Graph Visualization

Yifan Hu and Martin Nöllenburg

Synonyms

- Graph Drawing
- Graph Layout
- Network Visualization

Definition

Graph visualization is an area of mathematics and computer science, at the intersection of geometric graph theory and information visualization. It is concerned with visual representation of graphs that reveals structures and anomalies that may be present in the data, and helps the user to understand and reason about the graphs.

Overview

Graph visualization is concerned with visual representations of graph or network data. Effective graph visualization reveals structures that may be present in the graphs, and helps the users to understand and analyze the underlying data.

A graph consists of nodes and edges. It is a mathematical structure describing relations among a set of entities, where a node represents an entity, and an edge exists between two nodes if the two corresponding entities are related.

A graph can be described by writing down the nodes and the edges. For example, this is a social network of people and how they relate to each other:

{*Andre* ↔ *Beverly, Andre* ↔ *Diane, Andre* ↔ *Fernando, Beverly* ↔ *Garth, Beverly* ↔ *Ed, Carol* ↔ *Andre, Carol* ↔ *Diane, Carol* ↔ *Fernando, Diane* ↔ *Beverly, Diane* ↔ *Garth, Diane* ↔ *Ed, Farid* ↔ *Aadil, Farid* ↔ *Latif, Farid* ↔ *Izdihar, Fernando* ↔ *Diane, Fernando* ↔ *Garth, Fernando* ↔ *Heather, Garth* ↔ *Ed,*

Garth ↔ Heather, Heather ↔ Jane, Izdihar ↔ Mawsil, Jane ↔ Farid, Jane ↔ Aadil, Latif ↔ Aadil, Mawsil ↔ Latif}.

This social network tells us that "Farid" is a friend of "Aadil", "Latif" is a friend of "Aadil", and so on. However, this mathematical notation of the network does not convey immediately the structure of the network. On the other hand, Fig. 1 shows a visualization of this graph. We can see at a glance that this graph has two clusters. This example illustrates that graph visualization can give us an overall sense of the data. It reveals structures and anomalies, and helps us to ask questions that can in turn be answered through interacting with the visualization itself, or by writing algorithms to mine the data for evidence seen in the visualization.

In this chapter, a graph $G = (V,E)$ consists of a set of nodes (vertices) V, and a set of edges E, which are pairs of nodes. Denote by $n = |V|$ and $m = |E|$ the number of nodes and edges, respectively. If there is an edge from node i to node j, we denote that as $i \to j$. If the graph is undirected, then we denote the edge as $i \leftrightarrow j$, and call i and j neighboring (or adjacent) nodes.

Node-Link Diagrams and Layout Aesthetics

By far the most common type of graph layout is the so-called *node-link diagram* as seen in Fig. 1. Here nodes are represented by points or simple geometric shapes like ellipses or rectangles, whereas edges are drawn as simple curves linking the corresponding pair of nodes. In this chapter we restrict our attention to node-link diagrams; for alternative types of graph representations see the survey of von Landesberger et al (2011).

The algorithmic graph layout problem consists in finding node positions in the plane (or in 3-dimensional space) and edge representations as straight lines or simple curves such that the resulting layout faithfully depicts the graph and certain aesthetic quality criteria are satisfied and optimized. We list the most common criteria. The influence of most of them on human graph reading tasks has been empirically confirmed (Purchase 1997).

- **crossings:** the fewer edge crossings the better (a layout without crossings exists only for *planar* graphs)
- **bends:** the fewer edge bends the better; ideally edges are straight-line
- **edge lengths:** use as uniform edge lengths as possible
- **angular resolution:** angles between edges at the same node should be large
- **crossing angles** angles of pairs of crossing edges should be large
- **area and aspect ratio:** the layout should be as compact as possible
- **edge slopes:** few and regularly spaced edge slopes should be used (e.g., *orthogonal* layouts use only horizontal and vertical edges)
- **neighborhood:** neighbors of each node in the graph should be neighbors in the layout as well

The above list is not comprehensive and there may be additional application-specific constraints and criteria or global aspects such as displaying symmetries. Moreover, some criteria may contradict each other. Typically only a subset of criteria is optimized by a graph layout al-

