A General Introduction To Graph Visualization Techniques

Raga'ad M. Tarawneh[1], Patric Keller[2], and Achim Ebert[1]

1 **Computer Graphics and HCI Group**
University of Kaiserslautern, Germany
{tarawneh, ebert}@cs.uni-kl.de
2 **Software Engineering: Dependability Group**
University of Kaiserslautern, Germany
pkeller@cs.uni-kl.de

Abstract

Generally, a graph is an abstract data type used to represent relations among a given set of data entities. Graphs are used in numerous applications within the field of information visualization, such as VLSI (circuit schematics), state-transition diagrams, and social networks. The size and complexity of graphs easily reach dimensions at which the task of exploring and navigating gets crucial. Moreover, additional requirements have to be met in order to provide proper visualizations. In this context, many techniques already have been introduced. This survey aims to provide an introduction on graph visualization techniques helping the reader to gain a first insight into the most fundamental techniques. Furthermore, a brief introduction about navigation and interaction tools is provided.

1998 ACM Subject Classification A.1 Introduction and Survey, B.7.2 Design Aids

Keywords and phrases Graph Visualization, Layout Algorithms, Graph Drawing, Interaction Techniques

Digital Object Identifier 10.4230/OASIcs.VLUDS.2011.151

1 Introduction

One goal of information visualization is to provide techniques for converting (abstract) information e.g., in form of textual description into visual representations facilitating the perception and handling of hidden structures from underlying data sets [18]. In cases in which corresponding data elements have inherent relationships among each other, graph visualization methods are commonly applied to support the better understanding.

▶ Definition 1. Formally, a graph $G = (V, E)$ is a mathematical structure consisting of two sets, V the set of **vertices** (or nodes) of the graph, and E the set of **edges**. Each edge has a set of one or two vertices associated to it, which are called endpoints [53].

Many application areas use graphs to represent existing structures: For example, in social networks people of a group my represent the vertices of a graph where the different relations among them are represented by a set of edges. In other areas, like biology and chemistry graphs are widely used to represent molecular and genetic maps, as well as protein production paths. In the field of software engineering, graphs are used e.g., to represent the structure of complex software systems, or to represent the internal behavior/states of compilers. In the object-oriented field, graphs are used to depict the relations among different classes, e.g., UML diagrams. In general, any hierarchical structure may be represented as a tree, which is a subtype of a graph. An example for this sort of structure is the file structure of an

Proceedings of IRTG 1131 – Visualization of Large and Unstructured Data Sets Workshop 2011.
Editors: Christoph Garth, Ariane Middel, Hans Hagen; pp. 151–164

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


operation system. The organization structure of an institute may also be represented as a tree. For more information we refer to [53, 22].

Although graph visualization techniques are widely used in many application domains, they have some limitations one has to deal with. For example, the size of the represented graph may become an issue, e.g., providing layouts for very large graphs is possible, but often comes along with the loss of readability, at least for untrained users. This is associated with the limited human cognitive power and the screen space constraints given by the visualization devices. Providing a suitable technique helping the user interacting and navigating through the data is another important issue. The goal of graph visualization techniques is to increase the comprehension level of the data by providing intuitive, intelligible layouts as well as suitable interaction mechanisms.

This survey is organized as follows: In Section 2 we present a general overview concerning layout algorithms and a set of criteria to generate clean layouts. In order to increase the comprehension level of the visualized information, many interaction techniques have been proposed in the literature. In this context we present a brief introduction in Section 3. We conclude the survey by Section 4.

2 Graph Layout Algorithms

As mentioned in Section 1, a graph consists of a set of nodes connected by a set of edges. The trivial way to display this sort of data is to use node-link diagrams. They depict the relations among the data elements in form of lines [53, 22]. In [25], another visualization approach is proposed to display graph structures by exploiting space-filling techniques or space-nested layouts which implicitly represent the relations. This section provides an overview describing both approaches and the used algorithms.

2.1 Node-Link Layouts

The basic requirement of the node-link layout concerns the computation of the coordinates of the nodes and the representation of the lines. To increase the readability a clean layout should comply with the following criteria [29]:

- Nodes and edges should be evenly distributed.
- Edge-crossings should be minimized.
- Depict symmetric sub-graphs in the same way.
- Minimize the edge bending ratio.
- Minimize the edge lengths, which helps readers detecting the relations among different nodes faster.
- In cases where the data is inherently structured distribute the nodes into different layers. This increases the understandability of the underlying graph. For example, in data-flow diagrams it is recommended to separate the graph elements into different layers in a way that the final representation reflects the original semantics.

Many other criteria have been proposed in the literature, for more details please refer to [53, 29]. It is worth mentioning that it is hard to combine most of the criteria. Some of them conflict with others. In contrast, others are hard to realize in an efficient way. Many solutions have been proposed [29, 53] to overcome these issues. Most algorithms in practice represent a trade-off. Specifying the required criteria is an application dependent process. Prioritizing a set of criteria is an important pre-condition for finding suitable layout algorithms. The work of [41] concentrates on the topic of how to prioritize such criteria.

