Graph Visualisation and Navigation in Information Visualisation

I. Herman, G. Melançon, M. S. Marshall

Centre for Mathematics and Computer Sciences (CWI)
Kruislaan 413
1098 SJ, Amsterdam, The Netherlands
{I.Herman, G.Melancon, M.S.Marshall}@cwi.nl

Abstract
This is a survey on graph visualisation and navigation techniques, as used in information visualisation. Graphs appear in numerous applications, like web browsing, state–transition diagrams, computer data structures, etc. The ability to visualise and to navigate in these potentially very large, abstract graphs is often a crucial part of an application. Information visualisation has specific requirements, which means that this survey approaches the results of traditional graph drawing from a different perspective than the traditional surveys; as such it is a useful complementary survey to those.

Keywords: information visualisation, graph visualisation, graph drawing, navigation, focus+context, fish–eye, clustering.
1998 Computing Reviews Classification System: G.2.2., H.3.3, H.4.m, H.m, I.3.4, I.3.m, J.m

1. Introduction

Although the visualisation of graphs is the subject of this survey, it is *not* about graph drawing in general. Excellent bibliographic surveys[3], books[4], or even on–line tutorials[20] exist for graph drawing. Instead, the handling of graphs is considered with respect to *information visualisation*. To understand why graphs play a special and important role in this area, we will briefly characterize information visualisation before going into the technical details of our subject.

1.1 Information visualisation

Information visualisation can be defined as the visualisation and navigation of abstract data structures. Although many fundamental papers on information visualisation appeared more than 10 years ago, it is only in the past few years that the field began to emerge as a separate discipline.

The distinction between scientific visualization and information visualisation is more a historical artifact than a logical distinction. Judging purely from their names, one could reasonably expect scientific visualisation to be a subfield of information visualisation. However, scientific visualisation was developed much earlier in response to the need of scientists and engineers to view data in graphical format. As the graphical display of data evolved, applications were developed which no longer fit neatly into the category. Despite the artificiality of the distinction, it is informative to distinguish information visualisation from scientific visualisation. There are three essential differences between the two areas:

1) Scientific visualisation is usually closely related to mathematical structures and models. In contrast, information visualisation looks at abstract data structures such as computer program structures, database query results, hypermedia structures, or organisational charts. The major difference is that the data visualised by scientific visualisation systems often *have an inherent geometry* (e.g., the airflow around the wing of an aeroplane) which is not the case for most of the data handled by information visualisation systems. As a consequence, the geometric representation for the data has to be “invented” by implementors in information visualisation[†].
2) Human–computer interaction issues play a central role in information visualisation. It is not a coincidence that many research results are published, for example, at the yearly ACM SIGCHI conferences. The reason is that users of scientific visualisation are usually experts, whereas users of information visualisation can be people at all levels of expertise.

[†]A noteworthy exception is the visualisation of a physical network, where each node corresponds to a physical location[5].

3) Finally, the requirements of the target computing platforms are usually different. Scientific visualisation applications typically run on high–end computing systems such as graphics workstations, supercomputers, and CAVE systems. This is in sharp contrast to information visualisation systems which are more likely to run on "everyday" computers.

Of course, it is not our intention to present information visualisation as "conflicting" with scientific visualisation. Actually, much more relates these fields than separates them, such as the use of visual paradigms, the strong emphasis on interaction, their user–centric nature, to name just a few common aspects. We regard it as a very healthy development that, in the past few years, both IEEE's Visualization'XX conference and the Eurographics Visualisation workshop have adopted information visualisation as a separate and explicit track, alongside scientific visualisation and volume visualisation.

1.2 Graphs in Information Visualisation

Information visualisation has become a large field and "sub–fields" are beginning to emerge (see for example Card *et al.*[12] for a recent collection of papers from the last decade). A simple way to determine the applicability of graph visualisation is to consider the following question: *is there an inherent relation among the data elements to be visualised*? If the answer to the question is "no", than data elements are "unstructured" and the goal of the information visualisation 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 visualisation research dealing with unstructured data has a distinct flavour. 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 visualisation has specific requirements, which means that we will approach the results of traditional graph drawing from a different perspective than the traditional surveys[3,4,29]. We hope that this approach will make the overview useful and interesting.

1.3 Typical Application Areas

Graph visualisation 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 visualisation: "Where am I?" "Where is the file that I'm looking for?" Other familiar types of graphs include the hierarchy illustrated in an organisational chart and taxonomies which 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 as evolutionary 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.

1.4 Key Issues in Graph Visualisation

The size of the graph to view is a key issue in graph visualisation. 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 Figure 1). 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 of closure, papers in information visualisation rarely apply cognitive science 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. For this reason, an objective evaluation of the merits of a given approach is extremely difficult.

The rest of this survey is organised as follows: In Section 2, we try to give an impression of graph layout issues and limitations with regard to scaleability. Then, we discuss several approaches to navigation of large graphs (Section 3), followed by methods of reducing visual complexity through reorganisation of the data (Section 4). Afterwards, we discuss a few application systems which 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 which we found particularly important for graph visualisation (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 visualisation in information visualisation. As we shall see, this point of view differs, in many respects, from the traditional view of the Graph Drawing community. We'll give an account of the available results and discuss their rele-

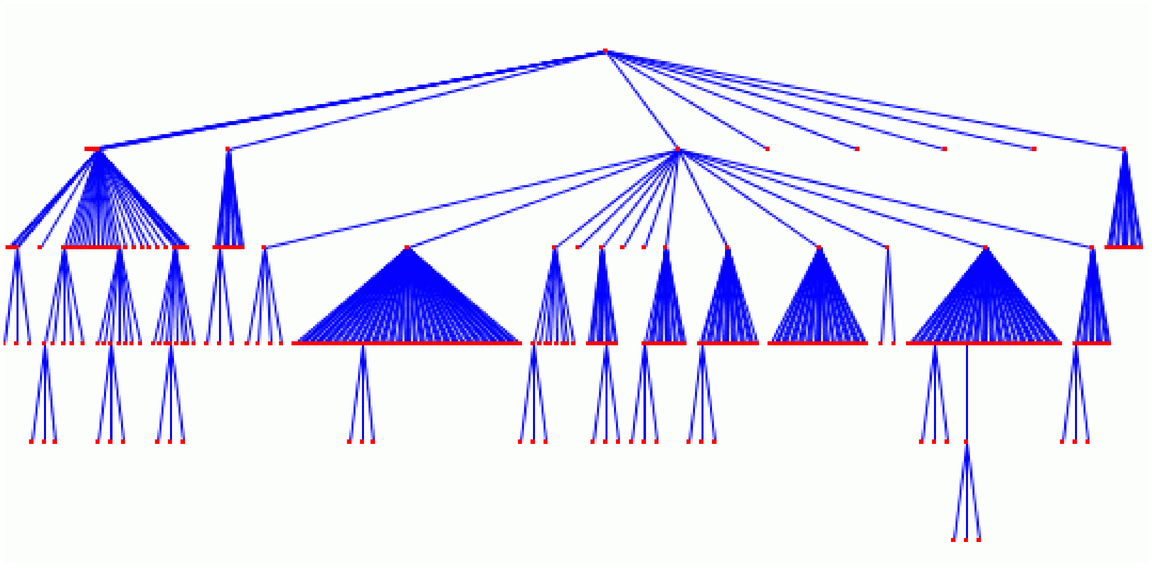

Figure 1: *Tree layout for a moderately large graph*

vance for graph visualisation, although, in general, we will not go too far into the technical details. For those desiring more information, we recommend the excellent book of Battista *et al.*[4] as one of the best starting points.

2.1 Background of Graph Drawing

The Graph Drawing community[†] 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[65]). The annotated bibliography by Battista *et al.*[3] gathers hundreds of papers studying what a *good* drawing of a graph is. And 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 intersecting edges. Layout algorithms may be categorised 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, non–deterministic approaches form a category which 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 sub–structures should be displayed in the same manner, edge–crossings should be kept to a minimum, etc.[‡] 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.*[4] for further examples.

The Reingold and Tilford algorithm for trees[89,106] (see Figure 1) is a good example of a layout algorithm achieving these aesthetics 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, minimisation of the full graph area might be an important criterion in applications. Although some of the rules

[†]http://www.cs.brown.edu/people/rt/gd.html

[‡]Actually, some aesthetics are quite arbitrary and are not seen as absolute rules any more[87,88].

