Visualizing large graphs

Yifan Hu[1] and Lei Shi[2]

[1] Yahoo Labs, 111 W 40th St, New York, NY 10018, USA. yifanhu@yahoo.com
[2] SKLCS, Institute of Software, Chinese Academy of Sciences, China.
shil@ios.ac.cn

Abstract. With the prevailence of big data, there is a growing need for algorithms and techniques for visualizing very large and complex graphs. In this article we review layout algorithms and interactive exploration techniques for large graphs. In addition we briefly look at softwares and datasets for visualization graphs, as well as challenges that need to be addressed.

1 Introduction

A graph is a mathematical notation describing relations among items. In this use, a node of the graph represents an item, and an edge exists between two nodes if the two corresponding items are related. Consider the following social network {*Farid–Aadil, Latif–Aadil, Farid–Latif, Carol–Andre, Carol–Fernando, Carol–Diane, Andre–Diane, Farid–Izdihar, Andre–Fernando, Izdihar–Mawsil, Andre–Beverly, Jane–Farid, Fernando–Diane, Fernando–Garth, Fernando–Heather, Diane–Beverly, Diane–Garth, Diane–Ed, Beverly–Garth, Beverly–Ed, Garth–Ed, Garth–Heather, Jane–Aadil, Heather–Jane, Mawsil–Latif*}. This social network tells us that "Farid" is a friend of "Aadil", "Latif" is a friend of "Aadil", and so on. However, staring at this mathematical notation of the social network for a moment, most of us may fail to find any structure in this social network. On the other hand, Fig. 1 shows a visualization of this graph. Immediately we can see that this graph has two clusters, and "Jane" and "Heather" are the two persons that connect these two social clusters together. This example illustrates that visualization of graphs can give us an overall sense of the data. It helps us find structures, locate anomalies, and formulate questions that can in turn be answered through interaction with the visualization and the underlining data.

In this article we look at algorithms and techniques for visualizing large graphs. For laying out large graphs, the scalability of the algorithm becomes very important, so is the ability of the algorithm in escaping from local minimum and achieving a globally optimal drawing. In addition, for visual exploration, it is crucial to reduce the complexity of the visual representation, using abstraction and compression techniques. Throughout the article we shall limit our attention to straight edge drawing of undirected graphs, mostly because scalable algorithms for these are more readily available. Our emphasis is on layout algorithms and interactive exploration techniques. Due to space limitation, no attempt is made to give a comprehensive review of the literature and history, nor of software and visualization systems. For further readings, the users are referred to [10,65,93].

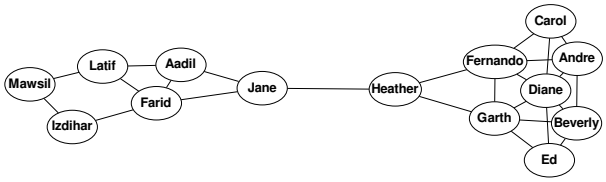


Fig. 1. Visualization of a small social network.

2 A Brief History

From the mid-20th century, due to demands from the military, transportation and communication industries, a growing number of graph problems, as well as the need to visualize graphs, have emerged. Given that drawing a small graph by hand is time-consuming, and medium to large graphs impossible, automatic generation of graph drawings became of interest, and were made possible by the increasing computing power.

In 1963, Tutte [97] proposed an algorithm to draw planar graphs by fixing the nodes on a face and placing the rest of the nodes at the barycenters of their neighbors. In 1984, Eades [25] proposed for the first time the spring-electrical model, and the force-directed solution framework, which was subsequently improved by Fruchterman and Reingold in 1991 [30]. Around the same time, Kamada and Kawai [61] proposed the stress model. The Fruchterman and Reingold force directed algorithm was extended to work on large graphs [12,44,56,84,96,104] through the multilevel (also known as multiscale) approach and fast force approximation techniques, and the convergence of the spring model was also improved by multilevel techniques [31,48].

For the remainder of this article we look at these and more recent algorithms, with emphasis on algorithms and techniques for visualizing *large* networks. The term large is relative to the computing power and memory available. In this paper, by large, we mean graphs of more than a few thousand vertices. Such large graphs occur in areas such as Internet mapping, social networks, biological pathways and genealogy. Large networks bring unique challenges in terms of scalability of the algorithm, and ability to find a globally optimal drawing.

Even with the most scalable and robust algorithm, not all graph can be aesthetically drawn in two or three-dimensional spaces. For example, many Internet graphs are of "small-world" nature [108]. Such graphs are difficult to layout in Euclidean space, and often require an interactive system to comprehend and explore. The size and complexity of such graphs necessitate techniques to simplify the visual representation. Such techniques include visual abstraction [37,85], compression [22,24,89], fisheye-like distortion and hyperbolic layout [37,53,69,77,76]. These topics are also discussed.

3 Algorithms for Large Graph Layout

An undirected graph G consists of a set of nodes (vertices) V, and a set of edges E, which are tuples of nodes. Denote by $|V|$ and $|E|$ the number of vertices and edges, respectively. If vertices i and j form an edge, we denote that $i \leftrightarrow j$, and call i and j neighboring vertices. Throughout the paper we use the terms network and graph interchangably.

A graph is traditionally visualized as a node-link diagram like Fig. 1. To get this diagram, we must assign a location to each node. This process is known as graph layout. We denote by x_i the location of vertex i in the layout. Here x_i is in d-dimensional Euclidean space. Typically $d = 2$ or 3.

The aim of laying out a graph (assigning a position to each node) is so that the resulting drawing gives an aesthetic visual representation of the connectivity information among vertices.

The most widely used techniques turn the problem of graph layout into one of finding a minimal energy configuration of a physical system. The two most popular methods in this category are the spring-electrical model [25,30], and the stress model [61].

We first introduce the spring-electrical model, and the multilevel approach and force approximation techniques that make this model feasible for large graphs. We note that the multilevel approach is not limited to the spring-electrical model. For convenience, we introduce it in the context of this model. We then discuss the stress model and the classical MDS algorithm (the strain model), and review recent efforts in making these models scalable. Finally we present the high-dimensional embedding and spectral algorithms. These two algorithms, though very fast, often do not give good drawings for real world graphs, and are included here for completeness.

3.1 Spring-electrical Model

The spring-electrical model was first introduced by Peter Eades in 1984 [25]. In this short paper, he wrote: "To embed a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system. The vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state... we make nonadjacent vertices repel each other..." This intuitive idea marks the conception of the spring-electrical model, as well as the force-directed solution framework now widely used for solving this and other virtual physical models.

Eades initially suggested to use logarithmic strength springs, and made the repulsive force inversely proportional to the distance for non-adjacent vertices. Later, Fruchterman and Reingold [30] modified the force formulae, so that the attractive spring force exerted on vertex i from its neighbor j is proportional to the squared distance between these two vertices,

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

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

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

The energy of this physical system [79] is

$$E(x) = \sum_{i \leftrightarrow j} \|x_i - x_j\|^3/(3K) - \sum_{i \neq j} K^2 \ln\left(\|x_i - x_j\|\right),$$

with its derivatives a combination of the attractive and repulsive forces.

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 vertex, and moving the vertices 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.