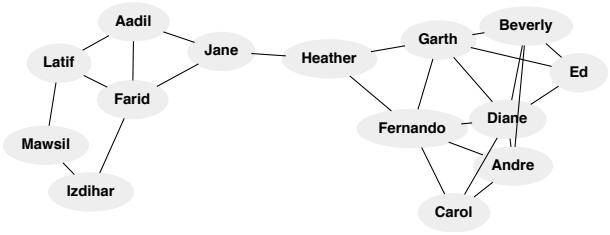

Fig. 1 Graph visualization of a small social network.

gorithm and trade-offs between different criteria need to be considered.

Key Research Findings

Undirected Graph Drawing

The layout algorithms and techniques presented in this section do not make use of edge directions, but they can obviously be applied to both undirected and directed graphs. Undirected graphs are often represented by node-link diagrams with straight-line edges. We denote by x_i the location of node i in the layout. Here x_i is a point in 2- or 3-dimensional Euclidean space.

Spring embedders (Eades 1984; Fruchterman and Reingold 1991; Kamada and Kawai 1989) are the most widely used layout algorithms for undirected graphs. They attempt to find aesthetic node placement by representing the problem as one of minimizing the energy of a physical system. The guiding principles are that nodes that are connected by an edge should be near each others, while no nodes should be too close to each other (cf. neighborhood aesthetic). Depending on the exact physical model, we could further divide the spring embedders into two types. For convenience, we call the first type Spring-electrical model, and the second type Spring/Stress model.

Spring-electrical model

This model was first introduced by Peter Eades (1984). A widely used variant, which is given below, is due to Fruchterman and Reingold (1991). The model is best understood as a system of springs and electrical charges, therefore we name this as the Spring-electrical model, to differentiate the spring/stress model that relies on springs only, even though historically both are called spring embedders.

In this model, each edge is replaced by a spring with an ideal length of 0, which pulls nodes that share an edge together. At the same time, imagine that nodes have the same type of electrical charges (e.g., positive) that push them apart. Specifically, there is an attractive spring force exerted on node i from its neighbor j, which is proportional to the squared distance between these two nodes,

$$F_a(i,j) = -\frac{\left\|x_i - x_j\right\|^2}{K}\frac{x_i - x_j}{\left\|x_i - x_j\right\|},\ i \leftrightarrow j, \tag{1}$$

where K is a parameter related to the nominal edge length of the final layout. The repulsive electrical force exerted on node i from *any* node j is inversely proportional to the distance between these two nodes,

$$F_r(i,j) = \frac{K^2}{\|x_i - x_j\|} \frac{x_i - x_j}{\|x_i - x_j\|}, \; i \neq j. \tag{2}$$

The spring-electrical model can be solved with a force-directed procedure by starting from an initial (e.g., random) layout, calculating the combined attractive and repulsive forces on each node, and moving the nodes along the direction of the force for a certain step length. This process is repeated, with the step length decreasing every iteration, until the layout stabilizes. This procedure is formally stated in Algorithm 1.

The spring-electrical model as described by equations 1-2 cannot be used directly for large graphs. The repulsive force exists on all pairs of nodes, so the computational complexity is quadratic in the number of nodes. Force approximation techniques based on space decomposition data structure, such as the Barnes-Hut algorithm, can be used to approximate the repulsive forces efficiently (Tunkelang 1999; Quigley 2001; Hachul and Jünger 2004).

For large graphs, the force-directed algorithm, which uses the steepest descent process and re-positions one node at a time to minimize the energy locally, is likely to be stuck at local minima, because the physical system of springs and electrical charges can have many local minimum configurations. This can be overcome by the multilevel technique. In this technique, a sequence of smaller and smaller graphs are generated from the original graph, each captures the essential connectivity information of its parent. The force-directed algorithm can be applied to this sequence of graphs, from small to large, each time using the layout of the smaller graph as the start layout for the larger graph. Combining the multilevel algorithm and the force approximation technique, algorithms based on the spring-electrical model can be used to layout graphs with millions of nodes and edges in seconds (Walshaw 2003; Hu 2005).