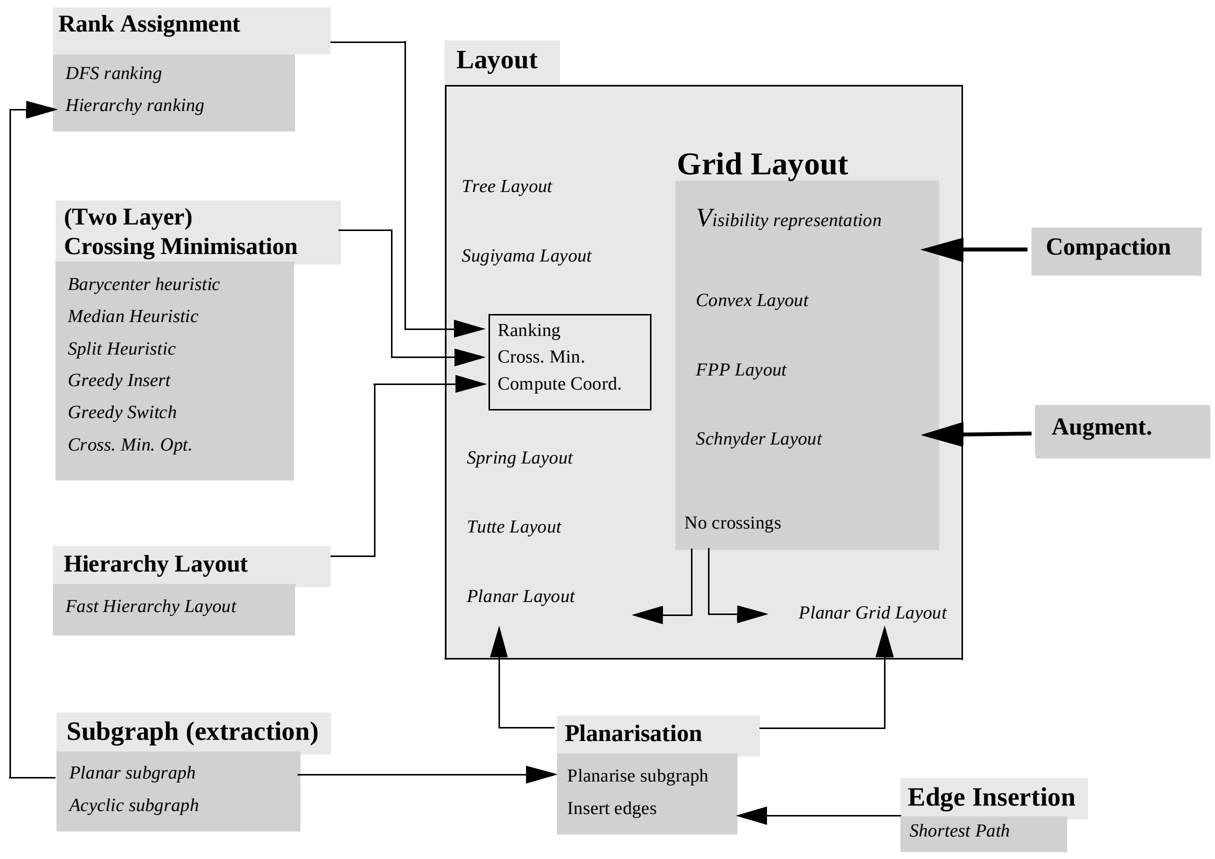

Figure 2: *Overview of graph layout algorithms (Reproduced from Mutzel et al.[84]. Courtesy of T. Mutzel, Max–Planck–Institut Saarbrücken, Germany).*

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 optimisation problem, used in a number of non–deterministic layout algorithms[22].

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[87] demonstrates that “reducing the crossings is by far the most important aesthetic, while minimising the number of bends and maximizing symmetry have a lesser effect”. Her work concludes by prioritising these aesthetics; see also Purchase *et al.*[88] for more details. Other authors[7,23] report differences in the perception of a graph depending on its layout. Unfortunately, usability studies necessitate a great effort, both to realise the experimentation itself and to analyse its results properly; but we regard this line of work as essential for information visualisation, too. They have recently gained credibility in the graph visualisation community as well, recognising their contribution to help focus on important issues in the area.

A wide variety of tasks related to graph drawing have been studied: layering a graph, turning it into an acyclic directed graph, planarisation of a graph, minimising the area occupied by a layout, minimising 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[4,29].

In graph visualisation, 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[109] or H3Viewer[83] 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. Indeed, 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, Figure 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 very 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 give more details on these techniques. However, beyond a certain limit, no algorithm will guarantee a proper layout of

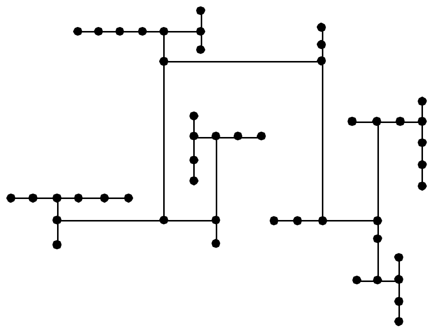

Figure 3: *H–tree layout*

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 visualisation process is often to cut down the size of the graph to display. As a result, classical layout algorithms still remain usable tools for visualisation, 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[55,86]. 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[79]. Predictability is very often ignored during analysis of classical layout algorithms, which are often only 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. 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.

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. Figure 2 gives a classification of existing layout techniques. This classification is the work of Mutzel *et al.*[84]; most of the algorithms are described in the book of Battista *et al.*[4]. We will concentrate 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[89,106] is probably the best known layout technique in the tree layout category (see Figure 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[98] which only perform well on balanced trees. Eades[30] gives a variation of the algorithm that behaves well in general (see Figure 3). The radial positioning by Eades[30] places nodes on concentric circles according to their depth in the tree (see Figure 4). A subtree is then laid out over a sector

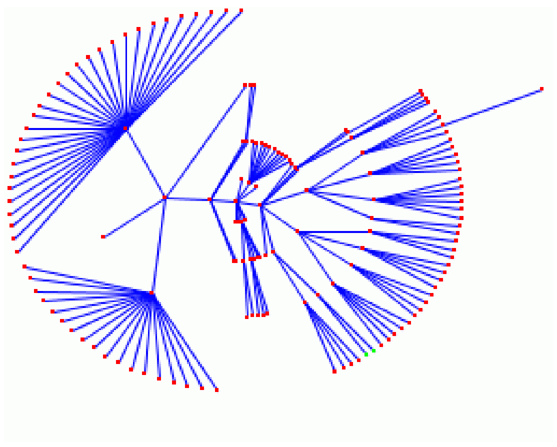

Figure 4: *Radial view*

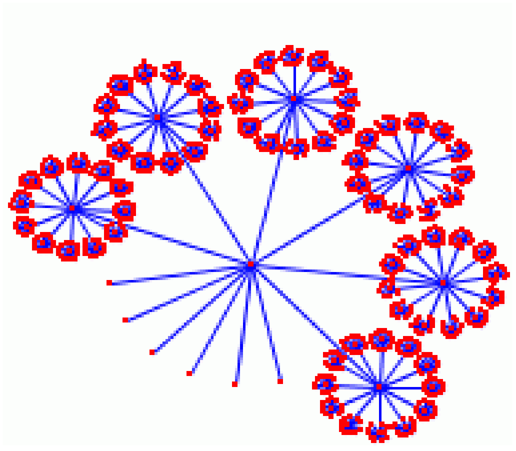

Figure 5: *Balloon view*

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[57,109]). The cone tree[16,92] algorithm can be used to obtain a "balloon view" of the tree by projecting it onto the plane[16,62], where sibling subtrees are included in circles attached to the father node. It is also possible to compute the nodes' position directly, without reference to cone trees[76] (see Figure 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 Figure 2, it contains a variety of layouts. Chapter 3.1 of the book by Battista *et al.*[4] 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" ar-

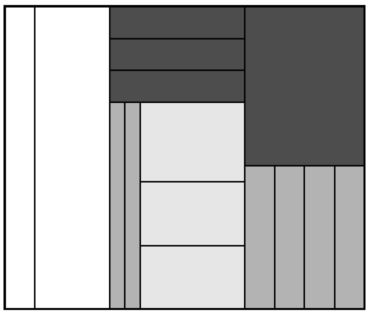

Figure 6: *Tree–maps; rectangles with identical shading belong to the same level of a (tree) hierarchy. (Adapted from Johnson and Schneiderman[63])*

senal, are also worth mentioning here: tree–maps[63] (see Figure 6), and onion graphs[99], which represent trees by sequences of nested boxes.

A separate box at the bottom of Figure 2 is devoted to *Planarity*. This is a critical issue in graph drawing, because 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[58] (see Chapter 3.3 in Battista *et al.*[4]; see also Mehlhorn and Mutzel[77] 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[48]; see also Chapter 6 in Battista *et al.*[4]).

In the case of information visualisation it makes sense to test for planarity only if there are very good reasons to believe that a graph might be planar. Indeed, a planar graph with N nodes has a linear number of edges[2,24]. It does make sense, however, to check for planarity when dealing with a small and sparse graph, such as a subgraph obtained by clustering a larger graph (see Section 4). In general, however, we can safely say that planarity is not a central issue for graph visualisation in information visualisation.

