# Chapter 4

# Big Data Visualization for Large Complex Networks

**Lei Shi, Yifan Hu, and Qi Liao**


4.1 Introduction ..... 79
4.2 Algorithms for Large Network Visualization ..... 82
4.2.1 Layout algorithms for large graphs ..... 83
4.2.1.1 Spring-electrical model–based algorithms . 83
4.2.1.2 Stress and strain model–based algorithms . 86
4.2.1.3 Other graph layout algorithms for large graphs ..... 91
4.2.2 Software for large graph layout ..... 91
4.3 Visual Abstraction of Large Multivariate Networks ..... 92
4.3.1 Hierarchical clustering–based large network visualization ..... 93
4.3.2 CompressedGraph: Large network visualization by graph simplifications ..... 95
4.3.3 OnionGraph: Multivariate network visualization by topology+attribute abstractions ..... 101
4.4 Applications of Large Complex Network Visualization in Security ..... 105
4.4.1 Compressed graph for identification of attacks ..... 105
4.4.1.1 Situation awareness ..... 105
4.4.1.2 Port scan and OS security holes ..... 105
4.4.1.3 DoS attacks ..... 107
4.4.1.4 Botnet detection ..... 107
4.4.2 ENAVis for enterprise network security management ... 108
4.4.2.1 User and application-level policy compliance ..... 108
4.4.2.2 Machine compromise and attack analysis . 110
4.4.3 OnionGraph for heterogeneous networks ..... 110
4.4.4 OntoVis for terrorism networks ..... 113
4.5 Summary and Future Directions ..... 114
References ..... 115


## 4.1 Introduction

Visualization methods are essential to the analysis of Big Data. In this chapter, we focus on large complex networks (interchangeably called the big network) that may have millions

of nodes, multiple node/edge attributes and nontrivial topologies beyond the random graph. We present both current challenges and state-of-the-art methods to visually analyze these networks. This should help practitioners and researchers in Big Data-related areas handle network data.

We start by introducing the concept of networks in the visualization research. In general, the network discussed here is composed of a set of nodes representing various types of entities or items, and a set of edges representing the complex relationship between these nodes. For example, social networks are typical instances of such a concept. The nodes can be registered users of an online application, or students in an offline teaching class. Both kinds of users can have rich demographics, for example, gender, age or interests. The edges in these social networks can be friendship ties with various degrees of familiarity, short-message communications with informative sending/receiving patterns, or even mutual hostile behaviors at school, indicating the negative relationship. On these networks, the problem of visualization arises as ordinary people fail to interpret the structure and useful patterns from the network only by looking at the list of nodes and edges. Researchers seek to design automatic algorithms to draw the networks into a virtual space, namely world coordinates, which can be further displayed in computer screens or printed on paper to amplify people's cognition on networks.

The fundamental challenge in network visualization is how to assign appropriate locations for nodes and how to connect edges between these nodes, known as the node and edge layout problem. This is referred to as the graph drawing problem where the term graph is often used interchangeably with the term network. Common graph drawing frameworks identify general constraints (called aesthetics) in the desired network layout, develop fast algorithms to satisfy these constraints, and then design visual representations to draw the network nodes and edges for better interpretation. Among known drawing aesthetics, the objective to minimize the edge crossings is often considered to be the most critical. While theoretical methods to avoid any or most edge crossings in the graph layout have been developed, known as the planarity-based approach, another class of methods called the force-directed approach is more popular in drawing real-life graphs that can have larger size and complexity. The force-directed method was first invented by Eades in his spring-electrical model [1] and later improved by Fruchterman and Reingold on the force formula [2]. The main idea is to translate the graph layout problem into the energy minimization of a physical system. Similar models include the stress model introduced by Kamada and Kawai [3], on which fast solvers have been available [4]. In visualizing networks given the node and the edge layouts, the node-link representation is the most popular, which draws nodes by enclosures and edges by lines connecting them. This representation exploits people's perception-level characteristics that tend to relate visual elements connected by uniform visual properties, also known as the connectedness principle of the Gestalt laws [5]. Other visual metaphors for networks, such as the adjacency matrix [6] and statistical charts [7], have also been studied and shown to outperform the node-link representation under certain conditions. The full technical details of the classical graph drawing algorithms and frameworks are not the focus of this chapter. Readers are referred to the book in [8] for a comprehensive understanding of the field. In particular, for ease of discussion, we will limit the scope in this chapter to the node-link visualization of simple and undirected graphs, that is, there are no loops (edges that connect one node to itself), no duplicate edges (more than one edge that connects the same pair of nodes together), and no hyperedges (edges that connect to more than two nodes) on the graph.