Spring/Stress Model

The spring model, also know as the stress model, assumes that there are springs connecting all pairs of nodes in the graph, with the ideal spring length equal to the graph theoretical distance between the nodes. The spring energy, also known as the stress, of this spring system is

$$\sum_{i \neq j} w_{\mathrm{ij}} \left(\|x_i - x_j\| - d_{\mathrm{ij}} \right)^2, \tag{3}$$

where d_{ij} is the ideal distance between nodes i and j. The layout that minimizes the above stress energy is an optimal layout of the graph. Typically $w_{\mathrm{ij}} = 1/{d_{\mathrm{ij}}}^2$.

The spring model was proposed by Kamada and Kawai (1989) in graph drawing, although it dates back to Multidimensional Scaling (MDS) (Kruskal 1964; Kruskal and Seery 1980), and the term MDS is sometimes used to describe the embedding process based on the stress model.

There are several ways to minimize the spring energy (3). A force-directed algorithm could be applied, where the

Algorithm 1 ForceDirectedAlgorithm(G, x, tol, K)

```
1  input: graph G = (V,E), initial positions x, tolerance tol, and nominal edge length K
2  set step = initial step length
3  repeat
4      x^0 = x
5      for (i ∈ V) {
6          f = 0   // f is a 2/3D vector
7          for (j ↔ i, j ∈ V) f ← f + F_a(i,j)   // attractive force, see equation (1)
8          for (j ≠ i, j ∈ V) f ← f + F_r(i,j)   // repulsive force, see equation (2)
9          x_i ← x_i + step * (f/||f||)   // update position of node i
10     }
11 until (||x − x^0|| < tol * K)
12 return x
```

force exerted on node i from all other nodes j ($j \neq i$) is

$$F(i,j) = -w_{\mathrm{ij}}(\left\|x_i - x_j\right\| - d_{\mathrm{ij}})\frac{x_i - x_j}{\left\|x_i - x_j\right\|}. \tag{4}$$

In recent years the stress majorization technique (Gansner et al 2004) became a preferred way to minimize the stress model due to its robustness.

In the stress majorization process, systems of linear equations are repeatedly solved,

$$L_w X = L_{w,d}\, Y \tag{5}$$

here X and Y are matrices of dimension $n \times 2$, and the weighted Laplacian matrix L_w is positive semi-definite and has elements

$$(L_w)_{\mathrm{ij}} = \begin{cases} \sum_{l \neq i} w_{\mathrm{il}}, & i = j \\ -w_{\mathrm{ij}}, & i \neq j \end{cases} \tag{6}$$

and the right-hand-side $L_{w,d}\ y$ has elements

$$\left(L_{w,d}\ y\right)_{\mathrm{i}} = \sum_{l \neq i} w_{\mathrm{il}}\, d_{\mathrm{il}}(y_i - y_l)/\left\|y_i - y_l\right\|. \tag{7}$$

Here matrix Y is the current best guess of the optimal layout in 2D. The solution X serves as Y in the next iteration. A good initial layout Y is often helpful to achieve a good final layout.

On large graphs, the stress model (3) is not scalable. Formulating the model requires computing the all-pairs shortest path distances, and a quadratic amount of memory to store the distance matrix. Furthermore the solution process has a computational complexity at least quadratic in the number of nodes. In recent years there have been attempts (Brandes and Pich 2007; Gansner et al 2013; Ortmann et al 2016) at developing more scalable ways of drawing graphs that still satisfy the user specified edge length as much as possible, without having to carry out the all-pairs shortest path distances.

Directed graph drawing algorithms

Directed graphs express relations that have a source and a target, e.g., senders and recipients of messages or hierarchical relationships in an organization. Using arrowheads to indicate directions, any undirected graph layout algorithm can be used for visualizing directed graphs. Yet, often the edge directions carry important information that should directly influence the layout geometry. In this section we discuss layout algorithms that make use of edge directions.