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 vertex i
10     }
11 until (||x − x^0|| < tol * K)
12 return x
```

This procedure can be enhanced by an adaptive step length updating scheme [14,56]. It usually works well for small graphs. For large graphs, this simple iterative procedure is prone to be trapped in one of the many local energy minima that exists in the space of all possible layouts. Instead, a multilevel approach can be be used to prevent local optima. Furthermore, force approximation techniques, often based on space decomposition schemes, can efficiently calculate the all-to-all electrical force so as to reduce the computational complexity from $|V|^2$ to $O(|V| \log |V| + |E|)$.

Fast force approximation Each iteration of the force-directed algorithm (Algorithm 1) involves two loops. The outer loop iterates (lines 5-10) over each of the $|V|$ vertices. Of the two inner loops (lines 7 and 8), the latter involves calculation of $|V| - 1$ repulsive forces. Thus the outer and inner loops together have an overall computational complexity of $O(|V|^2)$.

Fruchterman and Reingold [30] proposed to reduce the complexity by partitioning the space into grid cells, and only calculated repulsive forces for vertices in neighboring grid cells. This however ignores far away nodes and can introduce large errors. A better approach is to use a technique widely known in simulation of the n-body problem in physics. This technique, proposed by Barnes & Hut [8], approximates the repulsive forces in $O(n \log n)$ time with good accuracy, but does so without ignoring long range forces. It works by treating groups of far away vertices as supernodes, using a nested space decomposing data structure. This idea was adopted by Tunkelang [96] and Quigley [84]. They both used a quadtree (or octree in 3D) data structure.

A quadtree forms a recursive grouping of vertices (Figure 2 (a)), and can be used to efficiently approximate the repulsive force. When calculating the repulsive force on a vertex i, if a group of vertices, S, lies in a square that is sufficiently "far" from i, the whole group can be treated as a supernode. Otherwise we traverse down the tree and examine the four sibling squares. Figure 2 (right) shows all the supernodes (the squares) and the vertices these supernodes consist of, with reference to vertex i located at the top-middle part of the graph. In this case there are 936 vertices, and 32 supernodes.

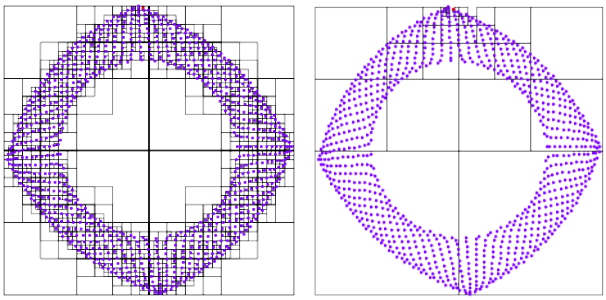


Fig. 2. An illustration of the quadtree data structure. Left: the overall quadtree. Right: supernodes with reference to a vertex at the top middle part of the graph, with $\theta = 1$.

Under a reasonable assumption [8,83] of the distribution of vertex positions, it can be proved that building the quadtree takes a time complexity of $O(|V| \log |V|)$. Finding all the supernodes with reference to a vertex i can be done in a time complexity of $O(log|V|)$. Overall, using a quadtree structure to approximate the repulsive force, the complexity for each iteration of the force-directed Algorithm 1 is reduced from $O(|V|^2)$ to $O(|V| \log |V|)$. This force ap-

proximation scheme can be further improved by considering force approximation at supernode-supernode level instead of vertex-supernode level [15].

The multipole method [42] is another force approximation algorithm with the same $O(|V| \log |V|)$ complexity, but it achieves this complexity independent of the distribution of vertex positions. It also relies on a quad-tree space decomposition. Hachul and Jünger applied this force approximation to graph drawing [44] in the FM^3 code.

Multilevel approach While fast approximation of long range forces resolves the quadratic complexity for the repulsive force calculation, the force-directed algorithm uses the steepest decent process and re-positions one vertex at a time to minimize the energy locally. Due to the fact that this physical system of springs and electrical charges can have many local minimum configurations, applying the force-directed algorithm to a random initial layout is unlikely to yield a global optimal final layout. A multilevel approach can overcome this limitation. In this approach, a sequence of smaller and smaller graphs are generated from the original graph, each captures the essential connectivity information of its parent. Global optimal layout can be found much more easily on small graphs. A good layout for a coarser graph is thus always used as a starting layout for its parent. From this initial layout, further refinement is carried out to achieve the optimal layout of the parent.

Multilevel approaches have been used in many large-scale combinatorial optimization problems, such as graph partitioning [43,51,106], matrix ordering [57,86], the traveling salesman problem [103], and were proved to be a useful meta-heuristic tool [105]. They were later used in graph drawing algorithms [31,45,48,104], sometimes under the name "multiscale". Note that a multilevel approach is not limited to the spring-electrical model, but for convenience we are introducing it in the context of this model.

A multilevel approach has three distinctive phases: coarsening, coarsest graph layout, and prolongation and refinement. In the coarsening phase, a series of coarser and coarser graphs, $G^0 = G, G^1, \ldots, G^l$, are generated, each coarser graph G^{k+1} encapsulates the information needed to layout its parent G^k, while containing fewer vertices and edges. The coarsening continues until a graph with only a small number of vertices is reached. The optimal layout for the coarsest graph can be found cheaply. The layouts on the coarser graphs are recursively prolonged to the finer graphs, with further refinement at each level.

Graph coarsening and initial layout is the first phase in the multilevel approach. There are a number of ways to coarsen an undirected graph. One often used method is based on edge collapsing (EC) [43,51,106]. In this scheme, a maximal independent edge set (MIES) is selected. This is a maximal set of edges, with no edges incident to the same vertex. The vertices corresponding to this edge set form a maximal matching. Each edge, and its corresponding pair of vertices, are coalesced into a new vertex. Figure 3 illustrates MIES and the result of coarsening using edge collapsing.

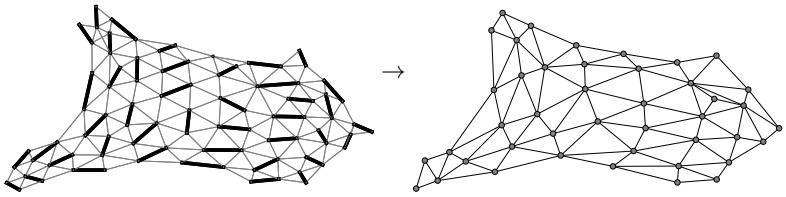


Fig. 3. An illustration of the edge collapsing based graph coarsening. Left: original graph with 85 vertices. Edges in a maximal independent edge set are thickened. Right: a coarser graph with 43 vertices resulted from coalescing thickened edges.

Alternatively, coarsening can be performed based on a maximal independent vertex set (MIVS) [7]. This is a maximal set of vertices such that no two vertices in the set are connected by an edge in the graph. Edges of the coarser graph are formed through linking two vertices in the maximal independent vertex set by an edge if their distance apart is no greater than three. This coarsening strategy is also known as filtering [31], and was applied to distance based embedding. Bartel *et al.* [9] conducted an experimental study, comparing many different strategies for coarsening, as well as components for other phases of the multilevel process.

Coarsest graph layout is carried out at the end of the recursive coarsening process. Coarsening is performed repeatedly until the graph is very small; at that point we can layout the graph using a suitable algorithm, for example, the force-directed Algorithm 1. Because the graph on the coarsest level is very small, it is likely that it can be laid out optimally.

The prolongation and refinement step is the third phase in a multilevel procedure. The layout on the coarser graphs are recursively interpolated to the finer graphs, with further refinement at each level. Because the initial layout on each level is derived from a good layout on the coarser level, it is much more likely that a globally optimal layout can be achieved.

Row "spring electrical" of Figure 4 shows drawings of two graphs using this multilevel force-directed algorithm [56]. The drawings are of good quality for both `dw256A`, a mesh-like graph, and `qh882`, a sparser graph.

Robust coarsening: while the coarsening schemes based on maximal independent edge or vertex sets work well for many graphs, for some real world graphs, these schemes are not suitable. One requirement for a good coarsening scheme is that the coarsening should reduce the vertex/edge size of the graph by no less than a constant factor. A typical graph that creates a problem for the standard coarsening schemes is the following. This graph has a few high degree nodes, and many other nodes randomly connected to one or a few of the high degree nodes. Such a graph is an extension of the star-graph, and is often seen in social networks, where high degree vertices are celebrities, and low degree vertices are followers. This types of graphs pose a challenge for a coarsening scheme based on edges collapsing, because a randomly select edge is likely to be connected to one of the high degree nodes, thus prevents all the edges connected to that

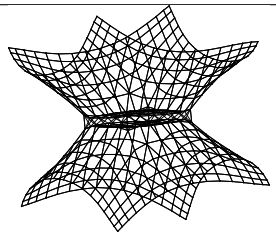
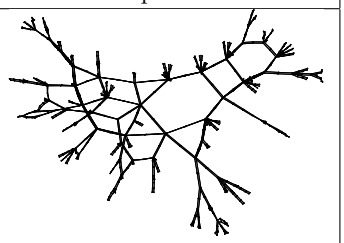
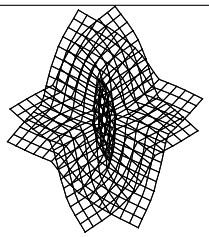
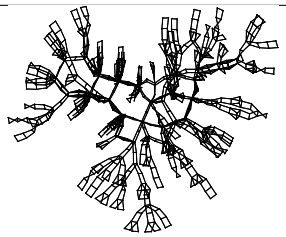
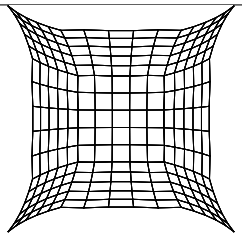
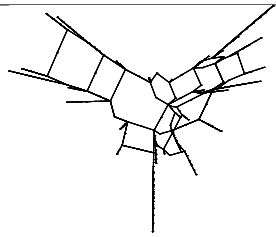
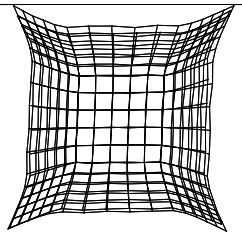
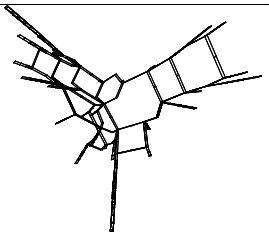
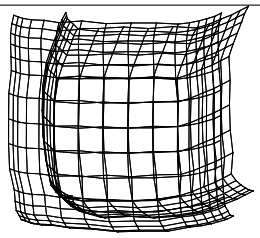
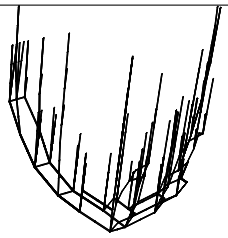
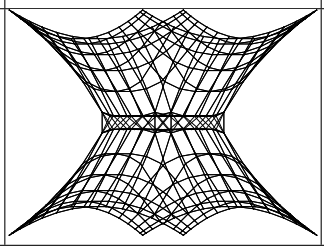
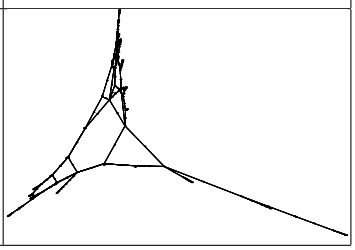
Algorithms	dw256A	qh882
spring electrical		
stress		
classical MDS		
Pivot MDS		
HDE		
Hall's		

Fig. 4. Result of some of the algorithms applied to two graphs, `dw256A` and `qh882`.