2.1.1 The Spring Layout Algorithm

The spring layout algorithm is widely known as ***force-directed layout***, which was originally proposed by Eades in 1984 [12]. Due to its simplicity and its ability to produce a symmetric layout, the force-directed layout is considered to be one of the popular node-link layouts. The spring layout algorithm represents the graph as a physical system, in which the graph nodes are considered to be a charged particles connected to each other using a set of springs. The first model was proposed by [12]. Each node is associated with two types of forces: *attraction forces* and *repulsive forces*. Given the node coordinates and the spring attributes the method aims at reducing the total energy of the the spring system by repositioning the nodes. The attraction force f_a is applied to the neighbor nodes which are connected by a spring, while the repulsive force f_r is applied to all graph nodes. These forces are defined as follows:

$$f_a(d) = k_a log(d), \qquad f_r(d) = \frac{k_r}{d^2} \tag{1}$$

where k_a and k_r are constants and d is the current distance between two nodes. Figure 5b shows a small example that emerged from applying this algorithm. Although, the force-directed approach produces clean, symmetric layouts with respect to graphs having moderate sizes, it is considered to be one of the expensive algorithms. Its time complexity exceeds $O(n^3)$ (see [53, 29, 22]), where n is the total number of nodes. Moreover, force-directed layouts lack in terms of predictability ([53, 29, 22, 58]), meaning that running the algorithm twice, produces different results. This leads to problems in maintaining the users mental map during the interaction with unstable layouts [58]. Despite the mentioned disadvantages, the force-directed layout algorithm has been widely used in many visualization frameworks [56]. Furthermore, the algorithm itself has been revisited and optimized many times to overcome its characteristic drawbacks (see [27, 14, 16, 13, 24, 53, 29, 24, 7]).

2.1.2 Topological Feature-Based Layout

The feature-based graph drawing concept has been proposed by Archambault et al. [2]. The concept is called TopoLayout, which is a multi-level, feature-based approach. The pipeline of this approach consists of four main steps, the first one is called the ***decomposition phase*** in which the graph is decomposed into many sub-graphs based on the topological features of each internal sub-graph. For example, if the nodes in one sub-graph are topologically connected among each other in form of a tree, then the set of nodes are grouped together representing a meta-node. Currently, TopoLayout detects seven topological features, including trees, complete graphs, bi-connected components, clusters, and the undefined structure that is called unknown feature. For more details we refer to [2]. The second step called the ***feature layout phase*** in which the meta-nodes or the grouped sub-graphs are laid out using one of the layout algorithms (tuned with its topological feature). The third phase called the ***crossing reduction phase*** aims to eliminate the crossing ratio in the produced layout. Finally, the ***overlap elimination phase*** aims to change the node sizes in the final layout to ensure that no nodes overlap each other. The final result for TopoLayout is a tree representing the graph hierarchy, in which each node represents a sub-graph in the original graph and each meta-edge represents the relation between tow sub-graphs in the original graph. This layout technique helps in drawing relatively large graphs. Also, it provides the user with details about the internal structure of the graph, which can be useful in extracting more information about the graph itself (see Figure 1). GrouseFlocks [3] was introduced to provide an interactive way to explore large input graphs through cuts of a superimposed hierarchy.

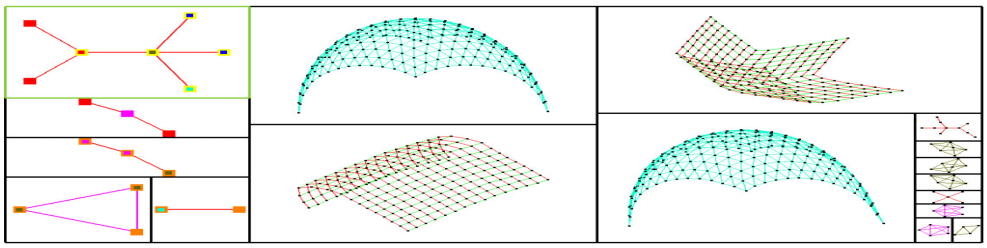


Figure 1 Layout generated by using the TopoLayout algorithm of [20].

The goal was to provide the user with the ability to see several different possible hierarchies of the same graph.

Before we introduce the tree layout concepts, it is worth to mention that both force-directed algorithms and the TopoLayout algorithm work perfectly with undirected graphs. Unfortunately, not many algorithms were designed for visualizing directed graphs. The Sugiyama algorithm was one of the first algorithm for drawing directed graphs [50]. The basic approach is to first layer the graph nodes, which means assigning a layer for each node and placing the nodes into the corresponding layer. Also, the algorithm includes two steps for reducing the edge-crossings and the node-overlappings. In general, directed graph layout algorithms are difficult to implement, this is due to the complexity of directed graphs. Therefore, many of the Sugiyama algorithm steps are considered to be NP-hard (see [17]), and some of them are NP-complete (see [11]).

2.1.3 Planar Graphs

Planar graphs are graphs that can be drawn without edge crossings in linear time. They emerge in various fields: CAD systems, circuit schematics, VLSI schematics, entity relationships diagrams and information system design [53, 29, 22]. To generate a planar layout for a general graph, some pre-requisites have to be fulfilled [22]:

- Testing whether it is possible to draw the given graph without edges crossings or not.
- Finding a planar layout algorithm satisfying the required application constrains.