Over the past few decades, we have witnessed the development of network visualization techniques in both theories and applications. For example, the Graphviz open source graph layout and rendering engine [9] developed at AT&T Labs has been widely applied in various domains as the standard graph drawing toolkit. The popular D3.js library in JavaScript

and the Prefuse library in Java both carry implementations of the force-directed graph layout algorithm and example routines for network visualization. Other relevant software packages developed in academia include Gephi [10], OGDF [11], and Tulip [12]. Nowadays, the visualization of small commodity networks have grown to an extent that beginners can draw an elegant graph within 100 lines of code in a short amount of time with the help of well-developed libraries. However, on the large complex networks collected in the current Big Data era, such as the massive online social networks and the semantic-rich knowledge graphs, established graph drawing methods often face new and unsolved challenges due to the unique properties of the big network. We shall briefly discuss these challenges by looking at the four Vs of Big Data.

On data volume, Facebook now hosts more than one billion registered users, and Twitter has 288 million monthly active users. Their social graphs are millions of times larger than the classical network data set measured in Zachary's Karate Club (34 nodes). It is difficult, if not impossible, to store the entire graph ($\sim 10^9$ nodes with attributes) for visualization in a commodity desktop. When the network is displayed in browser through cloud-based services, on a small portion of such graphs ($\sim 10^7$ nodes), the time to load the data can exceed the amount that can be tolerated by users ($> 1$ min). Furthermore, if the full layout is to be computed in-situ at the client side, a moderately-sized graph ($\sim 10^5$ nodes) can take an unacceptable time to process. Finally, in cases where all earlier mentioned difficulties can be settled, normal people can hardly interpret the network structure with more than $10^3$ nodes visually, known as the perception constraint. In summary, the sheer volume of big networks brings huge challenges to network visualization due to its growing space, time, and visual complexities.

On data variety, the social graphs on Facebook and Twitter have more than a hundred attributes on each account, including their user types, demographics, and various operational/security settings; the gene regulatory network can have tens of interaction types modeled on edges. Understanding the distribution of these attributes on big networks can be critical to many real-world problems, such as the network diffusion, evolution, and so on. While the straightforward method to encode node/edge attributes through visual channels can serve a limited number of such dimensions, the navigation and visual analysis of the entire high-dimensional network remains an open challenge. New methods need to be developed to visually represent the correlation between the topology and node/edge attributes of big networks.

On data velocity, all networks are live, that is, they change over time. This first duplicates the network size by the number of time slots under study, and makes the challenge on data volume even greater. On the other hand, the visualization of network dynamics brings the new requirement to stabilize the network layout. Furthermore, how to appropriately represent the time dimension on the network visualization remains an open problem, when the classical 2D network layout has been assigned to illustrate the topology information. This thread of research has attracted significant attention recently on the topic of dynamic network visualization. Due to space limits, we will not cover more details on this part. Readers can refer to the latest survey [13] for more information.

In a few reports, the fourth V of the Big Data is known as the value. On the visualization of large complex networks, the value for end users is greatly diminished by the increasing size and dimensionality of the network data. Users find it hard to understand the hairballs created on large networks and the information-intensive drawing of most multivariate network visualizations. The key to this challenge often depends on managing the complexity in the first three facets of big networks.