Layered graph layout

Layered graph layout is particularly useful for hierarchical graphs, which have no or only few cyclic relationships. For such graphs it is a natural to ask for a drawing, in which all or most edges point into the same direction, e.g., upward. A classic algorithmic framework for upward graph layouts is layered graph drawing, often also known as Sugiyama layout named after the fundamental paper by Sugiyama et al (1981). Each node is assigned to a layer and for each edge it is required that the layer of the source node is below the layer of the target node. For a detailed survey of layered graph layout methods see Healy and Nikolov (2014).

The process of computing layered graph layouts consists of a series of four steps. Each step influences the input to the subsequent steps and thus also the achievable layout quality.

1. **Cycle removal.** If the graph contains directed cycles, this step determines a small set of so-called *feedback edges* whose reversal yields an *acyclic* graph. The subsequent steps will draw this acyclic graph and in the end of the process the original edge directions are restored. Finding a minimum set of edges to reverse is equivalent to finding a so-called *minimum feedback arc set*, a classical NP-hard problem (Karp 1972). Several heuristics and approximations algorithms can be used to find small feedback edge sets in practice (Gansner et al 1993; Eades et al 1993).
2. **Layer assignment.** This step assigns each node to a horizontal layer, which can be thought of as an integer y-coordinate, such that all edges point upward. Multiple nodes can be assigned to the same layer. Typical goals in this phase are to find compact layer assignments that use few layers and distribute nodes evenly to the layers, while ensuring that all edges point from a lower layer to a higher layer. After the layer assignment, some edges may span several layers. Such long edges are usually subdivided by dummy nodes to guarantee that for the next step all edges connect nodes in neighboring layers. In order to keep edges short, a natural goal is to minimize the number of dummy nodes needed. Suitable layer assignments can be computed efficiently for most optimization goals, e.g., minimizing the number of layers based on topological sorting of the graph. For compact layouts with few dummy nodes and bounds on the number of nodes per layer the ILP-based algorithm of Gansner et al (1993) provides good results.

3. **Crossing minimization.** The crossing minimization step needs to determine the node order in each layer such that the number of edge crossings among the edges between two neighboring layers is minimized. Crossing minimization between two layers is again an NP-hard optimization problem (Eades and Wormald 1994). Several heuristics and (non-polynomial) exact algorithms for minimizing crossings between neighboring layers are known and evaluated in the experimental study by Jünger and Mutzel (1997). For computing node orders in all layers usually a layer-by-layer sweep method is applied that cycles through all layers and optimizes node orders in neighboring layers, but global multi-level optimization methods exist as well (Chimani et al 2011).
4. **Edge routing.** Once layers and node orders in each layer are fixed it remains to assign x-coordinates to all nodes (including dummy nodes) that respect the node orders and optimize the resulting edge shapes. A typical optimization goal is to draw edges as vertical as possible with few bends for long edges (or with at most two bends with a strictly vertical middle segment) while maintaining a minimum node spacing in each layer and avoiding node-edge overlaps. Algorithms for coordinate assignment either use mathematical programming (Sugiyama et al 1981; Gansner et al 1993) or a linear-time sweep method (Brandes and Köpf 2002).

Layouts for Specific Graph Classes

Previous sections covered visualization algorithms for general undirected and directed graphs. There is a variety of tailored algorithms for more specific graph classes, including prominent examples such as trees and planar graphs. These algorithms exploit structural properties when optimizing certain layout aesthetics. It is beyond the scope of this chapter to cover such specific graph classes and their visualization algorithms. Rather we refer to Rusu (2013) for a survey on tree layout and to Duncan and Goodrich (2013) and Vismara (2013) for surveys on planar graph layout. Further general books on graph drawing algorithms (Di Battista et al 1999; Tamassia 2013) contain chapters on specific layout algorithms.