high degree node from being chosen for collapsing. As a result the maximal independent edge set only contains very few edges, and coarsening based on MIES barely reduces the size of the graph.

One solution proposed in [19] is to find vertices that share the same neighbors, known as structural equivalent vertices. On a social network these vertices are people who share the same, or very similar, friends. These vertices are matched in pairs. The standard MIES scheme is then used to match the remaining unmatched vertices. Finally the matched vertices are merged to get the coarsened graph. This scheme is able to overcome the slow coarsening problem associated with graphs having star-graph likes sub-structures.

By combining the aforementioned fast force calculation techniques and the multilevel approaches, efficient implementations [44,56] of the spring-electrical model are capable of handling graphs of millions of vertices and edges [55]. Figure 5 gives some examples using such an implementation.

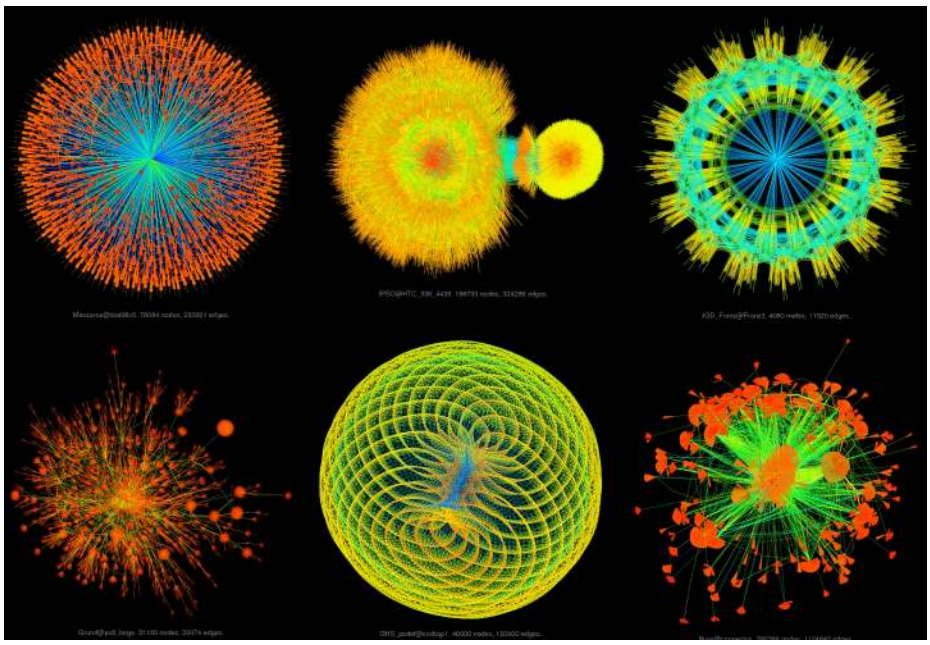


Fig. 5. Example drawings of large graphs

3.2 Stress and Strain Models

In many real life applications, edges of a graph often have quantitative information associated. For example, vertices are connected if the objects they represent

are similar, and each edge has a similarity measure attached. It is useful to layout such graphs so that the edge lengths reflect this information. While the spring-electrical model can be made scalable, it does have the limitation of not encoding edge length explicitly in the model. It is possible to assign weaker attractive force and stronger repulsive force for longer edges, but such treatment is not as principled as the following stress model.

Stress model The stress model assumes that there are springs connecting all pairs of vertices in the graph, with the ideal spring length equal to the predefined edge length. The stress energy 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 vertices i and j. The layout that minimizes the above stress energy is an optimal layout of the graph according to this model. In the stress model w_{ij} is a weight factor. A typical choice is $w_{\mathrm{ij}} = 1/{d_{\mathrm{ij}}}^2$. With this choice, (3) can be written as $\sum_{i \neq j} \left(\|x_i - x_j\|/d_{\mathrm{ij}} - 1 \right)^2$, thus this stress energy measures the relative difference between the actual edge length and ideal edge length. If $w_{\mathrm{ij}} = 1/d_{\mathrm{ij}}$, the resulting embedding is known as Sammon Mapping.

The stress model has its root in Multidimensional Scaling (MDS) [66,67], and the term MDS is sometimes used to describe the embedding process based on the stress model. Note that in graph drawing, only the lengths of the edges are known. If they are not given, then we assume that they equal to one (alternatively, Gansner et al. [38] proposed to set the edge length to the total number of non-common neighbors of the two end vertices). For non-adjacent vertex pairs i and j, typically d_{ij} is defined as the shortest path distance between them. This practice dates back at least to 1980 in social network layout [13], and in graph drawing using classical MDS[67], even though it is often attributed to Kamada and Kawai [61] in the graph drawing literature.

There are several ways to minimize (3). Kamada and Kwai [61] proposed to minimize the energy function (3) by using Newton's method on the coordinates related to one vertex at a time. In recent years stress majorization technique [38] became a preferred way to minimize the stress model due to its robustness.

Stress-majorization [38]: consider the expanded form of the stress energy (3),

$$stress(x) = \sum_{i \neq j} \left(w_{\mathrm{ij}} \|x_i - x_j\|^2 - 2 d_{\mathrm{ij}} w_{\mathrm{ij}} \|x_i - x_j\| + w_{\mathrm{ij}} {d_{\mathrm{ij}}}^2 \right)$$

Using the Cauchy-Schwartz inequality, the cost function is bounded by

$$stress(x) \leq g(x, y) = \sum_{i \neq j} \left(w_{\mathrm{ij}} \|x_i - x_j\|^2 - 2 d_{\mathrm{ij}} w_{\mathrm{ij}} \frac{(x_i - x_j)^T (y_i - y_j)}{\|y_i - y_j\|} + w_{\mathrm{ij}} {d_{\mathrm{ij}}}^2 \right),$$

with the bound tight when $y = x$. The idea of stress majorization is to minimize a sequence of quadratic functions $g\left(x, y^k\right)$ with regard to x. Here y^k is the result of minimizing $g\left(x, y^{k-1}\right), k = 1, 2, ...$ Initially $y^0 = x^0$.

The minimum of the quadratic function $g(x, y)$ with regard to x, derived by setting $\partial_{x_i} g(x, y) = 0$, is computed as

$$L_w x = L_{w,d}\ y \tag{4}$$

where the weighted Laplacian matrix L_w is positive semi-definite and has elements

$$(L_w)_{\mathrm{ij}} = \begin{cases} \sum_{l \neq i} w_{\mathrm{il}}, & i = j \\ -w_{\mathrm{ij}}, & i \neq j \end{cases} \tag{5}$$

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

$$(L_{w,d}\ y)_{\mathrm{i}} = \sum_{l \neq i} w_{\mathrm{il}}\ d_{\mathrm{il}}(\ y_i - y_l) / \left\| y_i - y_l \right\|. \tag{6}$$

In summary, the process of finding a minima of (3) becomes that of solving a series of linear systems (4), with the solution x served as y in the next iteration. This iterative process was found to be more robust than the Kamada and Kwai algorithm [61], although it still benefits from a good initial layout. Row "stress" of Figure 4 gives drawings given by the stress majorization algorithm. It performed very well on both graphs.

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 have a computational complexity at least quadratic to the number of nodes. In recent years there have been attempts at developing more scalable ways of drawing graphs that still satisfy the user specified edge length as much as possible. We will discuss some of these after we introduce the strain model.

Strain model (classical MDS) The strain model, also known as classical MDS [94], predates the stress model. Classical MDS tries to fit the inner product of positions, instead of the distance between points. Specifically, assume that the final embedding is centered around the origin. Furthermore, assume that in the ideal case, the embedding fits the ideal distance exactly: $\|x_i - x_j\| = d_{ij}$. It is then easy to prove [109] that the product of the positions, $x_i^T x_j$, can be expressed as the squared and double centered distance,

$$x_i^T x_j = -1/2 \left(d_{ij}^2 - \frac{1}{|V|} \sum_{k=1}^{|V|} d_{kj}^2 - \frac{1}{|V|} \sum_{k=1}^{|V|} d_{ik}^2 + \frac{1}{|V|^2} \sum_{k=1}^{|V|} \sum_{l=1}^{|V|} d_{kl}^2 \right) = b_{ij}. \tag{7}$$

In real data, it is unlikely that we can find an embedding that fits the distances perfectly, but we would still expect b_{ij} to be a good approximation of

$x_i^T x_j$. Therefore we try to find an embedding that minimizes the difference between the two,

$$\min_X \ ||X^T X - B||_F, \tag{8}$$

where X is the $|V| \times d$ dimensional matrix of x_i's, B is the $|V| \times |V|$ symmetric matrix of b_{ij}'s, and $||.||_F$ is the Frobenius norm. Denote the eigen-decomposition of B as $B = Q^T \Lambda Q$, then the solution to (8) becomes $X = \Lambda_d^{1/2} Q$, where Λ_d is the diagonal matrix of Λ, with all but the d-largest eigenvalues on the diagonal set to zero. In other words, strain model works by finding the top d eigenvectors of B, and uses that, scaled by the eigenvalues, as the coordinates. Because the strain model does not fit the distance directly, edge length in the layout may not fit the length specified by the user as well as the stress model. Nevertheless layout from the strain model can be used as a good starting point for the stress model. Row "classical MDS" of Figure 4 gives drawings using classical MDS. It performed better on the mesh-like `dw256A` graph than on the sparser `qh882` graph. On the latter it gives a drawing with many vertices close to each other, making details of the graph unclear, even though it captures the overall structure well.

MDS for large graphs In the stress model (as well as the strain model), the graph distances between all pairs of vertices have to be calculated, which necessitates an all-pairs shortest path calculation. Using Johnson's algorithm, this needs $O(|V|^2 \log |V| + |V||E|)$ computation time, and a storage of $O(|V|^2)$. Solution of the dense linear systems takes even longer time than the formation of the distance matrix. Therefore for very large graphs, the stress model is computationally expensive and memory prohibitive. A number of attempts have been made to approximately minimize the stress energy or to solve the strain model.

The **multiscale algorithm** of Hadany and Harel [45] improved the Kamada and Kwai [61] solution process by speeding up its convergence, Harel and Koren[49] improved that further by coarsening with k-centers. A multiscale algorithm of Gajer et al. [31] applies the multilevel approach in solving the stress model. In the GRIP algorithm [31], graph coarsening is carried out through vertex filtration, an idea similar to the process of finding a maximal independent vertex set. A sequence of vertex sets, $V^0 = V \subset V^1 \subset V^2, \ldots, \subset V_L$, is generated. However, coarser graphs are not constructed explicitly. Instead, a vertex set V^k at level k of the vertex set hierarchy is constructed so that distance between vertices is at least $2^{k-1} + 1$. On each level k, the stress model is solved by a force-directed procedure. The spring force on each vertex $i \in V^k$ is calculated by considering a neighborhood $N^k(i)$ of this vertex, with $N^k(i)$ the set of vertices in level k – chosen so that the total number of vertices in this set is $O(|E|/|V^k|)$. Thus the force calculation on each level can be done in time $O(|E|)$. It was proved that with this multilevel procedure and the localized force calculation algorithm, for a graph of bounded degree, the algorithm has close to linear computational and memory complexity. However in the actual implemen-