Existing literature on large complex network visualization can be roughly categorized into four classes of approaches, as shown in Figure 4.1: graph layout–based, graph mining–based, graph and view transformation–based, and interaction-based. In this chapter, we

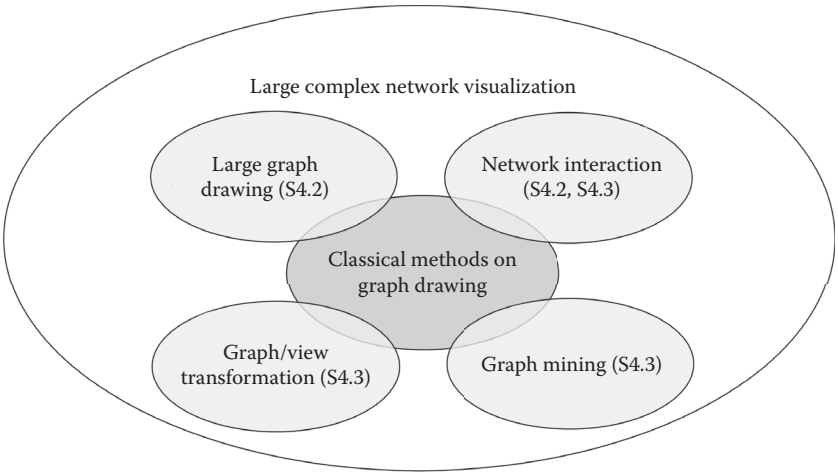



**FIGURE 4.1**: Classifications of large complex network visualization methods and the organization of this chapter.

will focus on the first three classes and their methodologies to tackle the data volume and variety challenges of large complex network visualizations. The fourth class related to network interactions is also involved in these discussions. The rest of this chapter is organized as follows:

- In Section 4.2, we talk about large graph drawing methods, which extend the classical drawing algorithms to support large and complex networks that may exhibit small world tendencies.
- In Section 4.3, we describe the visual abstraction methods for large and multivariate networks, which can employ both graph mining–based and graph transformation–based approaches.
- In Section 4.4, we give a few examples on the application of the proposed methods in real-world visualization problems.
- Finally in Section 4.5, we summarize existing approaches and potential future directions on large complex network visualization.

## 4.2 Algorithms for Large Network Visualization

In this section we look at algorithms and techniques for laying out large graphs. One important consideration when working on large graphs is that the scalability of the algorithm becomes very important. An algorithm that has, say, a computational complexity quadratic to the number of nodes may work well for graphs of a few hundred nodes. But such an algorithm would take a very long time to find a layout for a graph of a million nodes. Furthermore, an algorithm with quadratic memory complexity would not even run on a graph of a million nodes, because no single computer would be able to store $10^{12}$ real numbers in memory. In addition, many algorithms that can scale to a large size are iterative methods that attempt to find an optimal solution to a cost function. For large graphs, such a cost function tends to have many local minima. So the ability of the algorithm in escaping

from a local minimum and achieving a globally optimal drawing is also very important on large graphs.

### 4.2.1 Layout algorithms for large graphs

We first define some notations. 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.

A graph is traditionally visualized as a node-link diagram like Figure 4.2, which displays a 14-node social network. To get this diagram, we must lay out the graph, that is, assign a location to each node. 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 is 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 [1,2], and the stress model [3]. These are among the algorithms we will discuss next.

#### 4.2.1.1 Spring-electrical model–based algorithms

The spring-electrical model was first introduced by Peter Eades in 1984 [1]. He suggested embedding a graph by replacing the vertices by steel rings and 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. At the same time he made nonadjacent vertices repel each other. This intuitive idea describes the spring-electrical model, as well as the force-directed solution framework now widely used for solving this and other virtual physical models. The version of the spring-electrical model that is used today is a modification of Eades's formulation, due to Fruchterman and Reingold [2]. In this version, attractive spring force exerted on vertex $i$ from its neighbor $j$ is proportional to the squared distance between these two vertices,

$$F_a(i,j) = -\frac{\|x_i - x_j\|^2}{K} \frac{x_i - x_j}{\|x_i - x_j\|}, \; i \leftrightarrow j \tag{4.1}$$

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,

$$F_r(i,j) = \frac{K^2}{\|x_i - x_j\|} \frac{x_i - x_j}{\|x_i - x_j\|}, \; i \neq j \tag{4.2}$$