Examples of Application – Graph Visualization Software

There are many software packages and frameworks for visualizing and drawing graphs. A non-exhaustive list of non-commercial ones are given below. We divide the list into two parts. The follow are packages that can handle relatively large graphs

- Cytoscape is a Java based software platform particularly popular with the biology community for visualizing molecular interaction networks and biological pathways.
- Gephi is a Java based network analysis and visualization software package which is able to handle static and

dynamic graphs. It is supported by a consortium of french organizations.
- Graphviz is one of the oldest open-source graph layout and rendering engines, developed at AT&T Labs. It is written in C and C++ and hosts layout algorithms for both undirected (multilevel spring-electrical and stress models) and directed graphs (layered layout), as well as various graph theory algorithms. Graphviz is incorporated in Doxygen, and available via R and Python.
- OGDF is a C++ class library for automatic layout of diagrams. Developed and supported by German and Australian researchers, it contains spring-electrical algorithms with fast multipole force approximation, as well as layered, orthogonal and planar layout algorithms.
- Tulip is a C++ framework originating from University of Bordeaux I for developing interactive information visualization applications. One of the goals of Tulip is to facilitate the reuse of components; it integrates OGDF graph layout algorithms as plugins.

The following are a few other free software each with its own unique merit, but not designed to work on very large graphs. For example,

- Dunnart is a C++ diagram editor developed at Monash University, Australia. Its unique feature is the ability to layout diagrams with constraints.
- D3.js is a popular JavaScript library for manipulating web documents based on data. It contains spring-electrical model based layout modules solved by a force directed algorithm. Since D3 works with SVG, it cannot scale beyond a few thousand graphical objects. However, a WebGL based JavaScript library VivaGraph could be used for larger graph visualization in the browser.

Future Directions for Research

Since the 1980's, a great deal of progress has been made in laying out graphs. The key enabling techniques are fast force approximations, the multilevel approach, and techniques for the efficient solution of the stress models. In addition, progress in the speed of GPU and graphics library also made it possible to display graphs with millions of nodes and edges. Furthermore, there has also been progress in abstracting the visual complexity of large graphs, for example, by grouping similar nodes together, and representing certain subgraphs such as cliques as a motif. But as graphs become increasingly large, complex, and time-dependent, there are a number of challenges to be addressed.

The Increasing Size of the Graphs

The size of graphs is increasing exponentially over the years (Davis and Hu 2011). Social networks are one area where we have graphs of unprecedented size. As of late 2017, Facebook, for example, has over 2.07 billion monthly active users, while Twitter has over 330 million. Other data sources may be smaller, but just as complex. For instance, Wikipedia currently has 5.5 million interlinked articles, while Amazon offers around 400 million items, with recommendations connecting each

item to other like-items. All these pale in comparison when we consider that there are 100 billion interconnected neurons in a typical human brain, and trillions of websites on the Internet. Each of these graphs evolves over time. Furthermore, graphs like these tend to exhibit the small-world characteristic, where it is possible to reach any node from any other in a small number of hops. All these features present challenges to existing algorithms (Leskovec et al 2009).

The unique features of these networks call for rethinking of the algorithms and visual encoding approaches. The large size of some networks means that finding a layout for such a graph can take hours even with the fastest algorithms available. There has been work in using GPUs and multiple CPUs (e.g., (Ingram et al 2009)) to speed up the computation by a factor of 10–100.

Even though state of the art graph layout algorithms can handle graphs with many millions of nodes and billions of edges, with so many nodes and edges, the traditional node-link diagram representation is at its limit. A typical computer screen only has a few million pixels, and we are running out of pixels just to render every node.

One solution is to use a display with high resolution, including display walls, and various novel ways to manipulate such a display (e.g., gesture or touch based control). But even the largest possible display is unlikely to be sufficient for rendering some of the largest social networks. One estimate puts human eyes as having a resolution of just over 500 million pixels. Therefore even if we can make a display with higher resolution, our eyes can only make partial use of such a display at any given moment.

Since the purpose of visualization is to help us to understand the structures and anomalies in the data, for very large graphs, it is likely that we need algorithms to discover structures and anomalies first (Akoglu et al 2010), and display these in a less cluttered form, but allowing the human operator to drill down to details when needed.

Node-link diagram representation, while most common, may not be the most user-friendly to the general public, nor is it the most pixel-efficient. Other representations, such as a combination of node-link diagrams and matrices (Henry et al 2007), have been proposed.

Large complex networks call for fast and interactive visualization systems to navigate around the massive amount of information. A number of helpful techniques for exploring graphs interactively, such as link-sliding (Moscovich et al 2009), have been proposed. Further research in this area, particularly at a scale that helps to make sense out of networks of billions of nodes and edges, are essential in helping us to understand large network data.