The *Sugiyama Layout* box included in Figure 2 is named after the seminal work by Sugiyama on layout for general directed graphs[101]. The basic approach for laying out a directed graph is to first decide on a *layering* of its nodes; that is, assign to each node a layer number 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, that is a subgraph containing all nodes of the original graph, but such that once nodes are placed on their respective layers, edges will all point in the same direction (usually downwards). Another solution is not to extract a subgraph but turn the original graph into an acyclic one by reverting the direction for 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 minimisation[4,29] since the crossing of edges has been recognized as a major obstacle to the readability of graphs[87,88]. This is usually done by minimising the number of edge–crossings between two consecutive layers. This minimisation step is the really complex core of the whole algorithm. Note that these strategies do not address the problem of minimising the number of crossings in the whole graph: even with the restriction of looking at consecutive layers only, minimisation of edge–crossings is difficult and complex. In fact, Garey and Johnson proved the problem to be NP–hard[47] and Eades and Whitesides proved the corresponding decision problem to be NP–complete[31].

The complexity of a proper minimisation has motivated the development of various heuristics for computing a good order for the nodes on a layer. Tutte[103] was the first to propose a heuristics: 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 neighbours. This corresponds to the intuitive idea that a node should be kept "close" to its neighbours. 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 pre–processing possibilities for the algorithm in the *Sugiyama Layout* category. New improvements and perspectives to the problem were published recently[64,70], which include a detailed report on existing techniques[71], and a comparison of existing heuristics[72].

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 come back 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 minimisation in Chapter 9 of their book[4].

The *Spring Layout* box stands for all non–deterministic layout techniques, also called *Force–Directed Methods*. Eades[28] was the first to propose this approach in graph drawing, modelling 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[22,41,43,66]. Mathematically, the methods are based on an optimisation 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 minimisation without any supplementary efforts[41].

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[41] 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 visualisation, since unpredictability can be a major problem for interaction. However, in some cases, the lack of predictability can be compensated if the graph is small or sparse, by animating changes in the layout to help the user in adapting to the new drawing[60]. For further information on force–directed methods, the reader should refer to the comparison of non–deterministic techniques of Brandenburg *et al.*[9] or Chapter 10 in the book of Battista *et al.*[4].

We shall not discuss layouts on grids. We refer the reader to Battista *et al.*[4] 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 Figure 2. None of these techniques play a central role in graph visualisation.

2.3 Spanning Trees

A general problem with the majority of the available techniques is that they are only applicable for relatively small graphs[†]. The "traditional" concerns of Graph Drawing become much less relevant in graph visualisation, which typically deals with very large graphs. In general, it makes no sense to test a graph of several hundreds of 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[16,30,89,106] 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 that of 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 Jungnickel[65]). 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 visualisation, 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 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[8]. More sophisticated algorithms have been designed to satisfy various optimisation goals. If a weight function exists for the graph, algorithms exist to compute spanning trees minimising (or maximising) the total weight of the tree. One solution is to iteratively build a tree by adding edges adjacent

[†]This is clearly shown by the size of the graphs submitted each year to the so–called Graph Drawing Contest, although bigger graphs — and also graphs coming from real–life situations — have also been included in recent years.

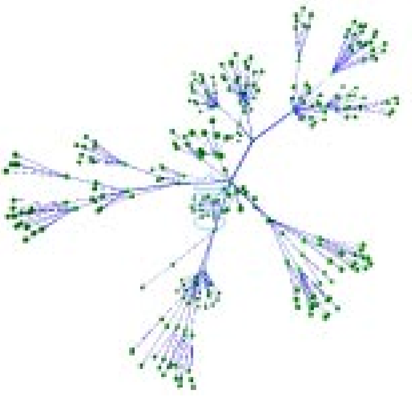

Figure 7: *3D version of a radial algorithm (Courtesy of S. Benford, University of Nottingham, UK)*

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 it will also affect the complexity of the extracting process (see, for example, Chapters 4 and 5 of Jungnickel[65]). 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 realising an optimisation 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 minimisation. 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.*[57]).

2.4 3D Layout

One popular technique is to display graphs in 3D instead of 2D. The hope is that the extra dimension would give, literally, more "space", and that this would ease the problem of displaying large structures. Furthermore, the user can navigate to find a view without occlusions. The simplest approach is to generalise classical 2D layout algorithms for 3D. Figure 7, for example, shows a 3D version of a radial tree algorithm, while Figure 8 is a generalisation[90] of the 2 dimensional approach using nested boxes[99]. Most force–directed methods are also described in dimension independent terms which allows them to be generalised to 3D (such as the approaches based on simulated annealing by Davidson and Harel[22] and also from Cruz and Twarog[21]). The reader may find further examples in the overview by Young[110].

In spite of their apparent simplicity, Figures 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[33]. As a consequence, virtually all 3D displays of graphs include addi-

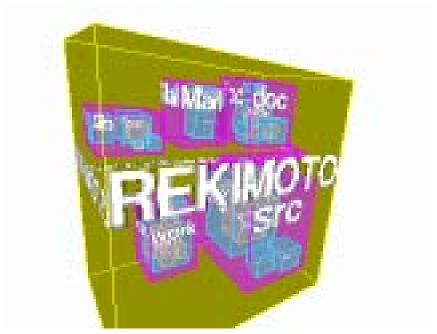

Figure 8: *Information cube.*
(Courtesy of J. Rekimoto, Sony Computer Science Laboratory, Inc., Japan[90])

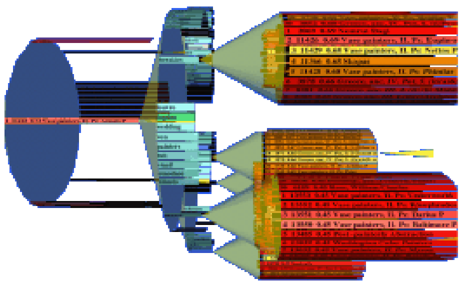

Figure 9: *A Cone Tree.*
(courtesy of M. Hemmje, GMD, Germany[52])

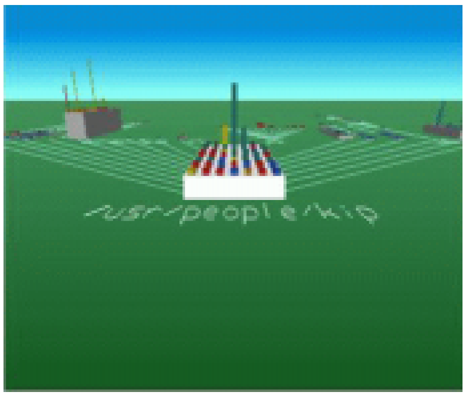

Figure 10: *The File System Navigator.*

tional 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 minimisation of edge–crossings is 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[92,93] (see Figure 9) is one of the best known 3D graph (in this case, tree) layout techniques in graph visualisation[†]. In contrast to the previous examples, cone trees have been developed directly for 3D, instead of generalising another 2D algorithm. Mathematically, the layout is quite simple. Nodes are 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 the "room", i.e., the box containing the full cone tree. The original idea of cone trees has been re–implemented by others[16,52,62] with, in some cases, a somewhat refined layout algorithm. Carrière and Kazman[16], 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 a better use of the available space. Jeong and Pang[62] replace the cones with discs, thereby reducing the occluded regions in space.

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

The interactive and visual aspects of cone trees are absolutely essential to make them really 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 the familiarity with the "real" human environment, 3D lends itself to the creation of real–world metaphors which should help in perceiving complex structures. One of the earliest examples is the File System Navigator (see Figure 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 very 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. More complex 3D metaphors include the Perspective Wall[93], which represents the data as posters on a big wall in virtual space. VizNet[38] and Vitesse[85] 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[11] displays an animated book in 3D with Web page contents, etc. Here again, we refer the reader to the overview of Young[110] for further examples.

In spite of all the technical developments in the area, and their undeniably attractive features, 3D graph visualisation techniques have significant difficulties. In our view, the main reason lies in the inherent cognitive difficulties with navigating in 3D, such as the discrepancy of using 2D screens and 2D input devices to interact in a 3D world, the missing motion and stereo cues (see the overview of Ware

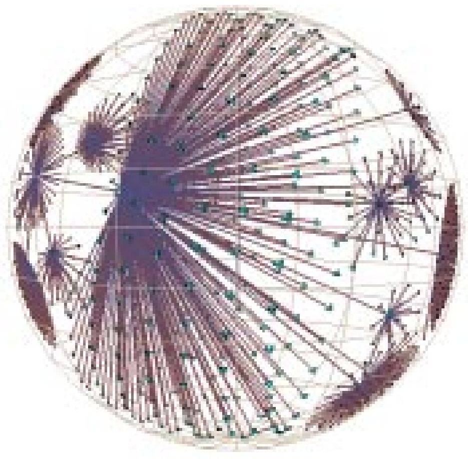

Figure 11: *Hyperbolic view of a tree in 3D. (Courtesy of T. Munzner, Stanford University, USA[82])*

and Franck[107] on how important these cues are), etc. If advanced VR–like systems such as a workbench or a CAVE are used, some of these difficulties may be solved. However, these facilities are not really widespread and are much too expensive to serve as a basis for most of the information visualisation applications. Of course, 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 visualisation.