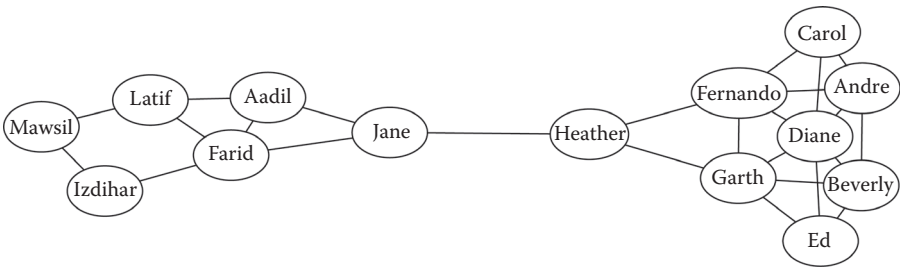



**FIGURE 4.2**: Visualization of a small social network.

The energy of this physical system [14] is

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

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 vertex along the direction of the force for a certain step length. This process is repeated for every vertex, with the step length decreasing after a complete pass over all the vertices. The process continues until the layout stabilizes.

This procedure can be enhanced by an adaptive step length updating scheme [15,16]. 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. Furthermore, the computational complexity of the spring-electrical model is $O(|V|^2)$, making it unsuitable for very large graphs. We discuss two techniques next to make the spring-electrical model scalable and better at finding a global optimal solution.

**Fast force approximation.** The force-directed algorithm involves computing the repulsion force, Equation 4.2, of all other nodes $j$ to one node $i$, and repeats this for every node $i$. Thus to compute the repulsion force once for every node requires a quadratic amount $(|V|^2)$ of force calculations.

One way to reduce this complexity is to use one of a few possible force approximation schemes. The scheme proposed by Barnes and Hut [17] 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 [18] and Quigley [19]. They both used a quadtree (or octree in 3D) data structure.

A quadtree forms a recursive grouping of vertices (Figure 4.3), 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.

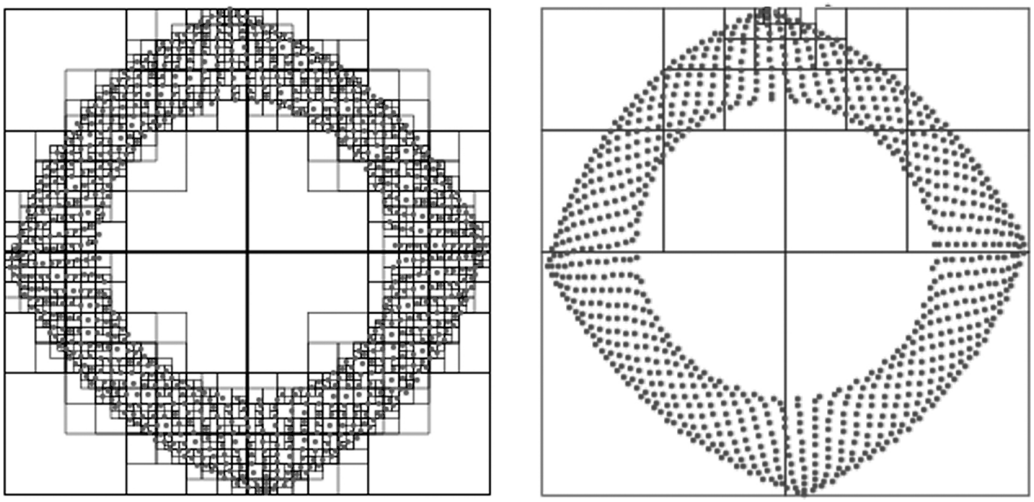


**FIGURE 4.3**: An illustration of the quadtree data structure. Left: the overall quadtree. Right: supernodes with reference to a vertex at the top-middle part of the graph, with $\theta = 1$.

Figure 4.3 (right) shows all the super-nodes (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.

Under a reasonable assumption [17,20] that the vertices do not distribute pathalogically (in practice this assumption is satisfied when the initial layout is not highly skewed in some areas), 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 using the spring-electrical model is $O(|V| \log |V|)$. This force approximation scheme can be further improved by considering force approximation at the supernode–supernode level instead of the vertex–supernode level [21].