tation, at the finest level, GRIP reverts back to the spring-electrical model of Fruchterman and Reingold [30], making it difficult to assess whether GRIP can indeed solve the stress model well.

LandmarkMDS [20] approximates the result of the classical MDS by choosing $k << |V|$ vertices as landmarks, and calculating a layout of these vertices using the classical MDS, based on distances among these vertices. The positions for the rest of vertices are then calculated by placing them at the weighted barycenter, with weights based on distances to the landmarks. So essentially classical MDS is applied to a $k \times k$ submatrix of the $|V| \times |V|$ matrix B. The complexity of this algorithm is $O(k|E| + k^2)$, and only $O(k|V|)$ distances need to be stored.

PivotMDS [12], on the other hand, takes a $|V| \times k$ submatrix C of B. It then uses the eigenvectors of CC^T as an approximation to those of B. The eigenvectors of CC^T is found via those of the $k \times k$ matrix C^TC. By an algebraic argument, if v is an eigenvector of C^TC, then

$$CC^T(Cv) = C(C^TCv) = \lambda(Cv),$$

hence Cv is an eigenvector of CC^T. Therefore PivotMDS proceeds by finding the largest eigenvectors of this smaller $k \times k$ matrix C^TC, then projects back to $|V|$-dimensional space by multiplying them with C. It uses these projected eigenvectors, scaled by the inverse of the quartic root of the eigenvalues, as coordinates. Using this technique, the overall complexity is similar to LandmarkMDS, but unlike LandmarkMDS, PivotMDS utilizes distances between landmark vertices (called pivots) and all vertices in forming the C matrix. In practice, PivotMDS was found to give drawings that are closer to the classical MDS than LandmarkMDS. It is extremely fast when used with a small number (e.g., $k \approx 50$) of pivots.

It is worth pointing out that, for sparse graphs, there is a limitation in both algorithms. For example, if the graph is a tree, and pivots/landmarks are chosen to be non-leaves, then two leaf nodes that have the same parent will have exactly the same distances to any of the pivots/landmarks, consequently their final positions based on these algorithms will also be the same. This problem may be alleviated to some extent by utilizing the layout given by these algorithms as an initial placement for a sparse stress model [13,38].

The **MaxEnt** algorithm [36] is based on the following argument. The user only specifies the length of edges. The stress model (3) assumes that the unknown distance between non-neighboring vertices should be the shortest graph-theoretic distance. This is reasonable, but does add artificial information that is not given in the input. In addition, calculating all-pairs shortest distances for large graphs is costly anyway. On the other hand, specifying only the length of edges leaves too many degrees of freedom in possible node placement. A reasonable yet efficient way to satisfy these degrees of freedom is thus needed. MaxEnt proposed to resolve the extra degrees of freedom in the node placement by maximizing a notion of entropy that is maximized when vertices are uniformly spread-out, subject to the edge length constrains. In the final implementation, MaxEnt min-

imized the following sparse stress function defined on the edges, together with a penalty term that helps to spread out vertices:

$$\min \sum_{\{i,j\}\in E} w_{ij} \left(\|x_i - x_j\| - d_{ij}\right)^2 - \alpha \sum_{\{i,j\}\notin E} ln \|x_i - x_j\| . \tag{9}$$

It turns out that a solution technique similar to stress-majorization can be employed, which involves solving repeatedly

$$L_w x = L_{w,d}\ x + \alpha\ b(x),$$

except that the matrices involved are all sparse, and there is an additional term $b(x)$ which is a sum of repulsive forces among non-neighboring vertices, and can be approximated efficiently using the fast force approximation technique discussed earlier. MaxEnt was found to be more efficient than the full stress model. Although it is not as fast as PivotMDS, MaxEnt does not suffer from the same limitation as PivotMDS on sparse graphs.

The **MARS** algorithm [62] attempts to approximate the full stress model. The full stress model is not scalable because of the need for the all-pairs shortest paths calculation. In addition, the dense Laplacian matrix (5) involved also make the memory requirement quadratic to the number of vertices. The idea of MARS is that although this Laplacian typically has slow decaying eigenvalues, when the diagonal part is zeroed-out, the off-diagonal matrix often has a quick decaying eigen-distribution, thus can be approximated well by dropping most of the smaller eigenvalues and approximating by a low rank singular-value decomposition (SVD). To avoid using every entry of the matrix to form the approximation, MARS utilized a result of Drineas et al. [21], which states that the extreme singular values and vectors of a matrix with quick decaying eigenvalues can be approximates well by the singular values and left singular vectors of a $|V| \times k$ matrix. Columns of the matrix are sampled from the original matrix with a probability proportional to the norm of the columns. This sampling technique works well in the context of the Laplacian (5), because computing a column of that matrix only requires a single-source shortest paths calculation. Thus getting k-columns requires only k applications of Dijstra's algorithm, with a complexity of $O(k(|E| + |V| \log |V|))$.

Once the SVD approximation for the off-diagonal matrix is formed, some algebraic manipulation is required to use this approximation to solve the linear system via pseudo-inverse. An additional challenge is that the right-hand-side of (4), shown in details in (6), also involved shortest path distances for all pairs of vertices. Fortunately, when $w_{ij} = 1/d_{ij}$, the product $w_{ij}d_{ij}$ is 1, and equation (6) becomes a sum of unit vectors pointing from all other vertices to vertex i. This can be approximated well using the fast force approximation techniques discussed earlier.

COAST [35] reformulates the stress model into two parts so that fast convex optimization techniques can be applied. The energy function it minimizes is

$$\min \sum_{\{i,j\}\in E} w_{ij} \left(\|x_i - x_j\|^2 - d_{ij}^2 \right)^2 - \alpha \sum_{\{i,j\}\notin E} \|x_i - x_j\|^2 . \tag{10}$$

This model is somewhat comparable to that of MaxEnt (equation (9)), in that the first part sums over the edges and the second part over non-neighboring vertices. The main difference is that COAST added a square to the two terms in the first part, and made the second part quadratic. This made the energy function quartic. When this function is expanded, all non-constant terms are of the form $x_i^T x_j$. Defining these as new artificial variables, the energy function can be converted to a quadratic function of the $|V|^2$ new variables, subject to the constraint that these artificial variables sit in a semi-definite cone. The complexity can be reduced further if we assume that x_i can be expressed as a linear combination of the smaller eigenvectors of the Laplacian, a reasonable assumption because positions of vertices that are closely connected should be smoothly varying. In the end, COAST solves a convex optimization problem with a small number of variables, and is comparable in speed to MaxEnt.

3.3 High-Dimensional Embedding

The high-dimensional embedding (HDE) algorithm [50] finds coordinates of vertices in a k-dimensional space, then projects back to two or three dimensional space.

First, a k-dimensional coordinate system is created based on k-centers, where the k-centers are chosen as in LandmarkMDS/PivotMDS. The graph distances from each vertex to the k-centers form a k-dimensional coordinate system. The $|V|$ coordinate vectors form an $|V| \times k$ matrix Y, where the i-th row of Y is the k-dimensional coordinates for vertex i. The dimension of this coordinate system is then reduced from k to d ($d = 2$ or 3) by principal component analysis, which finds d largest eigenvectors v_i of $Y^T Y$ (Y is first normalized so that the sum of each column is zero), and uses Yv_i as the coordinates of the final embedding.

Clearly, HDE has many commonalities to PivotMDS. Each utilizes k-centers, and each finds the largest eigenvectors of a $k \times k$ dimensional matrix derived by multiplying the transpose of a $|V| \times k$ matrix with itself. The main difference is that in high-dimensional embedding this $|V| \times k$ matrix consists of distances to the k-centers, while in PivotMDS this matrix consists of distances squared and double centered (see (7)). Due to its reliance on distances fro the k-centers, high-dimensional embedding may suffer from the same issue as PivotMDS and LandmarkMDS on sparse graphs. In practice it tends to do worse than PivotMDS on such graphs. Row "HDE" of Figure 4 gives drawings using HDE. It performs particularly badly on `qh882` graph, with vertices close to each other, obscuring many details.

3.4 Algorithms Based on the Spectral Information of the Laplacian

In 1970, Hall [46] remarked that many sequencing and placement problems could be characterized as finding locations of points which minimize the weighted sum of square distances. In our notation, what he proposed was to minimize

$$\sum_{i \leftrightarrow j} w_{ij} \left\| x_i - x_j \right\|^2, \text{ subject to } \sum_{k=1}^{|V|} x_k^2 = 1$$

where x_i is the 1-dimensional coordinate value for vertex i. The objective function can be written as

$$\sum_{i \leftrightarrow j} w_{ij} \left\| x_i - x_j \right\|^2 = x^T L_w x,$$

with $x = \{x_1, x_2, ..., x_{|V|}\}$. Here L_w is the weighted Laplacian matrix (5).