2.5 Hyperbolic Layout

The hyperbolic layout of graphs (mainly trees) is one of the new forms of graph layouts which has been developed with graph visualisation and interaction in mind. The first papers in this area are from Lamping *et al.*[73,74], followed by a series of papers by Munzner[81,82,83]. Both developed, for example, Web content viewers based on these techniques. Hyperbolic views, which can be implemented in either 2D or 3D, provide a distorted view of a tree (see Figure 11). It resembles the effect of using fish–eye lenses on traditional tree layouts. This distorted view makes it possible to interact with potentially very large trees, making it suitable for real–life applications. We will come back to this distortion effect later in this survey (see Section 3.2), when we will concentrate on navigation rather than pure layout.

Hyperbolic views represent a radically different direction in layout, when compared to the various algorithms described so far, due to their different geometrical background. In fact, some of the classical layout algorithms can be re–used 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 very few people in this community really understand the mathematics of hyperbolic visualisation, and it is also quite difficult to reproduce the results. Unfortunately, none of the papers are didactic enough to reveal this mystery. We will expound more on the main elements of these layout methods, with the hope that the reader will gain a better understanding and appreciation of the technique.

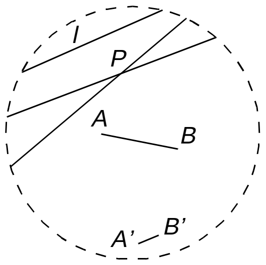

Figure 12: *Klein model for the hyperbolic plane. The line segments AB and $A'B'$ have an equal length in the hyperbolic sense.*

Hyperbolic geometry is based on an axiomatic system almost identical to the traditional Euclidean axioms with the exception of one, the so–called 5th 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., non–intersecting and co–planar), 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 very different in flavour: the traditional trigonometric equations are no longer valid, the sum of the internal angles of a triangle is no longer 180 degrees, etc.[†] (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 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 Figure 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 non–trivial work. The validity of the negation of Euclid’s 5th 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: they perform a layout algorithm in the hyperbolic plane or space, and then display

[†]The interested reader might want to refer to Coxeter[19] for further details. Also, look at the papers of Gunn[49] or Hausman *et al.*[50].

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 representations in Euclidean geometry. The original paper of Lamping *et al.* used the Poincaré model, whereas Munzner mostly uses the Klein model. In Figure 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 metric nature of hyperbolic geometry makes some very simple layout algorithms suddenly viable. As an example, consider the outline of the following tree placement algorithm (see Figure 13)[†]. The algorithm starts

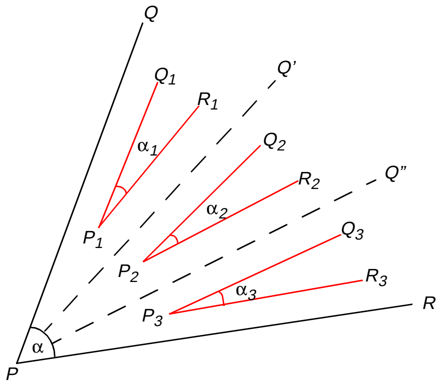

Figure 13: *A simple tree drawing algorithm on the Euclidean plane*

from the root of the tree, positioning the sub–trees recursively in a circular fashion. In each step, the algorithm determines a wedge to place a sub–tree; the goal is to find wedges in such a way that no crossing would occur between edges of different sub–trees. If the point P on the figure refers to a node, and the wedge QPR with angle α is the one assigned to the sub–tree starting at P, the main step of the algorithm is to define sub–wedges for the sub–trees of P (starting at P_1, P_2, and P_3). The angle α is divided into (for the sake of simplicity, equal) sub–angles, one for each sub–tree. 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 sub–wedges at some suitable distance from P. The next step is to determine the constraining wedges for these sub–tree; 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 sub–trees. 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 sub–tree. However, if the same algorithm is used on a hyperbolic plane, the situation is quite different. Figure 14 shows the same algorithm in the

[†]This algorithm is essentially the same as the one used in the paper of Lamping *et al.*[74].

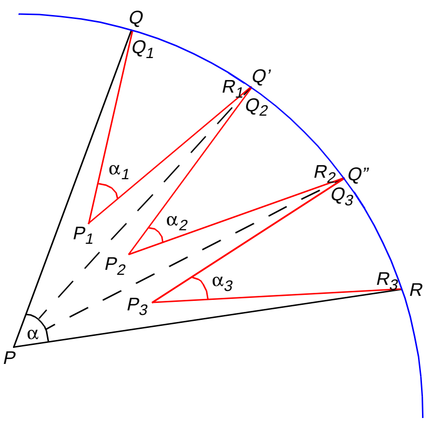

Figure 14: *The tree drawing algorithm on the hyperbolic plane, using the Klein model to visualize the results.*

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 *et al.*[74], "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, but this can be done. 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[82] which is, in fact, a generalisation 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 visualisation. No layout algorithm alone can overcome the problems raised by the large sizes of the graphs occurring in the visualisation applications. Furthermore, the task of revealing the *structure* of the graph calls for innovative approaches, too.

3.1 Zoom and Pan

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

Zooming can have, in fact, two forms. Geometric zooming just provides a blow up of the graph content; *semantic zooming* means that the information content changes and more details are shown when getting closer to a particular area of the graph. The technical difficulty in this case is not

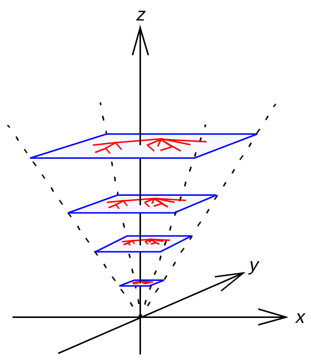

Figure 15: *A space–scale diagram (adapted from Furnas and Bederson[45]).*

with the zooming operation itself, but rather to assign 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 raise 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 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 runs away faster than the pan can keep up with. The net result is that the target is approached non–monotonously: it first moves away as the zoom dominates and only later comes back to the centre of the view, which can be quite disturbing.

This, and other, problems related to zoom and pan are, of course, not related to graphs only, nor is the elegant solution proposed by Furnas and Bederson[45] to alleviate them. Nevertheless, information visualisation systems based on graphs can greatly benefit from their approach; it is therefore worth giving a short description here. Furnas and Bederson introduce the concept of space–scale diagrams (see Figure 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, which contain information both on 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 specialised authoring system for semantic zooming[46].

3.2 Focus+Context Techniques

A well–known problem of zooming is that if one zooms on a focus, all contextual information is lost[†]. This may become a major usability obstacle; hence the importance of different exploration facilities where the user can focus on some detail *without* losing the context. 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, in some situations, focus+context techniques are indeed a good alternative, and full–blown applications systems often implement both.[‡]

3.2.1 Fisheye Distortion

Graphical fisheye views are very 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 Figure 16).

Mathematically, what happens is as follows. 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 $h(x)$ function should be concave, mapping monotonically the [0,1] interval onto [0,1] (see Figure 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 used, e.g., by Sarkar and Brown[95] is: $h(x) = (d+1)/(d+1/x)$. The factor $d \geq 1$ is a distortion factor, which can be set interactively by the user. Figure 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 Figure 16). Other variations are possible; the reader should consult, for example, the overview of Carpendale *et al.*[14] or Keahey *et al.*[68]

[†]Unless a separate window, for example, keeps the context visible, which is done by several systems. This solution is not fully satisfactory either, though.

[‡]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[44]. 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.

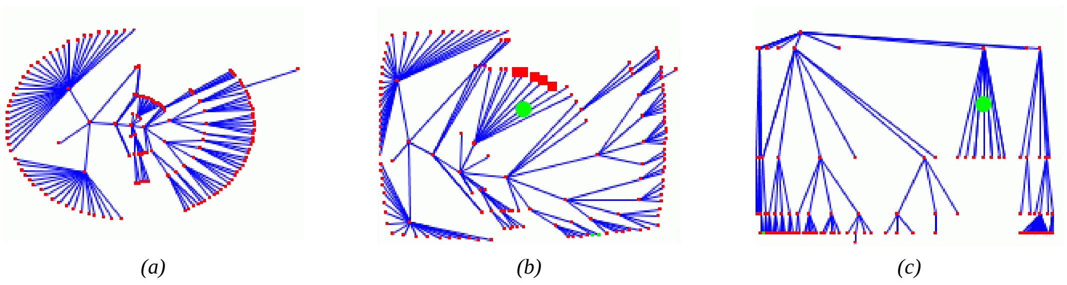

Figure 16: *Fisheye distortions. Figure (a) represents the graph without fisheye; figure (b) uses a polar fisheye, whereas figure (c) uses a cartesian fisheye with a different layout of the same graph. The dot on figures (b) and (c) are the focal points for the fisheye distortion. Note also the extra edge–crossing on figure (b).*

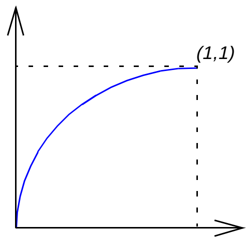

Figure 17: *Typical distortion function for fisheye*

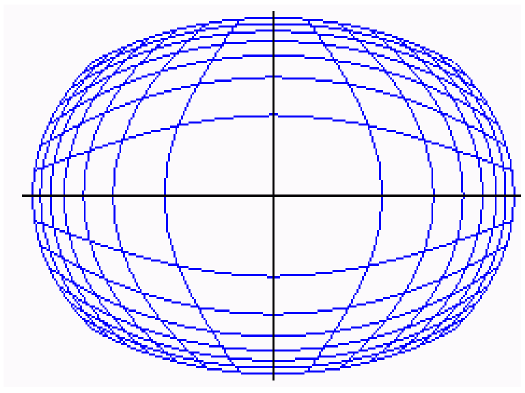

Figure 18: *Polar fisheye distortion; distortion value is 4.*

for further examples and for their visual effects. The final choice should depend on the style of the graph to be explored as well as the layout algorithm in use.