The multipole method [22] is another force approximation algorithm with the same $O(|V| \log |V|)$ complexity. The advantage of this approach is that its complexity is independent of the distribution of vertex positions. It also uses on a quad-tree space decomposition. Hachul and Jünger applied this force approximation to graph drawing [23] in the FM$^3$ code.

**Multilevel approach.** The force-directed algorithm with the spring-electrical model repositions one vertex at a time to minimize the energy locally. On a graph, this system of springs and electrical charges can have many local minimum configurations. Therefore, starting with a random initial layout, the algorithm often cannot converge to a global optimal final layout on large graphs. A multilevel approach can overcome this limitation. In this approach, a sequence of smaller and smaller graphs is generated from the original graph, each capturing the essential connectivity information of its parent. The 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 [24–26], matrix ordering [27,28], and the traveling salesman problem [29], and were proved to be a useful meta-heuristic tool [30]. They were later used in graph drawing algorithms [31–34], sometimes under the name "multiscale." Note that a multilevel approach is not limited to the spring-electrical model, but for convenience it is discussed 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, $G^0 = G, G^1, \ldots, G^l$, is generated, each coarser graph $G^{k+1}$ encapsulating the information needed to lay out 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 [24–26]. 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. Each vertex of the resulting coarser graph has an associated weight, equal to the number of original vertices it represents. Each edge of the coarser graph also has a weight associated with it. Initially, all edge weights are set to one. During coarsening, edge weights are unchanged unless both merged vertices are adjacent to the same neighbor. In this case, the new edge is given a weight equal to the sum of the weights of the edges it replaces. Figure 4.4 illustrates MIES and the result of coarsening using edge collapsing.

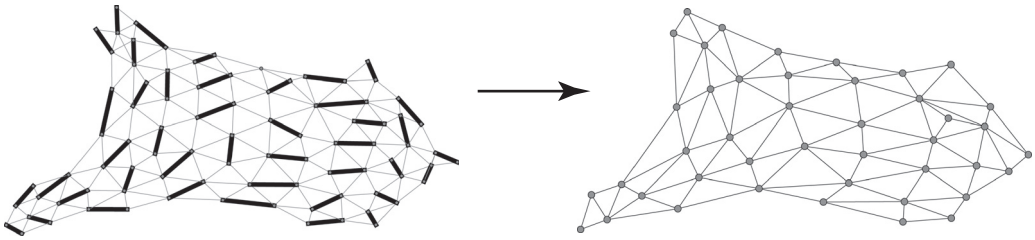


**FIGURE 4.4**: An illustration of edge collapsing–based graph coarsening. Left: original graph with 85 vertices. Edges in a maximal independent edge set are thickened. Right: a coarser graph with 43 vertices resulting from coalescing thickened edges.

The 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 because the graph on the coarsest level is very small, many algorithms, including the force-directed algorithm with the spring-electrical mode, can find a globally optimal layout.

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 layout is derived from the layout of a coarser graph, it is typically already well placed globally and what is required is just some local adjustment, therefore a conservative step-length update scheme should be used.

**Putting it all together.** When combining the fast force approximation scheme with the multilevel approach, the resulting spring-electrical algorithm can scale to graphs with millions of nodes and up to a billion edges [35]. Row "Spring electrical" of Figure 4.5 shows drawings of two graphs using such a multilevel force-directed algorithm [16]. The drawings are of good quality for both `dw256A`, a mesh-like graph, and `qh882`, a sparser graph. Two implementations of the multilevel force-directed algorithm are `sfdp` [16] and `fm`$^3$ [23]. Figure 4.6 gives some examples using sfdp.

#### 4.2.1.2 Stress and strain model–based algorithms

The spring-electrical model assumes a constant ideal edge length. But in many applications, edges of a graph often have a variable ideal length. For example, the length of an edge may represent the dis-similarity of the end nodes. It is useful to lay out such graphs so that similar nodes are closer while dissimilar nodes are further apart. While the spring-electrical model can be modified to take into account the edge length in a heuristic fashion, this kind of information is best encoded with the stress model.