The solution x of the minimization problem is the eigenvector corresponding to the smallest positive eigenvalue of the weighted Laplacian L_w. We can achieve a 2-dimensional layout by taking the two eigenvectors corresponding to the two smallest positive eigenvalues. Row "Hall's" of Figure 4 gives drawings employing Hall's algorithm. It performed reasonably well on the mesh like `dw256A` graph, but is close to useless on the sparser `1138_bus` graph.

Koren et al. [64] proposed an extremely fast algorithm for calculating the two extreme eigenvectors using a multilevel algorithm. The algorithm is called ACE (Algebraic multigrid Computation of Eigenvectors). Using this algorithm, they were able to layout graphs of millions of nodes in less than a minute. However, the fundamental weakness of Hall's algorithm on sparse graphs remains.

Table 6 summarizes most of the algorithms discussed in this section. Except the full stress model and the classical MDS (strain model), all the other algorithms can work on relatively large graphs.

4 Visual Abstraction of Large Graphs

As we have discussed so far, researchers made tremendous progress in scaling classical drawing algorithms to large graphs. This allows, for the first time, users to view a large graph with millions of nodes in a single picture. The users can quickly match patterns and discover insights from large graph drawings, with the help of the vast visual bandwidth and parallel processing capability of the human eyes and brain. However, a full exploitation of human's visual perception power requires certain drawing aesthetics on the graph visualization [93], and these aesthetics can be hard to satisfy when drawing large graphs. As the graph size increases, edge crossings appear quickly and then node occlusion becomes significant, hiding potential patterns that can be in existence.

We look at one important class of solution in this section, the graph abstraction methods. Generally, these methods apply graph simplification algorithms

Fig. 6. Algorithms discussed in this section and their complexity, quality, edge (whether edge length is taken into account), and free software that implement the algorithms. Rows are sorted by computational complexity and practical speed. Further details on software are given in the "Software and Data Sets" section. Note that k is typically a much smaller number than $|V|$, e.g., $k = 50$.

algorithm	computational complexity	quality	edge	software
Hall's Algorithm [a]	$O(\|E\|)$	poor	no	-
HDE	$O(k\|E\| + k^2)$	poor	no	-
Landmark MDS	$O(k\|E\| + k^2)$	medium	yes	-
pivotMDS	$O(k^2\|V\| + k\|E\| + k^2)$	good[c]	yes	[73]
spring electrical [b]	$O(\|V\|log\|V\| + \|E\|))$	good	no	[39,40,81]
MaxEnt	$O(\|V\|log\|V\| + \|E\|)$	good	yes	-
MARS	$O(k^3 + k(\|V\|log\|V\| + \|E\|) + k^2\|V\|)$	medium	yes	-
COAST	$(k^2\|v\| + k^4\|E\| + k^{4.5})$	good	yes	-
classical MDS	$O(\|V\|^2 \log\|V\| + \|V\|\|E\|)$	medium	yes	-
(full) stress	$O(\|V\|^2 \log\|V\| + \|V\|\|E\|)$	good	yes	[39,81]

[a]: with a multilevel implementation [64]. [b]: with fast force approximation. [c]: except on graphs with tree like sub-structures.

to reduce a large graph into smaller and simpler abstractions. The graph abstraction can be visualized by classical drawing algorithms. The layout normally fits better to the graph aesthetics than that of the original large graph. These abstractions are sometimes called the overview of large graphs. To access details and local patterns of the large graph, navigation methods are introduced to interactively explore graph abstractions. In this section, we also look at another class of relevant methods which transform the graph view for visual simplicity, normally without affecting the underlying graph topology.

4.1 Topology Compression

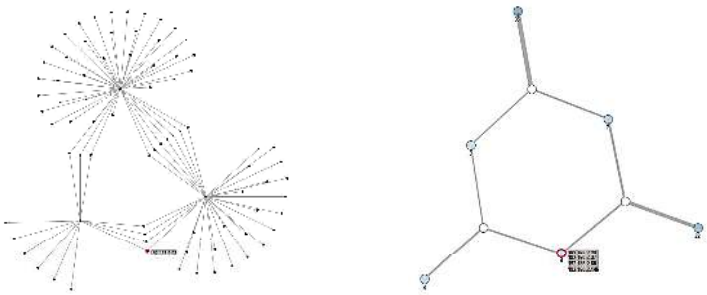


Fig. 7. Lossless compression a small network traffic graph [89].

The most straightforward method to simplify large graphs is to exploit the redundancy in graph topology [89]. As shown in Figure 7, a small traffic graph with 80 nodes can be compressed into a 9-node super graph without losing topology information. The underlying algorithm involves the classical concept of structural equivalence on graph [72]. Strictly, given a large graph G with adjacency matrix $W = \{w_{ij}\}_{i,j=1}^{n}$, the abstraction method by structural equivalence groups the nodes with the same row vector together into a super node, where the row vector for the ith node is $R_i = \{w_{ij}\}_{j=1}^{n}$. In the same approach, the directed and weighted large graphs are compressed by extending the adjacency matrix definition. With special treatments, cliques of nodes and fuzzy node groups can be supported.

Several other literature have proposed to compress the large graph with similar ideas, such as PhraseNet [101], graph coarsening [19] and motif simplification [22]. Specially in [22], Dunne and Shneiderman designed several visual glyphs to encode fan, connector and clique patterns extracted from large graphs (Figure 8). To further increase the compression rate over large graphs, more sophisticated algorithms have been studied. Modular decomposition [82] is a method to decompose the graph into a module tree. The nodes in each module have the same relationship to all the nodes outside the module. Power graph analysis [24] introduced a further relaxation that allows the edges to cross module boundaries.

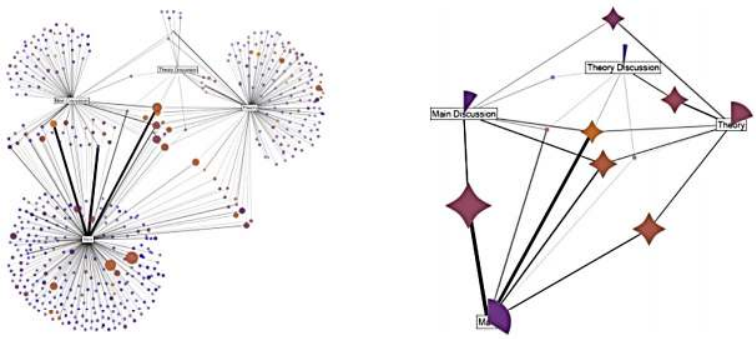


Fig. 8. Motif simplification of a network of wiki edits [22].

Though lossless compression of large graphs can be critical to understand the details of the original graph, it is extremely difficult to achieve on real graphs with small world nature. The compressed graph often retains a size comparable to the original graph and is hard to draw in good aesthetics. In contrast, graph clustering provides a general method to obtain a higher degree of compression for large graphs. The graph clustering algorithms (aka community detection) are well-studied in many research communities [27]. These algorithms depend on cohesive measures or criteria among graph nodes, such as the modularity [78] and ratio association [47]. The visualization methods on the top often apply clus-

tering algorithms recursively to generate hierarchical abstraction of large graphs [85,5,1,88]. Quigley and Eades proposed to use the geometric clustering to generate graph views with multiple levels of abstraction [85]. Lei et al. introduced the modularity-based hierarchical graph clustering to visually summarize the large graph within certain cluster depth [88]. Abello et al. designed Ask-GraphView in an overview+detail approach [1]. The overall cluster hierarchy tree is combined with a clustered view of the focused subgraph to allow exploration of the entire large graph. The general steps for the node clustering based large graph visualization is illustrated in Figure 9.

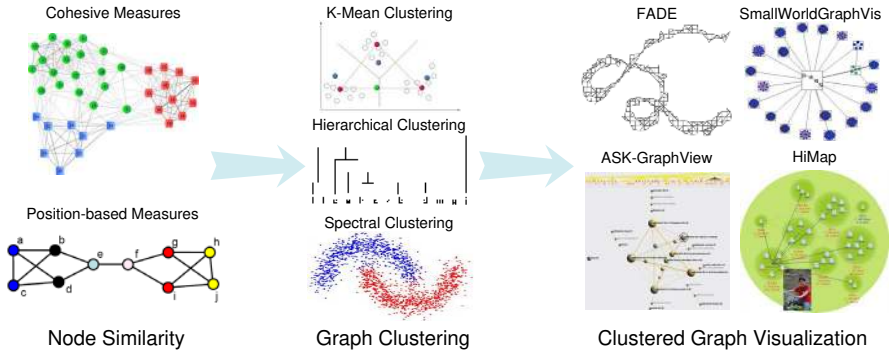


Fig. 9. Large graph visualizations through node clustering [85,5,1,88].

4.2 Semantic Abstraction

With the advent of the Internet and social media, many large graphs extracted from the information network are rich in context. This context can be externalized as node/edge attributes in graph, such as user's profile in a social graph and paper's venue in a citation graph. The semantic abstraction of graphs mainly depends on these attributes to create a super graph to explain or complement the original large graph visualization.

As one of the first proposal in this area, Wattenberg invented PivotGraph [107] (Figure 10(a)). PivotGraph developed a few operations on multivariate graphs. The primary roll-up operation pivots the graph nodes with the same value on one or two attributes into node aggregations. These attributes can be picked manually to generate different semantic abstractions. Combining these operations and a grid-based 2D layout, PivotGraph supports powerful attribute-centric analysis over large graphs. In OntoVis [87], Shen et al. proposed a method to abstract heterogeneous social networks based on their ontology graph. For attribute-based analysis, the graph can be filtered by selected nodes in the ontology graph. On structural abstraction, OntoVis provides methods such as degree-one node and duplicate path reductions.

Recently, OnionGraph was proposed to combine the advantage of both semantic and topology abstraction of large graphs into the method of heterogeneous abstraction [90] (Figure 10(b)). At its core, a three-level graph hierarchy is designed and interactively computed. The large graph is first compressed by node attributes like the pivot operation while allowing dynamic abstraction of different portions of the graph. This is the semantic abstraction layer. Second, the semantic abstraction is expanded through the relative regular equivalence algorithm by incorporating the neighborhood's attribute information, which forms the semantic+topology abstraction layer. Finally, the graph is further expanded to aggregate by the exact neighborhood topology, which generates the pure topology abstraction layer according to the structural equivalence concept. All three layers can be explored by the well-designed focus+context interaction which also supports multiple focuses on the abstraction.

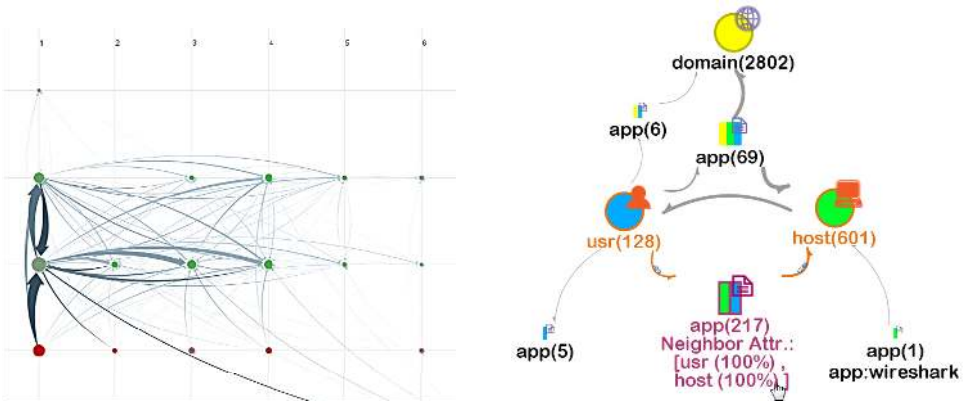


Fig. 10. Semantic and heterogeneous abstraction of large graphs. Left: PivotGraph of a communication network of people in a large company, x-axis is division, y-axis is office geography [107]. Right: OnionGraph of a host/domain-user-application network in a commodity Ethernet setting [90].

4.3 Interactive Exploration