Drawing a planar graph is supported by two well known algorithms, the first one called Fraysseix [9], Pach [28] and Pollack (FPP) [46] generates a drawing of a graph G on a grid of size $(2n-4)*(n-2)$ in $n\log(n)$ time. Later, the FPP algorithm was improved to run in linear time [28]. The second algorithm has been proposed by Schnyder [46]. It attempts to find a straight line embedding on a grid of n^2 nodes and runs in linear time. An example of a planar graph is shown in Figure 4b.

2.2 Tree Layout

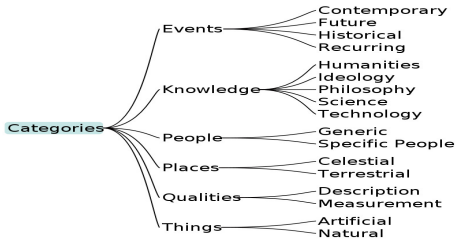
Many layout algorithms have been already proposed. In general, this may be attributed to the tree structure's simplicity and popularity. As a good starting point for tree layout algorithms we refer [53, 29].

2.2.1 Node-Link Tree layout Algorithms

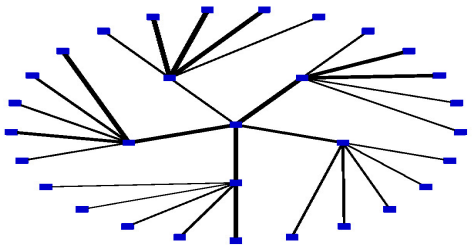
One of the basic approaches to draw a tree is to use node-link diagrams in which the parent-child relations are depicted as edges (see Section 2.1). The ***classical tree layout***

algorithm proposed by [42] is one of the early methods (see Figure 2a), it produces clean trees-representations in 2D and its implementation is straight forward. However, the technique is not declared space efficient because of its preference for one of two dominating growth directions, i.e., horizontal growth or vertical growth. To cope with this problem some compact tree layout algorithms have been implemented to produce a classical tree appearance in more compact fashion [10, 53, 29, 58, 22, 7].

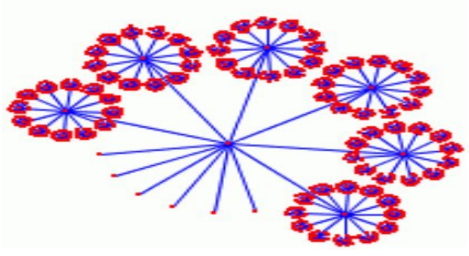
Another example of a node-link tree layout is the ***radial layout algorithm*** which was proposed by Eades [10]. A radial layout including its variations, places the root in the middle of co-centric circles and distributes the children of a sub-tree into circular shape according to their depth in the tree recursively. The radial layout uses space in more efficient way than the classical method. But it lacks the understandability of classical tree layouts, e.g. it is difficult to find the root of the tree (see Figure 2b) [53, 10, 39, 59]. As a sibling of the radial layout, the ***balloon layout*** has been introduced in [6]. Here, sibling sub-trees are drawn in a circle centered at their parents. This layout is effective in showing the tree structure. The balloon layout can be obtained by projecting a cone tree onto a plane [43, 53, 29, 58, 22] (see Figure 2c). ***H-Tree*** produces a classical layout for binary trees and works perfectly for balanced trees. But, again, it is hard to identify the root position [47] (see Figure 2d). All tree layout algorithms produce predictable results in at least linear time (the usual the complexity reaches from $O(n \log(n))$ to $O(n)$) [53, 29]. As a result of the comparison of different tree layout algorithms, we observed that the classical tree layout perfectly depicts the hierarchy structure of the tree, with sacrificing the screen space. While the radial layout, the h-tree layout, and the balloon tree layout use the screen space more efficiently but with difficulties in finding the root [53, 29, 58, 39].



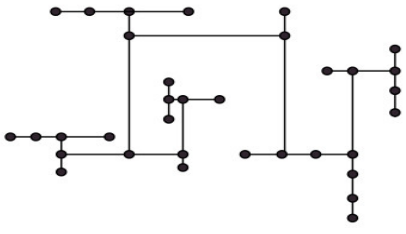
(a) Classical tree layout, produced with [19].



(b) Radial tree layout Example.



(c) Balloon tree layout: produced by [22].

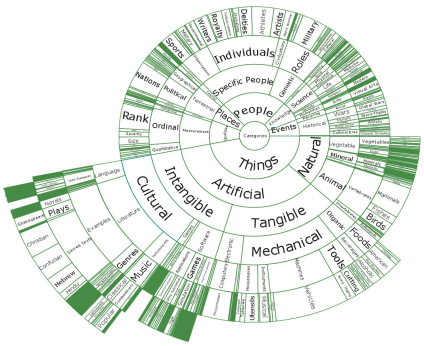


(d) H-Tree layout: produced by [22].