**The 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

$$\sum_{i \neq j} w_{\mathrm{ij}} \left( \|x_i - x_j\| - d_{\mathrm{ij}} \right)^2 \tag{4.3}$$

where $d_{\mathrm{ij}}$ is the ideal distance between vertices $i$ and $j$. The layout that minimizes the earlier mentioned stress energy is an optimal layout of the graph according to this model. In the stress model $w_{\mathrm{ij}}$ is a weight factor. A typical choice is $w_{\mathrm{ij}} = 1/{d_{\mathrm{ij}}}^2$. With this choice, Equation 4.3 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) [37,38], and the term MDS is sometimes used to describe the embedding process based on the stress model.

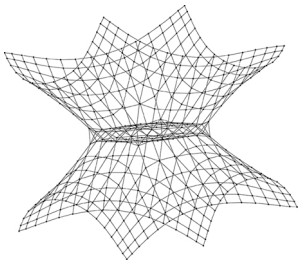
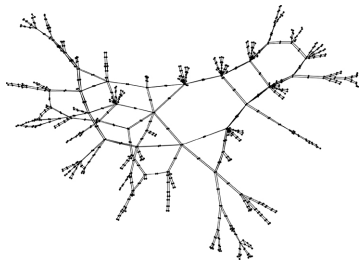
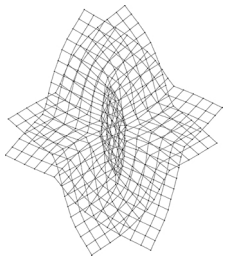
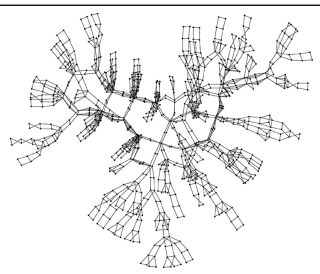
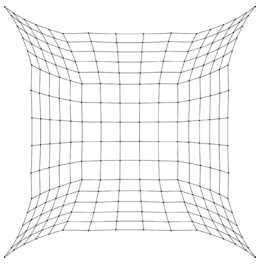
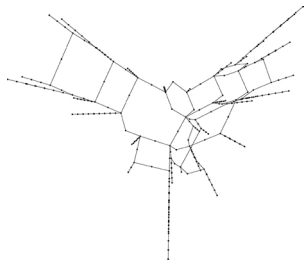
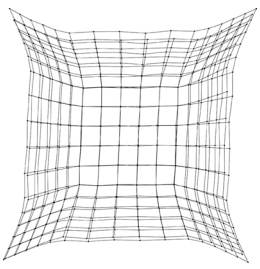
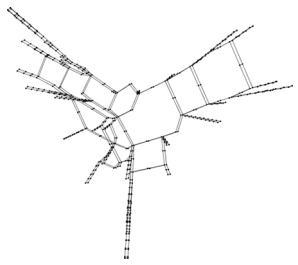
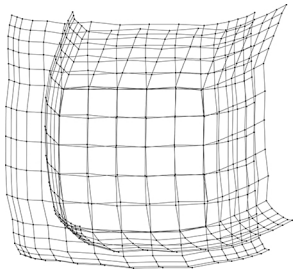
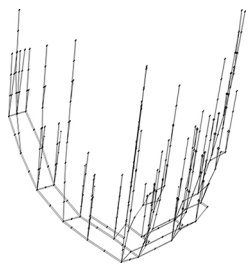
| Algorithms | dw256A | qh882 |
|---|---|---|
| Spring electrical |  |  |
| Stress |  |  |
| Classical MDS |  |  |
| Pivot MDS |  |  |
| HDE |  |  |