Finally, the stress model is currently the best algorithm for drawing graphs with predefined edge length. Improving its high computation and memory complexity is likely to remain an active area of research as well.

Time-varying and Complex Graphs

All the large and complex networks mentioned earlier are time-evolving. How to visualize such dynamic networks is an active area of research (Frishman and Tal 2007), both in

terms of graph layout (van Ham and Wattenberg 2008),and in displaying such graphs. For example, time-varying graph can be displayed as an animation, or as small multiples. Researchers have been studying Archambault and Purchase (2016) whether to use one form or the other, when measured in terms of preservation of the users' mental maps, and in improving comprehension and information recall. Dynamic network visualization will likely continue to be an area of strong interest.

Visualizing multivariate graphs is another challenging area. In a multivariate graph, each node and edge may be associated with several attributes. For example, in a social network, each node is a person with many possible attributes. Each edge represents a friendship which could entail multiple attributes as well, such as the type of friendship (e.g., school, work, church), and the length and strength of the friendship. Displaying these information in a way that helps our understanding of all these complex attributes is a challenge. It requires careful consideration of both visual design and interaction techniques (Wattenberg 2006; Kerren et al 2014).

References

Akoglu L, McGlohon M, Faloutsos C (2010) Oddball: Spotting anomalies in weighted graphs. In: Proceedings of the 14th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining (PAKDD 2010)

Archambault D, Purchase HC (2016) Can animation support the visualisation of dynamic graphs? Inf Sci 330(C):495–509

Brandes U, Köpf B (2002) Fast and simple horizontal coordinate assignment. In: Mutzel P, Jünger M, Leipert S (eds) Graph Drawing (GD'01), Springer-Verlag Berlin Heidelberg, LNCS, vol 2265, pp 31–44

Brandes U, Pich C (2007) Eigensolver methods for progressive multidimensional scaling of large data. In: Proc. 14th Intl. Symp. Graph Drawing (GD '06), LNCS, vol 4372, pp 42–53

Chimani M, Hungerländer P, Jünger M, Mutzel P (2011) An SDP approach to multi-level crossing minimization. In: Algorithm Engineering and Experiments (ALENEX'11), pp 116–126

Davis TA, Hu Y (2011) University of Florida Sparse Matrix Collection. ACM Transaction on Mathematical Software 38:1–18

Di Battista G, Eades P, Tamassia R, Tollis IG (1999) Algorithms for the Visualization of Graphs. Prentice-Hall

Duncan CA, Goodrich MT (2013) Planar orthogonal and polyline drawing algorithms. In: Tamassia R (ed) Handbook of Graph Drawing and Visualization, CRC Press, chap 7, pp 223–246

Eades P (1984) A heuristic for graph drawing. Congressus Numerantium 42:149–160

Eades P, Wormald NC (1994) Edge crossings in drawings of bipartite graphs. Algorithmica 11:379–403

Eades P, Lin X, Smyth WF (1993) A fast and effective heuristic for the feedback arc set problem. Information Processing Letters 47(6):319–323

Frishman Y, Tal A (2007) Online dynamic graph drawing. In: proceeding of Eurographics/IEEE VGTC Symposium on Visualization (EuroVis), pp 75–82

Fruchterman TMJ, Reingold EM (1991) Graph drawing by force directed placement. Software - Practice and Experience 21:1129–1164

Gansner ER, Koutsofios E, North SC, Vo KP (1993) A technique for drawing directed graphs. IEEE Trans Software Engineering 19(3):214–230

Gansner ER, Koren Y, North SC (2004) Graph drawing by stress majorization. In: Proc. 12th Intl. Symp. Graph Drawing (GD '04), Springer, LNCS, vol 3383, pp 239–250

Gansner ER, Hu Y, North SC (2013) A maxent-stress model for graph layout. IEEE Trans Vis Comput Graph 19(6):927–940

Hachul S, Jünger M (2004) Drawing large graphs with a potential field based multilevel algorithm. In: Proc. 12th Intl. Symp.