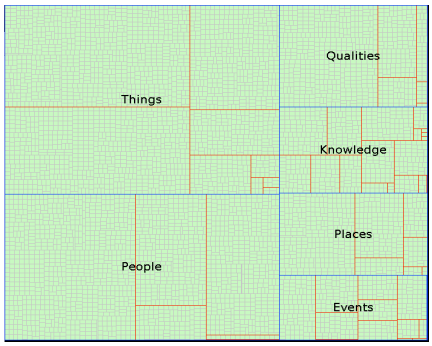
Figure 2 Tree Layout Examples.

2.2.2 Space-Filling Techniques

Space-filling techniques can be subdivided in two types: the ***Space-Division layout*** and the ***Space-Nested layout***. In the ***Space-Division layout***, the parent-child relation is depicted implicitly by attaching the children to their parent. ***Sunburst algorithm*** uses radial or circular space-filling techniques. The general belief of the developer community is that radial layout methodology better convey a hierarchy's structure without sacrificing the efficient screen space usage [49, 26]. One of the problems of this layout is that it is difficult to distinguish between the child-parent relationships and the sibling relationships, because both of them are expressed using adjacency. Moreover, due to the different number of children for each parent, the nodes sizes are difficult to control, the final layout might occupy a large space for node, which has many children. While other nodes are represented using a tiny thin rectangle that is not enough to show the node's label or the node's color (see Figure 3a). Whereas, in ***Space-Nested layouts*** the child-parent relationship is drawn using nested boxes. The idea is to place the children within their parent node. A good common example is the ***Treemap***, (see Figure 3b) [25, 48]. Nodes are represented as rectangles, each rectangle is subdivided into number of sub-rectangles equal to its children number. The subdivision process is performed recursively. This technique is popular because it uses the screen-space efficiently, and it shows the size of the leaves in a tree. However, this technique lacks the ability of showing the hierarchical structure of the tree. Also, due to the subdivision process it is highly possible to produce long and thin rectangles, which leads to problems in with the interaction (especially in selecting or highlighting the rectangles) [25, 48, 58, 22, 55].



(a) SunBurst layout.

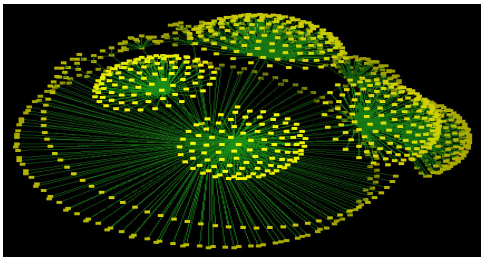


(b) TreeMap layout.

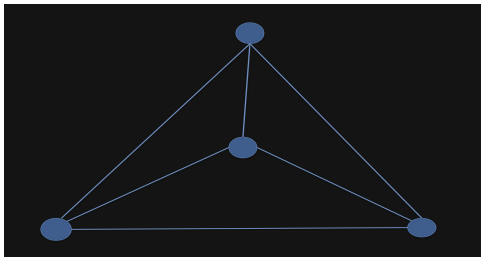
Figure 3 Examples of space-filling techniques [19].

2.3 Matrix Visualization

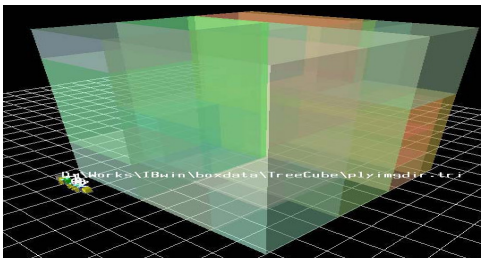
The ***matrix visualization*** is another technique that represents graph nodes relations implicitly (see Figure 5a). Here, each row and each column represent a node. The edge between two nodes is represented by the cell at which the corresponding row and column intersect. Edge attributes can be shown using different visual parameters such as color, size and shape. The scalability is limited, but the layout can produce clean representations of graphs having moderate size. However, the way the data is represented makes it difficult to detect the graph paths. For more details please refer to [1, 21].



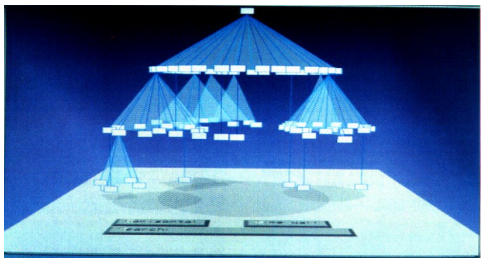
(a) Hyperbolic tree layout, produced with [52].



(b) Planar Graph Example.



(c) TreeCube layout, produced by [51].



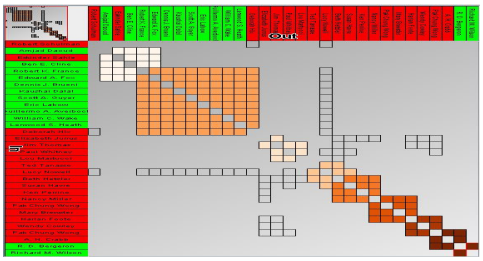
(d) Cone trees, produced by [43].

Figure 4 Graph layout Examples.

