency in 3D Information Visualization," *Proc. Third Ann. Workshop Information Technologies & Systems (WITS '93),* 1993.
[105] J.S. Risch, D.B. Rex, S.T. Dowson, T.B. Walters, R.A. May, and B.D. Moon, "The STARLIGHT Information Visualization System," *Proc. IEEE Conf. Information Visualization,* pp. 42-49, 1997.
[106] G.G. Robertson, J.D. Mackinlay, and S.K. Card, "Cone Trees: Animated 3D Visualizations of Hierarchical Information," *Human Factors in Computing Systems, CHI '91 Conf. Proc.,* pp. 189-194, 1991.
[107] G.G. Robertson, S.K. Card, and J.D. Mackinlay, "Information Visualization Using 3D Interactive Animation," *Comm. ACM,* vol. 36, no. 4, pp. 57-71, 1993.
[108] A. Robinson, *EBI Hyperbolic Viewer.* European Bioinformatics Inst., available at: http://industry.ebi.ac.uk/~alan/components, 1998.
[109] T. Roxborough and A. Sen, "Graph Clustering Using Multiway Ratio Cut," *Proc. Symp. Graph Drawing GD '97,* pp. 291-296, 1998.
[110] M. Sarkar and M.H. Brown, "Graphical Fish-Eye Views of Graphs," *Human Factors in Computing Systems, CHI '92 Conf. Proc.,* pp. 83-91, 1992.
[111] M. Sarkar and M.H. Brown, "Graphical Fisheye Views," *Comm. ACM,* vol. 37, no. 12, pp. 73-84, 1994.
[112] D. Schaffer, Z. Zuo, S. Greenberg, L. Bartram, J. Dill, S. Dubs, and M. Roseman, "Navigating Hierarchically Clustered Networks through Fisheye and Full-Zoom Methods," *ACM Trans. Computer-Human Interaction,* vol. 3, no. 2, pp. 162-188, 1996.
[113] Y. Shiloach, "Arrangements of Planar Graphs on the Planar Lattices," PhD thesis, Weizmann Inst. of Science, Rehovot, Israel, 1976.
[114] H. Shiozawa, K.-i. Okada, and Y. Matsushita, "3D Interactive Visualization for Inter-Cell Dependencies of Spreadsheets," *Proc. IEEE Symp. Information Visualization (InfoViz '99),* pp. 79-82, 1999.
[115] G. Sindre, B. Gulla, and H.G. Jokstad, "Onion Graphs: Aesthetics and Layout," *Proc. IEEE/CS Symp. Visual Languages (VL '93),* pp. 287-291, 1993.
[116] T.C. Sprenger, M. Gross, D. Bielser, and T. Strasser, "IVORY—An Object-Oriented Framework for Physics-Based Information Visualization in Java," *Proc. IEEE Symp. Information Visualization (InfoViz '98),* 1998.
[117] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for Visual Understanding of Hierarchical Systems Structures," *IEEE Trans. Systems, Man, and Cybernetics,* vol. 11, no. 2, pp. 109-125, 1989.
[118] K. Sugiyama and K. Misue, "Visualization of Structural Information: Automatic Drawing of Compound Digraphs," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 21, no. 4, pp. 876-892, 1991.
[119] W. Tutte, "How to Draw a Graph," *Proc. London Math. Soc.,* vol. 3, no. 13, pp. 743-768, 1963.
[120] X.G. Viennot, "Trees Everywhere," *Proc. 15th CAAP Conf.,* pp. 18-41, 1990.
[121] J.Q. Walker II, "A Node-Positioning Algorithm for General Trees," *Software—Practice and Experience,* vol. 20, no. 7, pp. 685-705, 1990.
[122] C. Ware and G. Franck, "Evaluation of Stereo and Motion Cues for Visualising Information in Three Dimensions," *ACM Trans. Graphics,* vol. 15, no. 2, pp. 121-140, 1996.
[123] C. Ware, *Information Visualization: Perception for Design.* Morgan Kaufmann, 2000.
[124] Y.C. Wei and C.K. Cheng, "Ratio Cut Partitioning for Hierarchical Designs," *IEEE Trans. Computer-Aided Design,* vol. 10, no. 7, pp. 911-921, 1991.
[125] J.J. van Wijk and H. van de Wetering, "Cushion Treemaps: Visualization of Hierarchical Information," *Proc. IEEE Symp. Information Visualization (InfoViz '99),* pp. 73-78, 1999.
[126] G.J. Wills, "Niche Works—Interactive Visualization of Very Large Graphs," *Proc. Symp. Graph Drawing GD '97,* pp. 403-415, 1998.
[127] R.M. Wilson and R.D. Bergeron, "Dynamic Hierarchy Specification and Visualization," *Proc. IEEE Symp. Information Visualization (InfoViz '99),* pp. 65-72, 1999.
[128] P. Young, "Three Dimensional Information Visualization (Survey)," Computer Science Technical Report, Centre for Software Maintenance Dept. of Computer Science, Univ. of Durham, available at: http://www.dur.ac.uk/~dcs3py/pages/work/documents/lit-survey/IV-Survey/index.html, 1996.
[129] J.-D. Zapata-Rivera, E. Neufeld, and J.E. Greer, "Visualization of Bayesian Belief Networks," *Proc. IEEE Visualization '99, Late Breaking Hot Topics,* pp. 85-88, 1999.
[130] R. Zeiliger, "Supporting Constructive Navigation of Web Space," *Proc. Workshop Personalized and Solid Navigation in Information Space,* 1998.

Ivan Herman graduated as applied mathematician in 1979 in Budapest, Hungary, and received his PhD at the University of Leiden, The Netherlands, in 1990. He is currently a senior researcher at the Centre for Mathematics and Computer Science (CWI) in Amsterdam and is head of the research group on information visualization. He has been chief designer and implementor of several graphics and multimedia systems, and is also author or coauthor of close to 50 scientific publications in international journals and conferences. He is currently cochair of the Ninth World Wide Web conference and of the second joint Eurographics/IEEE TSVG Symposium on Visualization. He has been a member of the Eurographics Executive Committee since 1987 and a member of its Executive Board since 1990. He is also member of the IEEE Computer Society and of the Advisory Committee of the World Wide Web Consortium.

Guy Melançon received his PhD in mathematics from the University of Québec in Montréal, Canada, in 1991 and recently defended his "habilitation" in computer science at the University of Bordeaux I, France. He is currently a scientific researcher at the Centre for Mathematics and Computer Science (CWI) in Amsterdam and also holds a permanent position at the University of Bordeaux I, France. He is the author or coauthor of many scientific publications in international journals and conferences in combinatorial mathematics and information visualization. He is currently coorganizer of the second joint Eurographics/IEEE TSVG Symposium on Visualization.

M. Scott Marshall received a BA in computer science from the University of California at Berkeley in 1992. He is currently a research associate at the Centre for Mathematics and Computer Science (CWI) in Amsterdam, working in the information visualization research group. He recently helped to implement an ISO standard for multimedia called PREMO and coauthored a book on the subject. His research interests include scientific, medical, and information visualization and knowledge representation. He is currently working on his PhD dissertation on graph visualization in cooperation with the University of Bordeaux, France.