This simple, albeit very powerful technique is an important navigation complementary to zoom and pan. It has one drawback, though, which implementors should be aware of. The essence of a fisheye view is to distort the position of each node. If the distortion were to be applied faithfully, the edges connecting the nodes should be distorted, too. Mathematically, the result of this distortion is a general curve. Usual graphics systems (e.g., X11, Java2D, OpenGL) do not offer the necessary facilities to transform lines into these curves easily (they can be, mathematically, fairly complex). The implementer's only choice is, therefore, to approximate the original line segments with a high number of points, transform 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 sought (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 Figure 16/b). This is one of those typical situations when the pragmatism required by information visualisation 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 organisation 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. 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 to interact 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 Figure 11). The equivalent of the focal point of the graphical fisheye view is the centre 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 centre point within the graph.

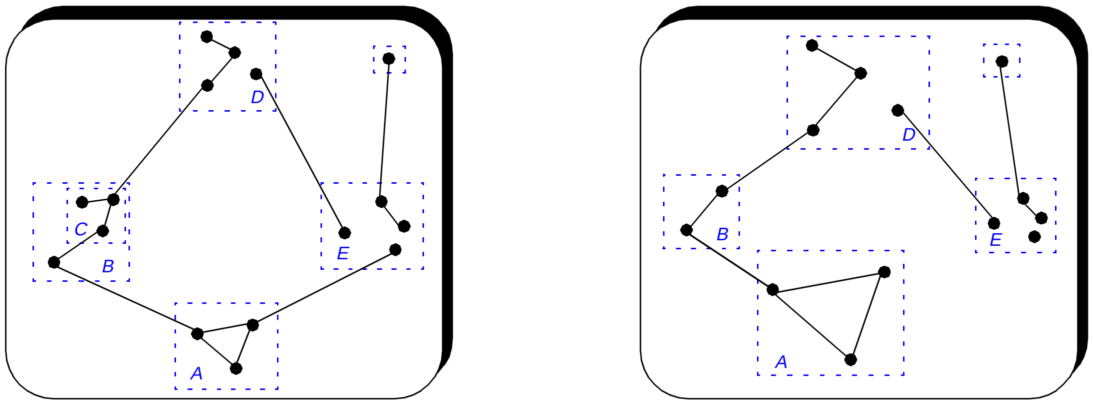

Figure 19: *Multifocal fisheye/zoom in a hierarchically clustered graph. The dotted rectangles denote the (logical) clusters. Note the disappearance of the cluster C on the right hand side.*

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[85], for example, the user has only limited navigation facilities in 3D; the main goal of mapping objects onto a sphere or an ellipsoid is indeed to achieve a focus+context distortion. More complex surfaces (like 3D surfaces of blended Gaussian curves) have also been used, for example, by Carpendale *et al.*[13,14], to achieve focus+context effects. Other 3D visualisation techniques, already cited in Section 2.4 (like the Perspective Wall[93]), should be mentioned in this context, too.

The hyperbolic layout is special because it is a graph layout algorithm which 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, Figure 5 shows a very balanced view of a tree, using a balloon layout algorithm[76]. 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[57]. We think that such research would provide valuable input for the implementors of graph visualisation systems.

3.2.3 Further Issues in Focus+Context Techniques

There are further issues in the area focus+context which can be of interest, some of which can be the basis for future research, too (a general characterisation and taxonomy of distortion techniques is also presented in Leung and Apperly[75]). 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.)[39,68,96]. The techniques can also be extended to 3D[15]. 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[95], but a finer control over this facility might lead to new insights, too. Note that the space–scale diagrams[45] (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 should also be applied [67,68,14], allowing the user to simultaneously concentrate on several important areas of the graph or to use the system in a cooperative environment[85].

An interesting example which combines various techniques, including multifocal zoom and focus+context, is provided by Schaffer *et al.*[97]. Their system also shows the fundamental importance of clustering, which we address elsewhere in this survey (see Section 4). They consider graphs which have a hierarchical clustering already. The left hand side of Figure 19 shows the drawing of the 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 right hand side of the same figure shows the 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 mathematics of distortion and shrinking used to achieve these results. Similar ideas can also be found in the DA–TU system of Huang *et al.*[61]. However, a lot still 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 emphasised several times that the size of the graph is a major problem in graph visualisation 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

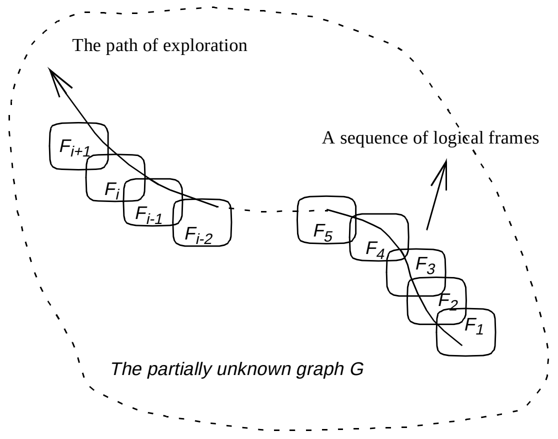

Figure 20: *Exploration of a huge graph (Reproduced from Huang et al.[59], courtesy of P. Eades, University of Newcastle, Australia)*

such 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[10,35,59,60,86,105].

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, e.g., Huang *et al.*[59]) along some trajectory (see Figure 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[105], 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.*[59] (who also implemented a tool along the same lines to explore the World Wide Web[60]) anticipate the user's future interaction by adding not only a new node to a frame, but also its immediate neighbours. Huang *et al.*[59] or North[86] 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.*[59] (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 visualisation.) North[86] and Brandes *et al.*[10] go further by providing dynamic control over the parameters which 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 in the preceding section, 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[69]. 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*[36,78]. We will refer to clustering which uses only structural information about the graph as *structure–based* clustering (also referred to as identifying *natural clusters*[94]). 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 visualisation are references to purely structure–based clustering, with a few notable exceptions[80,91]. 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 specialised 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 visualisation terms, filtering usually refers to the de–emphasis or removal of elements from the view, while search usually refers to the emphasis of an element or group of elements. Filtering and search can be accomplished by partitioning elements into two or more groups, and then emphasizing one of the groups.

By far the most common clustering approach in graph visualisation is to find clusters which are disjoint or mutually exclusive, as opposed to clusters which 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[78]. It is also known as the Ratio Cut technique in VLSI design[108].

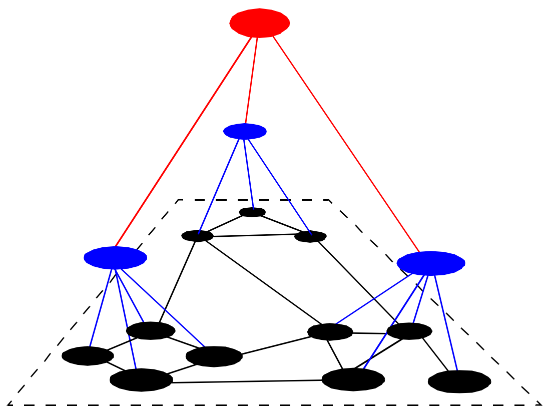

Figure 21: *A structure induced by hierarchical clustering. (Adapted from Eades and Feng[32])*

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[94]. Different results can 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. A small number of clusters allows for a fast processing and navigation. However, this number should not be too small, because otherwise the visible information content is too low.

A common approach 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[25,32,97]. Huang and Eades[61] 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 visualisation systems. One original solution is to omit the edges and position the nodes in a way which indicates their connectivity[109]. 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*[78]. 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 Figure 21). Hierarchical clustering can therefore be used to induce a hierarchy in a graph structure which 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 objects in the system exert repulsive force on the others and related objects 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[54], 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[37], 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 visualisation. In this survey, we will use the term to refer to *a measure of a graph feature which is associated with a node in the graph.* We have identified the concept of node metrics in several places in the literature[8, 55, 44, 69]. 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[55,56].