As large graphs are often abstracted into hierarchical structures for visualization, the most relevant interactions are on the manipulation of graph hierarchies, typically the hierarchy navigation interactions. In [26], Elmqvist and Fekete classified the hierarchical abstraction based visualization into five types: above traversal, below traversal, level traversal, range traversal and unbalanced traversal (Figure 11). The hierarchy navigation methods on large graphs generally work to change the hierarchy setting within each type of the classification or switch between different types. In [5], Auber et al. proposed the method to start from

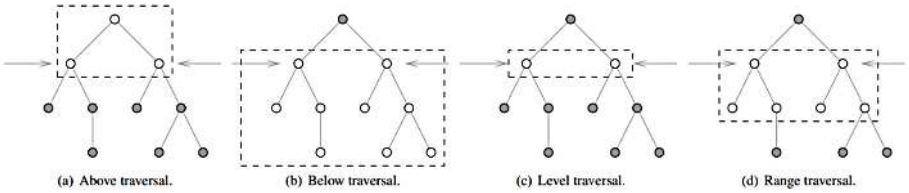


Fig. 11. The five types of hierarchical abstraction based visualization [26]. The interactive navigation methods on large graph visualization normally change the hierarchy setting within each type or switch between different types.

an above traversal and leverage an overview+detail navigation to create a below/range traversal abstraction on the focus. ASK-GraphView [1] allows the user to click on each node aggregation to expand with any traversal type and generates an unbalanced traversal. Similarly, Topological Fisheye [33] enables an interactive switching among unbalanced traversals by specifying some focuses on the graph. GrouseFlock [3] provides high-level hierarchy modification operators based on the low-level delete and merge operations.

The focus+context interaction is another classical technique for graph visualization. Though not designed for graph abstractions, this interaction can be useful for exploring large graphs. By the famous hyperbolic visualizer [70], large tree graphs can be presented with a user selected focus for details while preserving the context around the focus in its entirety. Topological Fisheye [33] achieves the similar level-of-detail rendering by a pre-computed multi-level coarsening tree. In [98], Van Ham and Perer proposed a method to start the graph analysis from a search, where the focus is essentially the search result (Figure 12). Graph context is expanded by a Degree-of-Interest diffusion from the initial focused nodes.

For more comprehensive survey on interaction over graph visualizations, readers can refer to [53,102].

4.4 View Transformation

Compared to the direct abstraction of large graph data, another class of methods work on the graph view generated from layout algorithms. These methods transform the large graph view with the objective to minimize the visual clutter invoked by the sheer graph size. Typical approaches in this class include edge filtering and bundling.

In [60], Jia et al. proposed a method to filter weak edges according to an edge centrality measure in large power-low graphs (Figure 13). Their experiments showed that the filtered graph can still maintain most topology features of the original large graph while significantly reducing visual complexity. Another approach by Van Ham et al. [99] constructed a minimal spanning tree on the large graph to reveal its backbone structure though many topology information are lost after the transformation.

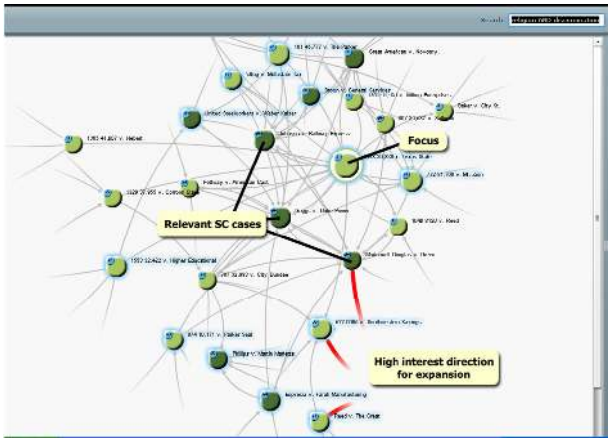


Fig. 12. Large graph exploration through Degree-of-Interest diffusion [98]. The data set is US legal document citation network.

The edge bundling approaches [32,54,68,16], on the other hand, try to aggregate similar edges into geometric clusters instead of removing them. The hierarchical edge bundling method [54] defines the edge similarity by their paths on the graph hierarchy. Specially for the tree visualization model, the edges are bent toward the polyline connecting the hierarchies of the two endpoints. In [16], Cui et al. proposed a framework to bundle edges in large general graphs (Figure 14). Their method is inspired by the road map design that splits the edge in straight line into multiple segments. The control mesh is generated first according to the layout patterns. Then all the edges are forced to pass certain control points on the mesh, leading to significant reductions in visual clutter of the graph drawing.

Though view transformations can not deal with very large graphs alone, an extra advantage is that they tend to be orthogonal to other graph abstraction methods, which promises its usage in combining multiple approaches.

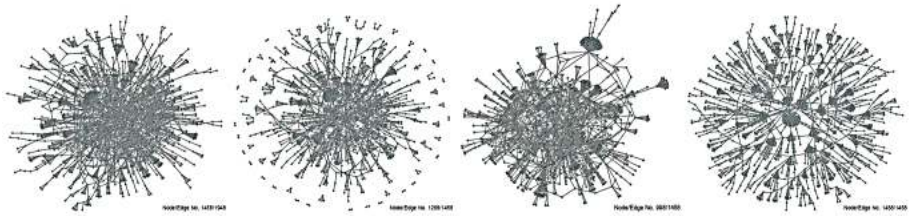


Fig. 13. Visualizations of a protein interaction graph: (a) Unsimplified; (b) Stochastic edge sampling; (c) Geodesic clustering; (d) Edge centrality based filtering [60].

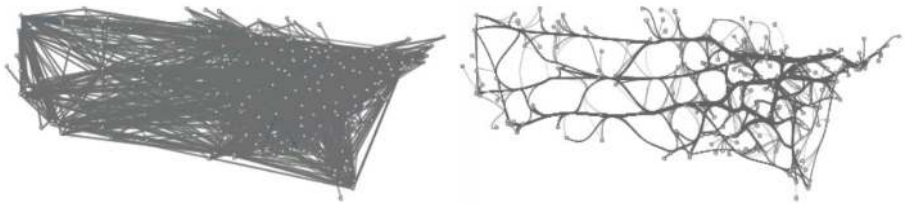


Fig. 14. Visualizations of an airline route graph. Left: Original layout. Right: Geometric edge bundling [16].

5 Software and Data Sets

There are many software and frameworks for visualizing and drawing graphs. Among these, non-commercial ones that can work with large graphs include:

- Cytoscape [17] is a Java based software platform particularly popular with biological community for visualizing molecular interaction networks and biological pathways.
- Gephi [40] 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 [39] is one of the oldest open-source graph layout and rendering engine, 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 [81] is a C++ class library for automatic layout of diagrams. Developed and supported by Germany and Australian researchers, it contains spring-electrical model based algorithm with fast multipole force approximation, as well as layered, orthogonal and planar layout algorithms.
- Tulip [95] is a C++ framework originated from University of Bordeaux I for developer of interactive information visualization applications. One of the goal of Tulip is to facilitate the reuse of components, it integrates OGDF graph layout algorithms as plugins

There are a lot of free graph drawing software available, and the list above is therefore non-exhaustive and only contains software that we are aware of, that can work on large graphs. There are other free software each with its own unique merit, but may not be designed to work on very large graphs. For example,

- Dunnart [23] is a C++ diagram editor. Its unique feature is the ability to layout diagrams with constrains, and has features such as constraint-based geometric placement tools, automatic object-avoiding poly-line connector

routing, and continuous network layout. It is developed at Monash University, Australia.
- D3.js [18] is a JavaScript library for manipulating web documents based on data. Within D3 are spring-electrical model based layout modules solved by a force directed algorithm. D3 makes it easy to take advantage of the full capabilities of modern browsers, making the resulting visualization highly portable. D3 is developed by Michael Bostock, Jeffrey Heer and Vadim Ogievetsky at Stanford University.

An important resource for testing algorithms and systems is the real world network data. There are quite a few websites that host large graph data from real world applications. For example, the University of Florida Matrix Collection [19] contains over 2600 sparse matrices (which can be considered as the adjacency matrix of graphs) contributed by practitioners from a wide range of fields, with the oldest matrices dating as far back as 1970. The Stanford Large Network Dataset [92] hosts about 50 networks from social networks, web crawls and others. The Koblenz Network Collection [63] has about 150 large networks. GraphArchive [41] contains many thousands of smaller graphs collected by the graph drawing community. House of Graphs [80] offers graphs that are considered interesting and relevant in the study of graph-theoretic problems.

6 Challenges in Large Graph Visualization

Since the 1980's, a great deal of progress has been made in visualizing large graphs. The key enabling ingredients include fast force approximations, the multilevel approach, algebraic techniques for the efficient solution of the stress and strain models, and techniques to abstract the visual complexity of large graphs. But as graphs become increasingly large, complex, and time-dependent, there are a number of challenges to be addressed.

6.1 The Ever Increasing Graph Size

The size of graphs is increasing exponentially over the years [19]. Social network is one area where we have graphs of unprecedented size. As of late 2013, Facebook, for example, has over 1.2 billion users, while Twitter has over 230 millions. Other data sources may be smaller, but just as complex. For instance, Wikipedia currently has 4.4 million interlinked articles, Amazon offers millions of items, with recommendations connecting each item to other like-items. All these pale in comparison when we consider that there are 100 billions interconnected neurons in a typical human brain, and trillions of websites on the Internet. Each of these graphs evolves over time. Furthermore, graphs likes 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. In addition, for these large networks, the average degree of nodes tends to be quite small, because there are a lot more nodes with smaller degree than those with very large ones (the power-law distribution). All these features [71] present challenges to existing algorithms.

While we can attempt to apply existing algorithms to such large and complex networks, often we find that the unique features of these networks call for rethinking of the algorithmic ingredients. For example, we have seen in previous sections that identifying vertices having very similar neighborhood is very important in the success of multilevel approach on graphs with celebrity-follower structures, as well as in abstracting and understanding these large graphs. There may well be other unique structures in large networks that should be identified and taken advantage of.

The large size of some networks also means that finding a layout for such a graph can take hours even with the fastest algorithms available. There have been work in using GPUs and multiple CPUs [4,6,28,58] to speed up the computation by a factor of 10 - 100. Given the size of graphs we are facing, further research in better hardware utilization and in algorithms that scale even better than quasilinear is warranted.