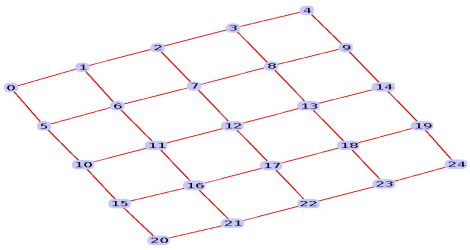
2.4 3D Layout

In addition to 2D representations, many layout algorithms have been extended to 3D. The reason behind it is that we are familiarized with 3D in the real world. So it is often more natural for us to explore data in 3D space. One example for a 3D layout is ***Treecube*** (see Figure 4c), a technique that has been proposed by [51] as an extension for the traditional treemap layout; it uses nested cubes to represent the parent-child relationships. The hyperbolic layout algorithm appeared for the first time in [32, 33], then it has successively been used by many others (e.g., [38, 37, 36]). The idea was to distribute the data entities over the hyperbolic space. Figure 4a shows an hyperbolic tree layout for a walrus-directory graph, which has been generated using the Walrus package [52]. This method can be displayed in 2D and 3D, providing a distorted view of the tree, which makes the interaction with large trees easier [22]. It is worth to mention that most of the force-directed techniques could be generalized easily to three dimensions (see [8]).

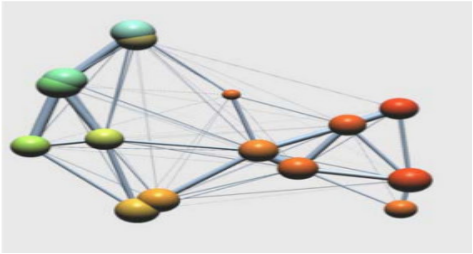
Conetree [43] is a technique that was originally developed to layout trees in 3D space. It places father nodes at the top of a cone with its children arranged evenly in the cone base. The layout has many layers; each one represents a tree level, in which all cones have the same height. The cone-base diameter for each layer is reduced in bottom-up fashion. This helps the lowest layer to fit into the width of the box containing the full cone tree, see Figure 4d . Based on [34], 3D visualization techniques face multiple challenges: First, objects in 3D may occlude each other which causes an issue while exploring the data set. Second, providing a suitable layout algorithm that assures less object-overlapping and reduces the edge-crossing is also considered as a big challenge. Third, the development of interaction techniques that are making the data exploration task easier and more intuitive is another big challenge. Finally, choosing an appropriate metaphor that increases the information understanding process is often hard to find. Many real-world metaphors were used to present data in 3D; examples can be found in [31].



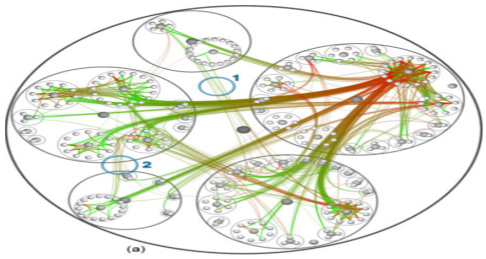
(a) Matrix visualization layout [21].



(b) Example of a force-directed layout.



(c) Clustering example by [54].



(d) Edge bundling example by[23].

Figure 5 Graph layout Techniques examples.

2.5 Nodes and Edges Clustering

Clustering techniques were introduced in graph visualization as a method to reduce visual cluttering in the final layout. This is achieved by producing an abstract view for the input graph. Reducing the number of visual elements does not only increases the clarity but also increases the rendering performance [30]. Clustering algorithms can be classified in two main types based on the criteria in the clustering process. The first type is called ***natural clustering***, here the structural information among the graph nodes is used to find a pattern of nodes having the same common criterion [44]. The second type is called the ***content-based clustering***, here the semantic meaning of the relations among the graph nodes is taken into account [57]. This type of clustering is rarely used, since it heavily depends on the application domain (reusing the same content-based clustering technique in another application usually is not possible). Therefore most graph visualization applications are using structure-based clustering algorithms. Many structural characteristics have been used as clustering criteria, such as the distances between graph nodes and node degrees. Natural clustering is widely applied to preserve the structure of the original graph [44]. This kind of clustering enables improved interaction facilities, because it eases applying of filtering techniques for the layout result and leads to an increased searching speed for specific data patterns. This could be accomplished by partitioning the nodes into a set of groups, then filter them based on a specified criteria, and finally narrow the search domain to the remaining clusters. Figure 5c shows an example of clustering techniques applied in the graph visualization field.

In [4], a clustering techniques for a special type of graph called small-world networks is presented. [53, 29] give a good overview on clustering and its applications, as well as a set of heuristics for each clustering method.

Another method for reducing the cluttering ratio is the ***edge clustering*** approach. Its goal is to free more space by grouping sets of edges that share the same end points, which reduces the visual cluttering in the final picture. Edge-bundling techniques are also proposed

to increase the readability of the graph. This is achieved by reducing the visual cluttering from the adjacency edges (see Figure 5d). For more details we refer the reader to [23].

Another example is the flow-map method: all edges that have the same source are grouped into one thick edge, generating a pattern of the edge-flow [40]. This technique intuitively shows the flow of the data from a single source to different destinations. However, it is only applicable for specific graph visualization applications, e.g., the migration path from a single source. Furthermore, in case of multiple sources, this approach causes a visual cluttering among the different flow maps, which leads to difficulties in reading the underlying graph.