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 display only the nodes with a particular degree or higher. 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[104]) has been applied in Figure 22, in which nodes with the highest Strahler metric values generate a *skeleton* or backbone which is then emphasized (see Herman *et al.*[55,56]).

Metrics can also be composed due to their numeric nature[56]. 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 cluster, search, or skeleton. The *Degree of Interest (DOI)* function of Furnas[44] is also an example of a metric which is composed of two other metrics (in this case, a metric based on distance and a level of detail).

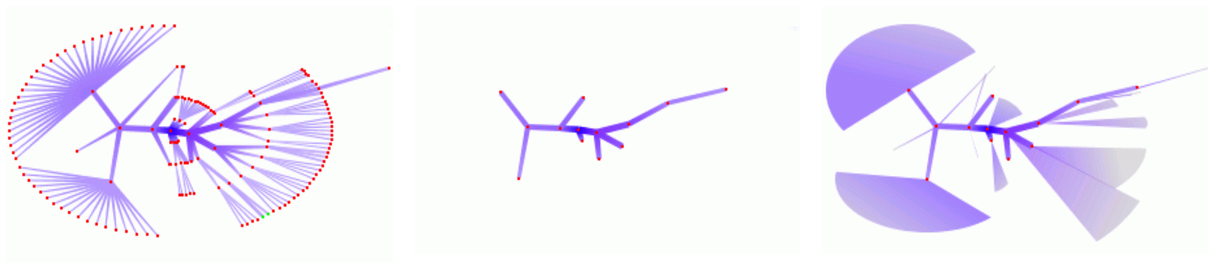
(a) (b) (c)

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

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[†].

Once a subset of nodes has been selected, as with a skeleton, a method of representing the un–selected 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[69] (see Figure 22):

- *ghosting*: de–emphasizing nodes, or relegating nodes to the background.
- *hiding*: simply not displaying the un–selected 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.*[100] in the IVORY system. Figure 22(c) 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[56]. The resulting graph, technically a compound graph, is a sort of high–level map or *schematic view*[17,56] of the original graph which is useful for navigation of the original graph.

Clustering is a difficult problem 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.*[4] 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 much more attention in future, especially from the information visualisation community.

[†]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.

5. Systems

The area of graph visualisation has reached a level of maturity in which large applications or application frameworks are being developed. However, it is extremely difficult to enumerate all the systems because of the sheer quantity. Some of them have a very short lifetime because they are research tools and others are embedded in specialised applications. A thorough classification, categorisation, and a systematic overview of all the graph visualisation 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 which 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.

SemNet[37] is one of the few systems to provide graph editing while still providing a comprehensive set of tools to visualise large graphs. It is also one of the earliest complete systems which we know about.

Clustering has been applied by many older systems such as SemNet[37], Narcissus[54], SKETCH[102], and the Navigational View Builder[80]. Some newer systems which cluster graphs are NicheWorks[109], DA–TU[61], STARLIGHT[91], and a new system used by Bell Laboratories[51] for network visualisation.

NicheWorks is an example of a system that can be adapted for very specific applications. As an example, it has been used in visualising Y2K related problems[34]. The *fsviz* system of Carrière and Kazman[16], the da Vinci system of the University of Bremen[42], or the Latour system developed at CWI[57] fall into the same category. We should also mention the company called Tom Sawyer Software[‡], which offers a number of products based on various graph drawing techniques.

There are a few systems that stand out because of unique features. The STARLIGHT[91] system performs content-based clustering and allows multiple mappings and layouts. It is one of the few systems which allows a 3D graph to be linked with locations on a plane (for associating nodes with geographical positions). Another system, SDM[18] 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 system called Web PATH[40] uses a fog effect in a 3D rendering of web history to limit the window of viewing.

[‡]http://www.tomsawyer.com.

The World Wide Web is one of the typical application areas where graph visualisation may play an important role in the future. H3View[82], based on hyperbolic viewing (see Section 2.5), is due to be part of a Web site management tool of SGI whereas the similar ideas of Lamping *et al.*[73,74] are also exploited by a commercial spin–off of the Xerox company, called Inxight[†]. Earlier in this survey, we referred to NESTOR[105] or WebOFDAV[60], which can be used as web navigation tools. Other examples in this category are the Harmony Browser[1], Mapa[27], or Fetuccino[6] (the latter also combines the results of a web search engine with a graph visualisation).

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 Europe or overseas. 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 online journal which gather a similar community as the graph drawing symposia. The home page of the journal is at Brown University, (http://www.cs.brown.edu/publications/jgaa/), 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, specialised 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 visualisation communities.
- As said before, the yearly *CHI'XX* and *UIST'XX* conferences, both sponsored by ACM SIGCHI, very often contain important papers for information visualisation, due to the importance of user interface issue. Similarly, the *ACM Transaction on Human Computer Interaction* can be a valuable source of information.
- The yearly *InfoVis'XX* symposia form a separate track within the well–known *IEEE Visualization* conference; these symposia 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 *InfoVis'XX* symposia (besides being sponsored by IEEE, too). Our own experience is that the academic level of *InfoVis'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 visualisation as a separate track. The workshop has also become a joint Eurographics/IEEE TCVG symposium. As it stands now, this format will stay the same for the years to come.
- Some traditional computer graphics journals, like the *ACM Transaction on Computer Graphics and Visualization*, or the *Computer Graphics Forum* (including the proceedings of the Eurographics conferences), have an increasing number of papers in information visualisation.
- Finally, application oriented journals or conference proceedings may also include papers on information visualisation 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.

[†]http://www.inxight.com.

References

1. K. Andrews, "Visualizing Cyberspace: Information Visualisation in the Harmony Internet Browser", *Proceedings of the IEEE Symposium on Information Visualization (InfoVis'95)*, IEEE CS Press, pp. 97–105, (1995).

2. P.K. Argawal, B. Aronov, J. Pach, R. Pollack, and M. Sharir, "Quasi–Planar Graphs Have a Linear Number of Edges", *Proceedings of the Symposium on Graph Drawing, GD '95*, Springer–Verlag, pp. 1–7, (1995).

3. G. di Battista, P. Eades, R. Tamassia, and I.G. Tollis, "Algorithms for drawing graphs: an annotated bibliography", *Computational Geometry: Theory and Applications*, **4**(5), pp. 235–282, (1994).

4. G. di Battista, P. Eades, R. Tamassia, and I.G. Tollis, *Graph Drawing: Algorithms for the Visualisation of Graphs*, Prentice Hall, (1999).

5. R. A. Becker, S. G. Eick, and A. R. Wilks, "Visualizing Network Data", *IEEE Transactions on Visualization and Computer Graphics*, **1**(1), pp. 16-28, (1995).

6. 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", *Proceedings of 8th International World Wide Web Conference*, Elsevier Science, pp. 575–587, (1999).

7. J. Blythe, C. McGrah, and D. Krackhardt, "The Effect of Graph Layout on Inference from Social Network Data", *Proceedings of the Symposium on Graph Drawing, GD '95*, Springer–Verlag, pp. 40–51, (1995).

8. R. A. Botafogo, E. Rivlin, and B. Schneiderman, "Structural Analysis of Hypertexts: Identifying Hierarchies and useful Metrics," *ACM Transactions on Information Systems*, **10**(2), (1992).

9. F.J. Brandenburg, M. Himsolt, and C. Rohrer, "An Experimental Comparison of Force–Directed and Randomized Graph Drawing Algorithms", *Proceedings of the Symposium on Graph Drawing GD '95*, Springer–Verlag, (1996).

10. U. Brandes and D. Wagner, "A Bayesian Paradigm for Dynamic Graph Layout", *Proceedings of the Symposium on Graph Drawing GD '97*, Springer–Verlag, pp. 236–247, (1997).

11. 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 Conference Proceedings*, ACM Press, pp. 111–117, (1996). Also in S.K. Card *et al.*[12].

12. S.K. Card, J.D. Mackinlay, and B. Shneiderman (eds), *Readings in Information Visualization*. Morgan Kaufmann Publishers, (1999).

13. M.S.T. Carpendale, D.J. Cowperthwaite, and F.D. Fracchia, "3D Pliable Surfaces", *Proceedings of the UIST'95 Symposium*, ACM Press, pp. 217–266 (1995).

14. 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", *Proceedings of the Symposium on Graph Drawing GD '95*, Springer–Verlag, pp. 127–139, (1996).

15. M.S.T. Carpendale, D.J. Cowperthwaite, and F.D. Fracchia, "Extending Distortion Viewing from 2D to 3D", *IEEE Computer Graphics & Applications*, **17**(4), pp. 42–51, (1997). Also in S.K. Card *et al.*[12].

16. J. Carrière and R. Kazman, "Research Report: Interacting with Huge Hierarchies: Beyond Cone Trees", *Proceedings of the IEEE Conference on Information Visualisation '95*, IEEE CS Press, pp. 74–81, (1995).

17. M.C. Chuah, "Dynamic Aggregation with Circular Visual Designs", *Proceedings of the IEEE Symposium on Information Visualisation (InfoVis'98)*, IEEE CS Press, pp. 30–37, (1998).