6.2 Dynamic 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 [29,11,91] that we have not touched upon due to space limit. There have been user studies to determine what are important when understanding dynamic visualizations, often in terms of what helps in preserving users' mental map [74]. Dynamic network visualization will likely continue to be area of strong interests.

How to draw complex small-world graph remains an open problem. In a good graph drawing, nodes connected by an edge should be kept near to each other. Yet in a small world graph, no node is more than a few hops away from other nodes. Therefore drawings of a small world graph using current algorithms almost always show a "hair ball", with a dense core in the middle of the display where most nodes try to be close to each other. There have been attempts mentioned in previous section to overcome this by drawing a spanning tree first, then adding on the remaining edges [100], or removing some edges with above or below average centrality [59]. Further researches are needed to visualize such graphs using these or other techniques, at the same time still visually capturing the small world nature of these graphs.

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.

6.3 Visual Representation and Abstraction

The size of graphs that can be feasibly laid-out with current algorithms already exceeds many millions of vertices and billions of edges. With so many vertices and edges, the traditional node-link diagram representation is close to a breaking point. 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 resolution higher than that for displaying the Facebook social network, our eyes cannot make use of such a display very well.

Since the purpose of visualization is to help us to understand the data and find structures and anomalies, for very large graphs, it is likely that we need algorithms to discover structures and anomalies first [2], and display these in a less cluttered form, but allowing the human operator to drill down to details when needed.

Given the limitation of screen and human vision resolutions, distilled display of the graph may also be important. Many of the abstraction methods discussed in previous sections fill a need in this space. Further research in this area may discover important graph patterns and structures that are universal in a class of graphs and thus can be abstracted into motifs. It may also reveal faster and more information-preserving ways to bundle edges.

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 representation, such as a combination of node-link diagrams and matrices [52], or maps [34], have been proposed.

Finally, large complex networks calls for fast and interactive visualization system to navigate around the massive amount of information. A number of helpful techniques for exploring graphs interactively, such as link-sliding [75], have been proposed. Further research in this area, particular 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.

7 Conclusions

In this article we review some of the most widely used algorithms, and the state of the art, for laying out large graphs. We also discuss techniques for the visual abstraction and compression of large graphs. While we are now able to visualize many large graphs efficiently and aesthetically, there remain significant number of challenges, due to the ever increasing size and complexity of graphs from real world applications. Thus the area of large graph visualization will remain one of constant innovations, both in graph theoretical research, as well as in applied algorithms and techniques.

References

1. J. Abello, F. van Ham, and N. Krishnan. ASK-GraphView: A large scale graph visualization system. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):669–676, 2006.

2. L. Akoglu, M. McGlohon, and C. Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *Proceedings of the 14th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining (PAKDD 2010)*, 2010.
3. D. Archambault, T. Munzner, and D. Auber. Grouseflocks: Steerable exploration of graph hierarchy space. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):900–913, 2008.
4. D. Auber and Y. Chiricota. Improved efficiency of spring embedders: taking advantage of GPU programming. In *7th IASTED International Conference on Visualization, Imaging and Image Processing*, pages 169–175. ACTA Press, 2007.
5. D. Auber, Y. Chiricota, F. Jourdan, and G. Melancon. Multiscale visualization of small world networks. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 75–81, 2003.
6. F. R. B. Monien and H. Salmen. A parallel simulated annealing algorithm for generating 3d layouts of undirected graphs. In *Proc. 3th Intl. Symp. Graph Drawing (GD '95)*, volume 1027 of *LNCS*, pages 90–101. Springer-Verlag, 1995.
7. S. T. Barnard and H. D. Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6:101–117, 1994.
8. J. Barnes and P. Hut. A hierarchical O(NlogN) force-calculation algorithm. *Nature*, 324:446–449, 1986.
9. G. Bartel, C. Gutwenger, K. Klein, and P. Mutzel. An experimental evaluation of multilevel layout methods. In *Proc. 18th Intl. Symp. Graph Drawing (GD '10)*, pages 80–91. Springer, 2010.
10. V. Batagelj. Visualization of large networks. In R. A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*. Springer, New York, 2009.
11. U. Brandes and M. Mader. A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In M. J. van Kreveld and B. Speckmann, editors, *Graph Drawing*, volume 7034 of *Lecture Notes in Computer Science*, pages 99–110. Springer, 2011.
12. U. Brandes and C. Pich. Eigensolver methods for progressive multidimensional scaling of large data. In *Proc. 14th Intl. Symp. Graph Drawing (GD '06)*, volume 4372 of *LNCS*, pages 42–53, 2007.
13. U. Brandes and C. Pich. An experimental study on distance based graph drawing. In *Proc. 16th Intl. Symp. Graph Drawing (GD '08)*, volume 5417 of *LNCS*, pages 218–229. Springer-Verlag, 2009.
14. I. Bruss and A. Frick. Fast interactive 3-D graph visualization. *LNCS*, 1027:99–11, 1995.
15. A. Burton, A. J. Field, and H. W. To. A cell-cell Barnes Hut algorithm for fast particle simulation. *Australian Computer Science Communications*, 20:267–278, 1998.
16. W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1277–1284, 2008.
17. Cytoscape. An open source platform for complex network analysis and visualization. `http://http://www.cytoscape.org/`.
18. D3.js. Data-driven documents. `http://d3js.org/`.
19. T. A. Davis and Y. Hu. University of Florida Sparse Matrix Collection. *ACM Transaction on Mathematical Software*, 38:1–18, 2011. `http://www.cise.ufl.edu/research/sparse/matrices/`.

20. V. de Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Advances in Neural Information Processing Systems 15*, pages 721–728. MIT Press, 2003.
21. P. Drineas, A. M. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering large graphs via the singular value decomposition. *Machine Learning*, 56:9–33, 2004.
22. C. Dunne and B. Shneiderman. Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In *Proceedings of the international conference on human factors in computing systems (CHI'13)*, pages 3247–3256, 2013.
23. T. Dwyer, K. Marriott, and M. Wybrow. Dunnart: A constraint-based network diagram authoring tool. In I. Tollis and M. Patrignani, editors, *Graph Drawing*, volume 5417 of *Lecture Notes in Computer Science*, pages 420–431. Springer Berlin Heidelberg, 2009.
24. T. Dwyer, N. H. Riche, K. Marriott, and C. Mears. Edge compression techniques for visualization of dense directed graphs. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2596–2605, 2013.
25. P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
26. N. Elmqvist and J.-D. Fekete. Hierarchical aggregation for information visualization: Overview, techniques and design guidelines. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):439–454, 2010.
27. S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010.
28. Y. Frishman and A. Tal. Multi-level graph layout on the gpu. *Journal IEEE Transactions on Visualization and Computer Graphics*, 13:1310–1319, 2007.
29. Y. Frishman and A. Tal. Online dynamic graph drawing. In *proceeding of Eurographics/IEEE VGTC Symposium on Visualization (EuroVis)*, pages 75–82, 2007.
30. T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force directed placement. *Software - Practice and Experience*, 21:1129–1164, 1991.
31. P. Gajer, M. T. Goodrich, and S. G. Kobourov. A fast multi-dimensional algorithm for drawing large graphs. *LNCS*, 1984:211 – 221, 2000.
32. E. Gansner, Y. Hu, S. North, and C. Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 187–194, 2011.
33. E. Gansner, Y. Koren, and S. North. Topological fisheye views for visualizing large graphs. In *IEEE Symposium on Information Visualization (InfoVis'04)*, 2004.
34. E. R. Gansner, Y. Hu, and S. Kobourov. Visualizing Graphs and Clusters as Maps. *IEEE Computer Graphics and Applications*, 30:54–66, 2010.
35. E. R. Gansner, Y. Hu, and S. Krishnan. Coast: A convex optimization approach to stress-based embedding. In S. Wismath and A. Wolff, editors, *Graph Drawing*, volume 8242 of *Lecture Notes in Computer Science*, pages 268–279. Springer, 2013.
36. E. R. Gansner, Y. Hu, and S. C. North. A maxent-stress model for graph layout. *IEEE Trans. Vis. Comput. Graph.*, 19(6):927–940, 2013.
37. E. R. Gansner, Y. Koren, and S. North. Topological fisheye views for visualizing large graphs. *IEEE Transactions on Visualization and Computer Graphics*, 11:457–468, 2005.