3 Interaction Techniques

The goal of the visualization techniques mentioned above is to increase the comprehension level of the given data. Often, this goal cannot be achieved by only producing a static image representing the data. The ability of interacting with data has to be provided. Therefore, interaction and navigation techniques to facilitate the data exploration mission have been researched (e.g., [22, 48, 58]). In this section, we list a selection of interaction concepts that have been applied together with the visual layout algorithms. In [60] a summary of popular interaction techniques is presented:

- *Selecting:* giving the user the ability to highlight and process specific objects.
- *Abstracting/Elaborating:* changing the level of detail of the representation scheme. This allows users to get different insights into the data.
- *Reconfiguring:* giving the user the ability to change the layouts for the same representation scheme, such as sorting the graph nodes based on a specific criteria.
- *Encoding:* switching between different layout methods, such as converting the node-link diagram into a sunburst layout.
- *Exploring:* this is related to giving the user the ability to change the view point of the graph layout. Zooming and panning are examples of this category.
- *Filtering:* removing unnecessary detail and displaying the remaining items in a more visible fashion. The main concept is to filter the data nodes based on their attributes in order to make the querying process easy and fast. For more examples see [5].
- *Connecting:* giving users the ability to highlight the paths between relevant objects and the focus object.

3.1 Zooming and Panning

Zooming and Panning are basic tools for exploring large amounts of information. Panning means moving the camera across the scene, while zooming allows users to switch between the abstract and the detailed views. Geometric zooming adjusts the screen transformation and thereby allows increasing or decreasing the magnification of the displayed graph. Semantic zooming means that not only the size of objects but also the displayed information may change when approaching a particular area of the graph.

Both, zooming and panning, are complement to each other. An example are geographical maps, like the ones used by Google Earth: suppose the user zoomed into an area next to Frankfurt in Germany. If he or she wants to change the view to another area, lets say Amman in Jordan, he or she usually has to zoom out first to get a better overview, then pan to the Amman region, and finally again zoom into Amman. Doing this procedure without panning in the middle will need a long time to find the destination [22, 58]. In [15], an elegant model was introduced to explain how zooming and panning work together. The proposed concept is called space-scale diagrams. It defines an abstract space by first creating multiple copies

of the original 2D image. In a second step, they are stacked up to construct an inverted pyramid, on which each copy is placed in a certain magnification level. The space-scale diagram can be used with both zooming types, not only for the geometric zooming but also for the semantic zooming [15, 22].

3.2 Focus+Context Techniques

Focus+context techniques are addressing the problem of losing context when zooming into given data. Suppose you have zoomed into a picture, the result would be that you can only see the zoomed-in area without having an idea about the surrounding areas in the picture. Here, focus+context comes into play: it gives users the ability to see the primary object in a detailed view (focus) together with an overview of all the surrounding information (context). In general, losing context is considered to be an issue in information visualization applications. In order to alleviate this problem, focus+context techniques appeared to give the user the ability to focus on some details without losing the global context [45]. This concept does not replace the zooming and panning methods, but rather complements them. The majority of visualization application systems implement both techniques together as an interaction tool.

Many approaches provide focus+context views. Overview+detail is one of the earliest focus+context approaches, in which separate display regions for different resolutions are used. It enables users to switch between different displays frequently [35]. Fish-eye is one of the most popular focus+context techniques [45]: the area of interest becomes larger while at the same time the other regions of the layout are successively shown with less detail. In the fish-eye approach, computing the hyperbolic coordinates is faster than the layout algorithm, which is considered as an issue for the interaction with the visualization [22, 58]. The distortion appears as a negative consequence of this technique, which leads to destroying many aesthetics criteria controlling the layout algorithm, e.g., unwanted edge crossings might appear [22, 58].

4 Conclusion

The purpose of this survey was to give a brief and general overview on fundamental graph visualization techniques, a sub-field of information visualization. Graph visualization focuses on representing abstract data elements and the relationships between them visually, thus reflecting the structure of the data. The goal is to increase the cognitive level of the local and global structure.

Node-link diagrams were the first introduced approaches to depict graphs. In this regards, a graph is drawn using a set of points representing the graph vertices which are connected by lines or curves representing the graph edges. These approaches perform well for graphs of moderate size. However, data sets reflecting real world data often become very large. Consequently, this sort of algorithms appear to be insufficient and do not scale well. To adapt to larger graph sizes, new layout schemes have been developed. Space-filling techniques such as Treemaps are one approach attempting to display relatively large graphs, specifically trees, by representing the relations between the nodes implicitly. Therefore, it is difficult to answer the question: which approach performs better; This highly depends on the application and the particular user requirements. On one hand, node-link approaches lack the scalability but are able to display the relations between graph elements. On the other hand, space-filling techniques are space-efficient but lack in terms of understandability.