18. M. C. Chuah, S. F. Roth, J. Mattis, and J. Kolojejchick, "SDM: Malleable Information Graphics", *Proceedings of the IEEE Symposium on Information Visualization*, IEEE CS Press, pp. 36–42, (1995).

19. H.S.M. Coxeter, *Introduction to Geometry*, John Wiley & Sons, Inc., (1973).

20. I. F. Cruz and R. Tamassia, "*Online tutorial on Graph drawing*", URL: http://www.cs.brown.edu/people/rt/papers/gd-tutorial/gd-constraints.pdf.

21. I. F. Cruz and J. P. Twarog, "3D Graph Drawing with Simulated Annealing", *Proceedings of the Symposium on Graph Drawing GD '95*, Springer–Verlag, pp. 162–165, (1995).

22. R. Davidson and D. Harel, "Drawing Graphs Nicely Using Simulated Annealing", *ACM Transaction on Graphics*, **15**(4), pp. 301–331, (1996).

23. E. Dengler and W. Cowan, "Human Perception of Laid–Out Graphs", *Proceedings of the Symposium on Graph Drawing GD '98*, Springer–Verlag, pp. 441–444, (1998).

24. A. Denise, M. Vasconcellos, and D.J.A. Welsh, "The random planar graph", *Congressus Numerantium*, **113**, pp. 61–79, (1996).

25. C. A. Duncan, M.T. Goodrich, and S. G. Kobourov, "Balanced Aspect

26. Trees and Their Use for Drawing Very Large Graphs", *Proceedings of the Symposium on Graph Drawing GD '98*, Springer–Verlag, pp. 111–124, (1998).

27. D. Durand and P.Kahn, "MAPA", *Proceedings of Ninth ACM Conference on Hypertext and Hypermedia (Hypertext'98)*, Pittsburgh, USA, (1998).

28. P. Eades, "A Heuristic for Graph Drawing," *Congressus Numerantium*, **42**, pp. 149–160 (1984).

29. P. Eades and K. Sugiyama, "How to Draw a Directed Graph", *Journal of Information Processing*, **13**(4), pp. 424–434, (1990).

30. P. Eades, "Drawing Free Trees", *Bulletin of the Institute for Combinatorics and its Applications*, pp. 10–36, (1992).

31. P. Eades and S.H. Whitesides, "Drawing Graphs in Two Layers", *Theoretical Computer Science*, **131**(2), pp. 361–374, (1994).

32. P. Eades and Q.–W. Feng, "Multilevel Visualization of Clustered Graphs", *Proceedings of the Symposium on Graph Drawing GD '96*, Springer–Verlag, pp. 101–112, (1997).

33. P. Eades, M. E. Houle, and R. Webber, "Finding the Best Viewpoints for Three–Dimensional Graph Drawings", *Proceedings of Symposium on Graph Drawing GD '97*, Springer–Verlag, p. 87–98, (1998).

34. S. G. Eick, "A Visualization tool for Y2K", *IEEE Computer*, **31**(10), p. 63–69, (1998).

35. J. Eklund, J. Sawers, and R. Zeiliger, "NESTOR Navigator: A tool for the Collaborative Construction of Knowledge Through Constructive Navigation", *Proceedings of Ausweb'99, The Fifth Australian World Wide Web Conference*, Southern Cross University Press, (1999).

36. B. Everitt, *Cluster Analysis*, First edition, Heinemann Educational Books Ltd., (1974).

37. 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*, Lawrence Erlbaum Associates, Inc., pp. 201–233, (1988). Also in S.K. Card *et al.*[12].

38. K.M. Fairchild, “Information Management Using Virtual Reality–Based Visualisations”, *Virtual Reality: Application and Explorations*, Academic Press, (1993).

39. A. Formella and J. Keller, “Generalized Fisheye Views of Graphs”, *Proceedings of the Symposium on Graph Drawing GD ’95*, Springer–Verlag, pp. 242–253, (1995).

40. E. Frécon and G. Smith, “WebPath — A Three Dimensional Web History”, *Proceedings of the IEEE Symposium on Information Visualisation (InfoVis’98)*, IEEE CS Press, (1998).

41. A. Frick, A. Ludwig and H. Mehldau “A Fast Adaptive Layout Algorithm for Undirected Graphs”, *Proceedings of the Symposium on Graph Drawing GD ’93*, Springer–Verlag, pp. 389–403, (1994).

42. M. Fröhlich and M. Werner, “Demonstration of the Interactive Graph Visualization System da Vinci”, *Proceedings of the DIMACS Workshop on Graph Drawing ’94*, Springer–Verlag, (1995).

43. T.M.J. Fruchterman and E.M. Reingold, “Graph Drawing by Force–Directed Placement,” *Software — Practice & Experience*, **21**, pp. 1129–1164, (1991).

44. G.W. Furnas, “Generalized Fisheye Views”, *Human Factors in Computing Systems, CHI ’86 Conference Proceedings*, ACM Press, pp. 16–23, (1986).

45. G.W. Furnas and B.B. Bederson, “Space–scale Diagrams: Understanding Multiscale Interfaces”, *Human Factors in Computing Systems, CHI ’95 Conference Proceedings*, ACM Press, pp. 234–241, (1995).

46. G.W. Furnas and X. Zhang, “MuSE: A Multi–Scale Editor”, *Proceedings of the UIST’98 Symposium*, ACM Press, (1998).

47. M.R. Garey and D. S. Johnson (1983). “Crossing number is NP–complete”, *SIAM Journal of Algebraic and Discrete Methods* **4**(3), pp. 312–316, (1983).

48. A. Garg and R. Tamassia, “On the Computational Complexity of Upward and Rectilinear Planarity Testing”, *Proceedings of Symposium on Graph Drawing, GD’95*, Springer–Verlag, pp. 286–297, (1995).

49. C. Gunn, “Visualizing Hyperbolic Space”, *Proceedings of the Workshop on Computer Graphics and Mathematics*, Springer–Verlag, pp. 299–313, (1992).

50. B. Hausmann, B. Slopianka, and H.–P. Seidel, “Exploring Plane Hyperbolic Geometry”, *Proceedings of the Workshop on Visualization and Mathematics*, Springer–Verlag, pp. 21–36, (1998).

51. T. He, “Internet–Based Front–End to Network Simulator”, *Data Visualiation ‘99, Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, Springer–Verlag, pp. 247–252, (1999).

52. M. Hemmje, C. Kunkel, and A. Willet: “LyberWorld — A Visualization User Interface Supporting Fulltext Retrieval”, *Proceedings of ACM SIGIR’94*, ACM Press, (1994).

53. M. Hemmje, *Actual Listing of Information Visualization Systems*, URL: http://www.darmstadt.gmd.de/~hemmje/Viri/visual.html, (1996).

54. R. J. Hendley, N. S. Drew, A. M. Wood, and R. Beale, “Narcissus: Visualising Information”, *Proceedings of the IEEE Symposium on Information Visualization*, IEEE CS Press, pp. 90–96, (1995).

55. I. Herman, M. Delest, and G. Melançon, “Tree Visualisation and Navigation Clues for Information Visualisation”, *Computer Graphics Forum*, **17**(2), pp. 153–165, (1998).

56. 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 Visualiation ‘99, Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, Springer–Verlag, pp. 13–22, (1999).

57. I. Herman, G. Melançon, M.M. de Ruiter, and M. Delest, *Latour — a Tree Visualisation System*, Reports of the Centre for Mathematics and Computer Sciences, INS-R9904, http://www.cwi.nl/InfoVisu/papers/LatourOverview.pdf, (1999).

58. J. Hopcroft and R. E. Tarjan, “Efficient Planarity Testing”, *Journal of the ACM*, **21**(4), pp. 549–568, (1974).

59. M.L. Huang, P. Eades, and J. Wang, “Online Animated Graph Drawing Using a Modified Spring Algorithm”, *Journal of Visual Languages and Computing*, **9**(6), 1998.

60. M.L. Huang, P. Eades, and R.F. Cohen, “WebOFDAV — Navigating and Visualizing the Web On–line with Animated Context Swapping”, *Proceedings of the 7th World Wide Web Conference*, Elsevier Science, pp. 636–638, (1998).

61. M.L. Huang and P. Eades, “A Fully Animated Interactive System for Clustering and Navigating Huge Graphs”, *Proceedings of the Symposium on Graph Drawing GD ’98*, Springer–Verlag, pp. 374–383, (1998).

62. C.–S Jeong and A. Pang, “Reconfigurable Disc Trees for Visualizing Large Hierarchical Information Space”, *Proceedings of the IEEE Symposium on Information Visualisation (InfoVis’98)*, IEEE CS Press, (1998).

63. B. Johnson and B. Schneiderman, “Tree–maps: a Space–filling Approach to the Visualisation of Hierarchical Information Structures”, *Proceedings of IEEE Visualisation’91*, IEEE CS Press, pp. 275–282, (1991). Also in S.K. Card *et al.*[12].