38. E. R. Gansner, Y. Koren, and S. C. North. Graph drawing by stress majorization. In *Proc. 12th Intl. Symp. Graph Drawing (GD '04)*, volume 3383 of *LNCS*, pages 239–250. Springer, 2004.
39. E. R. Gansner and S. North. An open graph visualization system and its applications to software engineering. *Software - Practice & Experience*, 30:1203–1233, 2000.
40. Gelphi. The open graph viz platform. `https://gephi.github.io/`.
41. GraphArchive. Exchange and archive system for graphs. `http://www.graph-archive.org/`.
42. L. F. Greengard. *The rapid evaluation of potential fields in particle systems*. The MIT Press, Cambridge, Massachusetts, 1988.
43. A. Gupta, G. Karypis, and V. Kumar. Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Transactions on Parallel and Distributed Systems*, 5:502–520, 1997.
44. S. Hachul and M. Jünger. Drawing large graphs with a potential field based multilevel algorithm. In *Proc. 12th Intl. Symp. Graph Drawing (GD '04)*, volume 3383 of *LNCS*, pages 285–295. Springer, 2004.
45. R. Hadany and D. Harel. A multi-scale algorithm for drawing graphs nicely. *Discrete Applied Mathematics*, 113:3–21, 2001.
46. K. M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 17:219–229, 1970.
47. J. Han and M. Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann, 2001.
48. D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. *J. Graph Algorithms and Applications*, 6:179–202, 2002.
49. D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. *Journal of graph algorithms and applications*, 6:179–202, 2002.
50. D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. *LNCS*, pages 207–219, 2002.
51. B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories, Allbuquerque, NM, 1993. Also in Proceeding of Supercomputing'95 (`http://www.supercomp.org/sc95/proceedings/509_BHEN/SC95.HTM`).
52. N. Henry, J.-D. Fekete, and M. J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE Transactions on Visualization and Computer Graphics*, 13:1302–1309, 2007.
53. I. Herman, G. Melancon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6:24–43, 2000.
54. D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12:741–748, 2006.
55. Y. Hu. A gallery of large graphs. `http://research.att.com/~yifanhu/GALLERY/GRAPHS/index.html`.
56. Y. Hu. `http://www2.research.att.com/~yifanhu/PUB/graph_draw_small.pdf`Efficient and High Quality Force-Directed Graph Drawing. *Mathematica Journal*, 10:37–71, 2005.
57. Y. Hu and J. A. Scott. A multilevel algorithm for wavefront reduction. *SIAM Journal on Scientific Computing*, 23:1352–1375, 2001.
58. S. Ingram, T. Munzner, and M. Olano. Glimmer: Multilevel mds on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, 15:249–261, 2009.

59. Y. Jia, J. Hberock, M. Garland, and J. Hart. On the visualization of social and other scale-free networks. *IEEE Transactions on Visualization and Computer*, 14:1285–1292, 2008.
60. Y. Jia, J. Hoberock, M. Garland, and J. C. Hart. On the visualization of social and other scale-free networks. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1285–1292, 2008.
61. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.
62. M. Khoury, Y. Hu, S. Krishnan, and C. E. Scheidegger. `http://www2.research.att.com/~yifanhu/PUB/lowrank_sm.pdf`Drawing Large Graphs by Low-Rank Stress Majorization. *Comput. Graph. Forum*, 31(3):975–984, 2012.
63. Koblenz. The koblenz network collection. `http://konect.uni-koblenz.de`.
64. Y. Koren, L. Carmel, and D. Harel. Ace: A fast multiscale eigenvectors computation for drawing huge graphs. In *INFOVIS '02: Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02)*, pages 137–144, Washington, DC, USA, 2002. IEEE Computer Society.
65. E. Krujaa, J. Marks, A. Blair, and R. Waters. A short note on the history of graph drawing. In *Proc. 9th Intl. Symp. Graph Drawing (GD '01)*, pages 272–286. Springer-Verlag, London, UK, 2002.
66. J. B. Kruskal. Multidimensioal scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29:1–27, 1964.
67. J. B. Kruskal and J. B. Seery. Designing network diagrams. In *Proceedings of the First General Conference on Social Graphics*, pages 22–50, Washington, D.C., July 1980. U. S. Department of the Census. Bell Laboratories Technical Report No. 49.
68. A. Lambert, R. Bourqui, and D. Auber. Winding Roads: Routing edges into bundles. *Computer Graphics Forum*, 29:853–862, 2010.
69. J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS (CHI '95)*, pages 401–408. ACM, 1995.
70. J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the international conference on Human factors in computing systems (CHI'95)*, 1995.
71. J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6:29–123, 2009.
72. F. Lorrain and H. C. White. Structural equivalence of individuals in social networks. *The Journal of Mathematical Sociology*, 1(1):49–80, 1971.
73. MDSJ. Multidimensional scaling for java. `http://www.inf.uni-konstanz.de/algo/software/mdsj/`.
74. K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *J. Vis. Lang. Comput.*, 6(2):183–210, 1995.
75. T. Moscovich, F. Chevalier, N. Henry, E. Pietriga, and J. Fekete. Topology-aware navigation in large networks. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 2319–2328, New York, NY, USA, 2009. ACM.
76. T. Munzner. Exploring large graphs in 3d hyperbolic space. *IEEE Comput. Graph. Appl.*, 18:18–23, 1998.

77. T. Munzner and P. Burchard. Visualizing the structure of the world wide web in 3d hyperbolic space. In *VRML '95: Proceedings of the first symposium on Virtual reality modeling language*, pages 33–38, New York, NY, USA, 1995. ACM.
78. M. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences (PNAS)*, 103(23):8577–8582, 2006.
79. A. Noack. An energy model for visual graph clustering. In *Proceedings of the 11th International Symposium on Graph Drawing (GD 2003)*, volume 2912 of *LNCS*, pages 425–436. Springer, 2004.
80. H. of Graphs. A database of interesting graphs. `https://hog.grinvin.org/`.
81. OGDF. Open graph drawing framework. `http://www.ogdf.net/`.
82. C. Papadopoulos and C. Voglis. Drawing graphs using modular decomposition. *Journal of Graph Algorithms and Applications*, 11(2):481–511, 2007.
83. S. Pfalzner and P. Gibbon. *Many-Body Tree Methods in Physics*. Cambridge University Press, Cambridge, 1996.
84. A. Quigley. *Large scale relational information visualization, clustering, and abstraction*. PhD thesis, Department of Computer Science and Software Engineering, University of Newcastle, Australia, 2001.
85. A. Quigley and P. Eades. FADE: graph drawing, clustering, and visual abstraction. *LNCS*, 1984:183–196, 2000.
86. I. Safro, D. Ron, and A. Brandt. Multilevel algorithms for linear ordering problems. *J. Exp. Algorithmics*, 13:1.4–1.20, 2009.
87. Z. Shen, K.-L. Ma, and T. Eliassi-Rad. Visual analysis of large heterogeneous social networks by semantic and structural abstraction. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1427–1439, 2006.
88. L. Shi, N. Cao, S. Liu, W. Qian, L. Tan, G. Wang, J. Sun, and C.-Y. Lin. HiMap: Adaptive visualization of large-scale online social networks. In *Proceedings of the IEEE Pacific Visualization Symposium*, pages 41–48, 2009.
89. L. Shi, Q. Liao, X. Sun, Y. Chen, and C. Lin. Scalable network traffic visualization using compressed graphs. In *IEEE International Conference on Big Data*, pages 606–612, 2013.
90. L. Shi, Q. Liao, H. Tong, Y. Hu, Y. Zhao, and C. Lin. Hierarchical focus+context heterogeneous network visualization. In *Proceedings of the IEEE Pacific Visualization Symposium*, 2014.
91. L. Shi, C. Wang, Z. Wen, H. Qu, C. Lin, and Q. Liao. 1.5d egocentric dynamic network. *IEEE Transactions on Visualization and Computer Graphics*, to appear.
92. snap. Stanford large network dataset collection. `http://snap.stanford.edu/data`.
93. R. Tamassia. *Handbook of Graph Drawing and Visualization*. Chapman & Hall/CRC, 2013.
94. W. S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17:401–419, 1952.
95. Tulip. Data visualization software. `http://http://www.cytoscape.org/`.
96. D. Tunkelang. *A Numerical Optimization Approach to General Graph Drawing*. PhD thesis, Carnegie Mellon University, 1999.
97. W. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 13:743–768, 1963.
98. F. van Ham and A. Perer. "Search, Show Context, Expand on Demand": Supporting large graph exploration with degree-of-interest. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):953–960, 2009.
99. F. van Ham and M. Wattenberg. Centrality based visualization of small world graphs. *Computer Graphics Forum*, 27(3):975–982, 2008.

100. F. van Ham and M. Wattenberg. Centrality based visualization of small world graphs. *Computer Graphics Forum*, 27:975–982, 2008.
101. F. van Ham, M. Wattenberg, and F. B. Viegas. Mapping text with phrase nets. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1169–1176, 2009.
102. T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J.-D. Fekete, and D. W. Fellner. Visual analysis of large graphs. *EuroGraphics - State of the Art Report*, pages 37–60, 2010.
103. C. Walshaw. A multilevel approach to the travelling salesman problem. *Oper. Res.*, 50:862–877, 2002.
104. C. Walshaw. A multilevel algorithm for force-directed graph drawing. *J. Graph Algorithms and Applications*, 7:253–285, 2003.
105. C. Walshaw. Multilevel refinement for combinatorial optimisation problems. *Annals of Operations Research*, pages 325–372, 2004.
106. C. Walshaw, M. Cross, and M. G. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, 47:102–108, 1997.
107. M. Wattenberg. Visual exploration of multivariate graphs. In *Proceedings of the international conference on human factors in computing systems (CHI'06)*, pages 811–819, 2006.
108. D. Watts and S. Strogate. Collective dynamics of "small-world" networks. *Nature*, 393:440–442, 1998.
109. G. Young and A. S. Householder. Discussion of a set of points in terms of their mutual distances. *Psychometrica*, 3:19–22, 1938.