Graph Drawing (GD '04), Springer, LNCS, vol 3383, pp 285–295

van Ham F, Wattenberg M (2008) Centrality based visualization of small world graphs. Computer Graphics Forum 27(3):975–982

Healy P, Nikolov NS (2014) Hierarchical drawing algorithms. In: Tamassia R (ed) Handbook of Graph Drawing and Visualization, CRC Press, chap 13, pp 409–454

Henry N, Fekete JD, McGuffin MJ (2007) Nodetrix: a hybrid visualization of social networks. IEEE Transactions on Visualization and Computer Graphics 13:1302–1309

Hu Y (2005) Efficient and high quality force-directed graph drawing. Mathematica Journal 10:37–71

Ingram S, Munzner T, Olano M (2009) Glimmer: Multilevel mds on the gpu. IEEE Transactions on Visualization and Computer Graphics 15:249–261

Jünger M, Mutzel P (1997) 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. J Graph Algorithms Appl 1(1):1–25

Kamada T, Kawai S (1989) An algorithm for drawing general undirected graphs. Information Processing Letters 31:7–15

Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW, Bohlinger JD (eds) Complexity of Computer Computations, pp 85–103, DOI 10.1007/978-1-4684-2001-2_9

Kerren A, Purchase H, Ward MO (eds) (2014) Multivariate Network Visualization: Dagstuhl Seminar # 13201, Dagstuhl Castle, Germany, May 12-17, 2013, Revised Discussions, Lecture Notes in Computer Science, vol 8380, Springer International Publishing

Kruskal JB (1964) Multidimensioal scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika 29:1–27

Kruskal JB, Seery JB (1980) Designing network diagrams. In: Proceedings of the First General Conference on Social Graphics, U. S. Department of the Census, Washington, D.C., pp 22–50, bell Laboratories Technical Report No. 49

von Landesberger T, Kuijper A, Schreck T, Kohlhammer J, van Wijk JJ, Fekete JD, Fellner DW (2011) Visual analysis of large graphs: State-of-the-art and future research challenges. Computer Graphics Forum 30(6):1719–1749

Leskovec J, Lang K, Dasgupta A, Mahoney M (2009) Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. Internet Mathematics 6:29–123

Moscovich T, Chevalier F, Henry N, Pietriga E, Fekete J (2009) Topology-aware navigation in large networks. In: CHI '09: Proceedings of the 27th international conference on Human factors in computing systems, ACM, New York, NY, USA, pp 2319–2328

Ortmann M, Klimenta M, Brandes U (2016) A sparse stress model. In: Graph Drawing and Network Visualization - 24th International Symposium, GD 2016, Athens, Greece, September 19-21, 2016, Revised Selected Papers, pp 18–32

Purchase HC (1997) Which aesthetic has the greatest effect on human understanding? In: Proc. 5th Intl. Symp. Graph Drawing (GD '97), Springer-Verlag, LNCS, pp 248–261

Quigley A (2001) Large scale relational information visualization, clustering, and abstraction. PhD thesis, Department of Computer Science and Software Engineering, University of Newcastle, Australia

Rusu A (2013) Tree drawing algorithms. In: Tamassia R (ed) Handbook of Graph Drawing and Visualization, CRC Press, chap 5, pp 155–192

Sugiyama K, Tagawa S, Toda M (1981) Methods for visual understanding of hierarchical systems. IEEE Trans Systems, Man and Cybernetics SMC-11(2):109–125

Tamassia R (2013) Handbook of Graph Drawing and Visualization. Chapman & Hall/CRC

Tunkelang D (1999) A numerical optimization approach to general graph drawing. PhD thesis, Carnegie Mellon University

Vismara L (2013) Planar straight-line drawing algorithms. In: Tamassia R (ed) Handbook of Graph Drawing and Visualization, CRC Press, chap 6, pp 193–222

Walshaw C (2003) A multilevel algorithm for force-directed graph drawing. J Graph Algorithms and Applications 7:253–285

Wattenberg M (2006) Visual exploration of multivariate graphs. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, New York, NY, USA, CHI '06, pp 811–819