Along with the visual aspects, suitable and intuitive interaction techniques are key elements to gain better insights into the visualized data. Many interaction methods were introduced in the literature. In this context, zooming and panning are fundamental interaction techniques, but using them separated can cause the loss of context. Therefore, focus+context techniques were proposed to alleviate these drawbacks. Overview+detail, for example, constitutes an approach using separate display regions to resolve those issues. Fish-eye methods can provide different level of details at the same time by integrating them in a single display region. This allows the users to zoom without losing their focus.

References

1 J. Abello and F. van Ham. Matrix zoom: A visual interface to semi-external graphs. In *Proceedings of the IEEE Symposium on Information Visualization*, INFOVIS '04, pages 183–190, Washington, DC, USA, 2004. IEEE Computer Society.

2 D. Archambault, T. Munzner, and D. Auber. Topolayout: Multilevel graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics*, 13:305–317, 2007.

3 D. Archambault, T. Munzner, and D. Auber. Grouseflocks: Steerable exploration of graph hierarchy space. *IEEE Transactions on Visualization and Computer Graphics*, 14:900–913, 2008.

4 D. Auber, Y. Chiricota, F. Jourdan, and G. Melançon. Multiscale visualization of small world networks. In *Proceedings of the Ninth annual IEEE conference on Information visualization*, INFOVIS'03, pages 75–81, Washington, DC, USA, 2003. IEEE Computer Society.

5 E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 73–80, New York, NY, USA, 1993. ACM.

6 J. Carriere and R. Kazman. Research report: Interacting with huge hierarchies: beyond cone trees. In *Proceedings of the 1995 IEEE Symposium on Information Visualization*, INFOVIS '95, pages 74–, Washington, DC, USA, 1995. IEEE Computer Society.

7 Graph Drawing Community, November 2011. `http://www.graphdrawing.org/`.

8 I. F. Cruz and J. P. Twarog. 3d graph drawing with simulated annealing. In *Proceedings of the Symposium on Graph Drawing*, GD '95, pages 162–165, London, UK, UK, 1996. Springer-Verlag.

9 H. de Fraysseix, J.s Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

10 P. Eades. Drawing free trees. *Bulletin of the Institute for Combinatorics and its Applications, pp. 10 -36*, 1992.

11 P. Eades and S. Whitesides. Drawing graphs in two layers. *Theor. Comput. Sci.*, 131(2):361–374, 1994.

12 P.A. Eades. A heuristic for graph drawing. In *Congressus Numerantium*, volume 42, pages 149–160, 1984.

13 B. Finkel and R. Tamassia. Curvilinar graph drawing using the force-directed method. In *Proc. 12th Int. Symposium on Graph Drawing, 2004, Springer LNCS 3383*, pages 448–453, 2004.

14 T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Softw: Pract. Exper.*, 21(11):1129–1164, November 1991.

15 G. W. Furnas and B. B. Bederson. Space-scale diagrams: understanding multiscale interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*,

CHI '95, pages 234–241, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

16 P. Gajer and S. G. Kobourov. Grip: Graph drawing with intelligent placement. In *Proceedings of the 8th International Symposium on Graph Drawing*, GD '00, pages 222–228, London, UK, UK, 2001. Springer-Verlag.

17 M. R. Garey and D. S. Johnson. Crossing Number is NP-Complete. *SIAM Journal on Algebraic and Discrete Methods*, 4(3):312–316, 1983.

18 N. Gershon, S.T Card, and S. G. Eick. Information visualization tutorial. In *CHI '99 extended abstracts on Human factors in computing systems*, CHI EA '99, pages 149–150, New York, NY, USA, 1999. ACM.

19 J. Heer, Ch. Collins, and M. Dudek, November 2011. `http://prefuse.org/`.

20 Ch. Heine, November 2011. `http://www.informatik.uni-leipzig.de/~hg/libgraph/`.

21 N. Henry and J. D. Fekete. Matrixexplorer: a dual-representation system to explore social networks. *IEEE Transactions on Visualization and Computer Graphics*, 12:677–684, 2006.

22 I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, January 2000.

23 D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Trans. Vis. Comput. Graph.*, 12(5):741–748, 2006.

24 D. Hume. *Graph Drawing Algorithms*, pages 400–401. Springer London, 2 edition, 2006.

25 B. Johnson and B. Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2nd conference on Visualization '91*, VIS '91, pages 284–291, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

26 H. H. Kagdi and J. I. Maletic. Onion graphs for focus+context views of uml class diagrams. In Jonathan I. Maletic, Alexandru Telea, and Andrian Marcus, editors, *VISSOFT*, pages 80–87. IEEE Computer Society, 2007.

27 T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31(1):7–15, April 1989.

28 G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996.

29 M. Kaufmann and D. Wagner. *Drawing Graphs: Methods and Models (Lecture Notes in Computer Science)*. Springer, 1 edition, January 2001.

30 D. Kimelman, B. Leban, T. Roth, and D. Zernik. Reduction of visual complexity in dynamic graphs. In *Proceedings of the DIMACS International Workshop on Graph Drawing*, GD '94, pages 218–225, London, UK, UK, 1995. Springer-Verlag.

31 E. Kleiberg, H. van de Wetering, and J. J. Van Wijk. Botanical visualization of huge hierarchies. In *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, INFOVIS '01, pages 87–, Washington, DC, USA, 2001. IEEE Computer Society.