64. M. Juenger and P. Mutzel, “2–layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms”, *Journal of Graph Algorithms and Applications*, **1**, pp. 33–59, (1997).

65. D. Jungnickel, *Graphs, Networks and Algorithms*, Springer Verlag, (1999).

66. T. Kamada and S. Kawai, “An Algorithm for Drawing General Undirected Graphs”, *Information Processing Letters*, **31**, pp. 7–15, (1989).

67. K. Kaugars, J. Reinfelds, and A. Brazma, “A Simple Algorithm for Drawing Large Graphs on Small Screens”, *Proceedings of the Symposium on Graph Drawing GD ’94*, Springer–Verlag, pp. 278–281, (1995).

68. T.A. Keahey and E.L. Robertson, “Techniques for Non–Linear Magnification Transformations”, *Proceedings of the IEEE Symposium on Information Visualisation (InfoVis’97)*, IEEE CS Press, pp. 38–45, (1997).

69. D. Kimelman, B. Leban, T. Roth, and D. Zernik, “Reduction of Visual Complexity in Dynamic Graphs”, *Proceedings of the Symposium on Graph Drawing GD ’93*, Springer–Verlag, (1994).

70. M.R. Laguna, R. Martí, and V. Vals, “Arc Crossing Minimization in Hierarchical Digraphs with Tabu Search”, *Computers and Operations Research*, **24**(12), pp. 1165–1186, (1997).

71. M. Laguna and R. Martí, “GRASP and Path Relinking for 2–Layer Straight Line Crossing Minimization”, *INFORMS Journal on Computing*, **11**, pp. 44–52, (1999).

72. 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).

73. 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 Conference Proceedings*, ACM Press, (1995).

74. J. Lamping and R. Rao, “The Hyperbolic Browser: A Focus+context Technique for Visualizing Large Hierarchies”, *Journal of Visual Languages and Computing*, **7**(1), pp. 33–55, (1996). Also in S.K. Card *et al.*[12].

75. Y. K. Leung and M. D. Apperly, “A Review and Taxonomy of Distortion–Oriented Presentation Techniques”, *ACM Transactions on Computer–Human Interaction*, **1**(2), pp. 126–160, (1994). Also in S.K. Card *et al.*[12].

76. G. Melançon and I.Herman, *Circular Drawings of Rooted Trees*, Reports of the Centre for Mathematics and Computer Sciences, INS-R9817, http://www.cwi.nl/InfoVisu/papers/circular.pdf, (1998).

77. K. Mehlhorn and P. Mutzel, “On the Embedding Phase of the Hopcroft and Tarjan Planarity Testing Algorithm”, *Algorithmica*, **16**, pp. 233–242, (1996).

78. B. Mirkin, *Mathematical Classification and Clustering*, Kluwer Academic Publishers, (1996).

79. K. Misue, P. Eades, W. Lai, and K. Sugiyama, “Layout Adjustment and the Mental Map”, *Journal of Visual Languages and Computing*, **6**, pp. 183–210, (1995).

80. S. Mukherjea, J.D. Foley, and S. Hudson, “Visualizing Complex Hypermedia Networks through Multiple Hierarchical Views”, *Human Factors in Computing Systems, CHI ’95 Conference Proceedings*, ACM Press, pp. 331–337, (1995).

81. T. Munzner and P. Burchard, “Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space”, *Proceedings of the VRML’95 Symposium*, ACM SIGGRAPH, ACM Press, (1995).

82. T. Munzner, “H3: Laying out Large Directed Graphs in 3D Hyperbolic Space”, *Proceedings of the 1997 IEEE Symposium on Information Visualization (InfoVis’97)*, IEEE CS Press, pp. 2–10, (1997).

83. T. Munzner, “Drawing Large Graphs with H3Viewer and Site Manager”, *Proceedings of the Symposium on Graph Drawing GD ’98*, Springer–Verlag, pp. 384–393, (1998).

84. 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, S. Leipert, “A Library of Algorithms for Graph Drawing”, *Proceedings of the Symposium on Graph Drawing GD ’97 Symposium*, Springer–Verlag, pp. 456–457, (1998).

85. L. Nigay and F. Vernier, “Design method of interaction techniques for large information space”, *Proceedings of Advanced Visual Interfaces (AVI’98)*, ACM Press, (1998).

86. S. North, “Incremental Layout in DynaDAG”, *Proceedings of the Symposium on Graph Drawing GD ’95*, Springer–Verlag, pp. 409–418, (1995).

87. H.C. Purchase, “Which Aesthetic has the Greatest Effect on Human Understanding?”, *Proceedings of the Symposium on Graph Drawing GD ’97*, Springer–Verlag, pp. 248–261, (1998).

88. H.C. Purchase, R.F. Cohen, and M. James, “Validating Graph Drawing Aesthetics”, *Proceedings of the Symposium on Graph Drawing GD ’95*, Springer–Verlag, pp. 435–446, (1995).

89. E.M. Reingold and J.S. Tilford, “Tidier Drawing of Trees”, *IEEE Transactions on Software Engineering*, **SE-7**(2), pp. 223–228, (1981).

90. J. Rekimoto and M. Green, "The Information Cube: Using Transparency in 3D Information Visualisation", *Proceedings of the Third Annual Workshop on Information Technologies & Systems (WITS'93),* (1993).

91. J. S. Risch, D. B. Rex, S. T. Dowson, T. B. Walters, R. A. May, and B. D. Moon, "The STARLIGHT Information Visualization System", *Proceedings of the IEEE Conference on Information Visualisation*, IEEE CS Press, pp. 42–49, (1997).

92. 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 Conference Proceedings*, ACM Press, pp. 189–194, (1991).

93. G.G. Robertson, S.K. Card, and J.D. Mackinlay, "Information Visualization Using 3D Interactive Animation", *Communication of the ACM*, **36**(4), pp. 57–71, (1993). Also in S.K. Card *et al.*[12].

94. T. Roxborough and A. Sen, "Graph Clustering Using Multiway Ratio Cut", *Proceedings of the Symposium on Graph Drawing GD '97*, Springer Verlag, pp. 291–196, (1998).

95. M. Sarkar and M.H. Brown, "Graphical Fish–eye views of graphs", *Human Factors in Computing Systems, CHI '92 Conference Proceedings*, ACM Press, pp. 83–91, (1992).

96. M. Sarkar and M.H. Brown, "Graphical Fisheye Views", *Communications of the ACM*, **37**(12), pp. 73–84, (1994).

97. 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 Transactions on Computer–Human Interaction*, **3**(2), pp. 162–188, (1996).

98. Y. Shiloach, *Arrangements of Planar Graphs on the Planar Lattices*, PhD Thesis, Weizmann Institute of Science, Rehovot, Israel, (1976).

99. G. Sindre, B. Gulla, H. G. Jokstad, "Onion Graphs: Aesthetics and Layout", *Proceedings of IEEE/CS Symposium on Visual Languages (VL'93)*, IEEE CS Press, pp. 287–291, (1993).

100. T. C. Sprenger, M. Gross, D. Bielser, and T. Strasser, "IVORY — An Object–Oriented Framework for Physics–Based Information Visualization in Java", *Proceedings of the IEEE Symposium on Information Visualisation (InfoVis'98),* IEEE CS Press, (1998).

101. K. Sugiyama, S. Tagawa and M. Toda, "Methods for Visual Understanding of Hierarchical Systems Structures", *IEEE Transactions on Systems, Man and Cybernetic*s **SMC–11**(2), pp. 109–125, (1989).

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

103. W. Tutte, "How to draw a Graph", *Proceedings of the London Mathematical Society* **3**(13), pp. 743–768, (1963).

104. X.G. Viennot, "Trees everywhere", *Proceedings of the 15th CAAP Conference*, Springer–Verlag, pp. 18–41, (1990).

105. R. Zeilinger, "Supporting Constructive Navigation of Web Space", *Proceedings of the Workshop on Personalized and Solid Navigation in Information Space,* (1998).

106. J.Q. Walker II, "A Node-positioning Algorithm for General Trees", *Software — Practice and Experience*, **20**(7), pp. 685–705, (1990).

107. C. Ware and G. Franck, "Evaluation of Stereo and Motion Cues for Visualising Information in Three Dimensions", *ACM Transactions on Graphics*, **15**(2), pp. 121–140, (1996).

108. Y. C. Wei and C. K. Cheng, "Ratio Cut Partitioning for Hierarchical Designs," *IEEE Transactions on Computer Aided Design*, **10**(7), pp. 911–921,(1991).

109. G.J. Wills, "Niche Works — Interactive Visualization of Very Large Graphs", *Proceedings of the Symposium on Graph Drawing GD '97*, Springer–Verlag, pp. 403–415, (1998).

110. P. Young, *Three Dimensional Information Visualisation (Survey)*, Computer Science Technical Report, No. 12/96, Centre for Software Maintenance Department of Computer Science, University of Durham, URL: http://www.dur.ac.uk/~dcs3py/pages/work/documents/lit-survey/IV-Survey/index.html, (1996).