32 J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '95, pages 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

33 Jonh Lamping and Ramana Rao. The hyperbolic browser: A focus+context technique for visualizing large hierarchies. *Journal of Visual Languages & Computing*, 7(1):33 – 55, 1996.

34 M. Larrea, D. Urribarri, S. Martig, and S. Castro. Spherical layout implementation using centroidal voronoi tessellations. *CoRR*, abs/0912.3974, 2009.

35 H. Lieberman. Powers of ten thousand: navigating in large information spaces. In *UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 15–16+, New York, NY, USA, 1994. ACM.

36 T. Munzner. H3: laying out large directed graphs in 3d hyperbolic space. In *Proceedings of the 1997 IEEE Symposium on Information Visualization (InfoVis '97)*, pages 2–, Washington, DC, USA, 1997. IEEE Computer Society.

37 T. Munzner. Drawing large graphs with h3viewer and site manager (system demonstration). In *In Proceedings of Graph Drawing'98, number 1547 in Lecture Notes in Computer Science*, pages 384–393. Springer-Verlag, 1998.

38 T. Munzner and P. Burchard. Visualizing the structure of the world wide web in 3d hyperbolic space. *Proceedings of the first symposium on Virtual reality modeling language VRML 95*, pages 33–38, 1995.

39 Q. V. Nguyen and M. L. Huang. Space-optimized tree: a connection+enclosure approach for the visualization of large hierarchies. *Information Visualization*, 2:3–15, March 2003.

40 D. Phan, L. Xiao, R. Yeh, P. Hanrahan, and T. Winograd. Flow map layout. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 219–224, 2005.

41 H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In *Proceedings of the 5th International Symposium on Graph Drawing*, GD '97, pages 248–261, London, UK, UK, 1997. Springer-Verlag.

42 E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Trans. Softw. Eng.*, 7(2):223–228, March 1981.

43 G. G. Robertson, J. D. Mackinlay, and S. K. Card. Cone Trees: animated 3D visualizations of hierarchical information. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, CHI '91, pages 189–194, New York, NY, USA, 1991. ACM.

44 T. Roxborough and A. Sen. Graph clustering using multiway ratio cut. In *Proceedings of the 5th International Symposium on Graph Drawing*, GD '97, pages 291–296, London, UK, UK, 1997. Springer-Verlag.

45 M. Sarkar and M. H. Brown. Graphical fisheye views of graphs. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '92, pages 83–91, New York, NY, USA, 1992. ACM.

46 W. Schnyder. Embedding planar graphs on the grid. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, SODA '90, pages 138–148, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.

47 Y. Shiloach. *Arrangements of Planar Graphs on the Planar Lattices*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1976.

48 G. Sindre, B. Gulla, and H. G. Jokstad. Onion graphs: aesthetics and layout. In *VL*, pages 287–291, 1993.

49 J. Stasko and E. Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Proceedings of the IEEE Symposium on Information Vizualization 2000*, INFOVIS '00, pages 57–, Washington, DC, USA, 2000. IEEE Computer Society.

50 K. Sugiyama, Sh. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *Ieee Transactions On Systems Man And Cybernetics*, 11(2):109–125, 1981.

51 Y. Tanaka, Y. Okada, and K. Niijima. Treecube: Visualization tool for browsing 3d multimedia data. *International Conference on Information Visualisation*, 0:427, 2003.

52 CAIDA The Cooperative Association for Internet Data Analysis, November 2011. `http://www.caida.org/tools/visualization/walrus/`.

53 I. G. Tollis, G. Di Battista, P. Eades, and R. Tamassia. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, July 1998.

54 F. v. van Ham and J. J. v. van Wijk. Interactive Visualization of Small World Graphs. In *INFOVIS '04: Proceedings of the IEEE Symposium on Information Visualization (INFOVIS'04)*, pages 199–206, Washington, DC, USA, 2004. IEEE Computer Society.

55 Jarke J. Van Wijk and Huub van de Wetering. Cushion treemaps: Visualization of hierarchical information. In *Proceedings of the 1999 IEEE Symposium on Information Visualization*, INFOVIS '99, pages 73–78, Washington, DC, USA, 1999. IEEE Computer Society.

56 C. Walshaw. A multilevel algorithm for force-directed graph drawing. In *Proceedings of the 8th International Symposium on Graph Drawing*, GD '00, pages 171–182, London, UK, UK, 2001. Springer-Verlag.

57 M. Wattenberg. Visual exploration of multivariate graphs. In *In Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 811–819. ACM Press, 2006.

58 C. Weiwei and Q. Huamin. A Survey on Graph Visualization. *Hong Kong University of Science and Technology Clear Water Bay, Kowloon, Hong Kong*, 2008.

59 K.-P. Yee, D. Fisher, R. Dhamija, and M. Hearst. Animated exploration of dynamic graphs with radial layout. In *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, INFOVIS '01, pages 43–, Washington, DC, USA, 2001. IEEE Computer Society.

60 J. S. Yi, Youn Ah Kang, J. Stasko, and J. Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1224–1231, 2007.