**FIGURE 4.5**: Result of some of the algorithms applied to two graphs, `dw256A` and `qh882`.

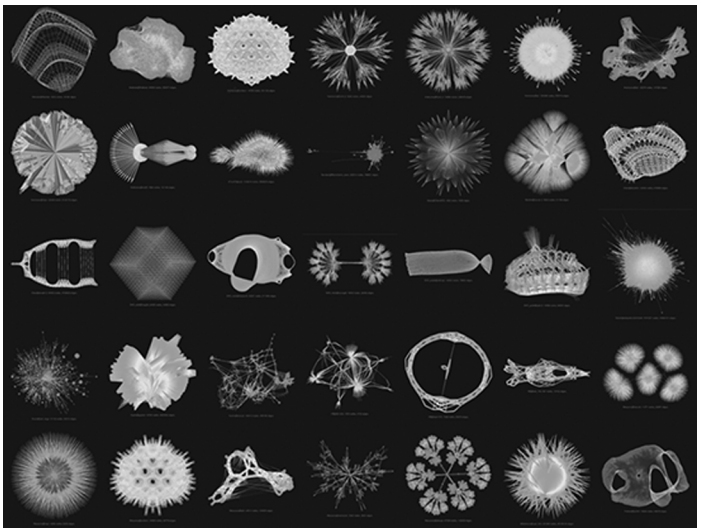


**FIGURE 4.6**: Example drawings of some graphs from the University of Florida Matrix Collection. (From T. A. Davis and Y. Hu. University of Florida Sparse Matrix Collection. *ACM Transaction on Mathematical Software*, 38:1–18, 2011 [36].)

When used to embed multidimensional data, distance among any pairs of item can be easily calculated using the appropriate metric defined in that space. However when used to embed graphs, typically only the lengths of the edges are known. In this case the distance between two non-adjacent nodes are assumed to be their shortest path distance. This practice dates back to at least 1980 in social network layout [39], and in graph drawing using classical MDS [38], even though it is often attributed to Kamada and Kawai [3] in the graph drawing literature.

There are several ways to minimize Equation 4.3. A simple (though not always robust) approach is to place each node iteratively [4],

$$x_i = \frac{1}{\sum_{j \neq i} w_{ij}} \sum_{j \neq i} w_{ij} \left( x_j + d_{ij} \frac{x_i - x_j}{\|x_i - x_j\|} \right)$$

Note that $x_i$ and $x_j + d_{ij} \frac{x_i - x_j}{\|x_i - x_j\|}$ are of distance $d_{ij}$ from each other. So the earlier mentioned simply places node $i$ at the average of the ideal positions with regard to the other nodes. Here we treat the right-hand side as known by using the current best guess of node positions. A more robust and preferred approach is to employ the majorization technique [4]. While the technique is typically derived by a careful use of Cauchy–Schwarz inequality on the stress function, it can be derived simply by rearranging the earlier mentioned equation as

$$\sum_{j \neq i} w_{ij}(x_i - x_j) = \sum_{j \neq i} w_{ij} d_{ij} \frac{x_i - x_j}{\|x_i - x_j\|}$$

The left-hand side can be written as the product of a $|V| \times |V|$ weighted Laplacian matrix and a $|V| \times d$ tall matrix of node positions. The right-hand side is seen as a weighted sum of unit vectors, and is assumed to be known by using the current best guess of node positions. The stress majorization process thus repeatedly solves the earlier mentioned linear system, and uses the resulting solution as the new node position for the right-hand side. We iterate

until the node positions no longer change. Row "Stress" of Figure 4.5 displays drawings given by the stress majorization algorithm. It performed very well on both graphs.

On large graphs, the stress model in Equation 4.3 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 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.

**The strain model (classical MDS)**. The strain model, also known as classical MDS [40], 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 [41] that the product of the positions, $x_i^T x_j$, can be expressed as the squared and double centered distance,

$$x_i^T x_j = -1/2 \left( d_{ij}^2 - \frac{1}{|V|} \sum_{k=1}^{|V|} d_{kj}^2 - \frac{1}{|V|} \sum_{k=1}^{|V|} d_{ik}^2 + \frac{1}{|V|^2} \sum_{k=1}^{|V|} \sum_{l=1}^{|V|} d_{kl}^2 \right) = b_{ij} \tag{4.4}$$

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,

$$\min_X \ ||X^T X - B||_F \tag{4.5}$$

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 Equation 4.5 becomes $X = \Lambda_d^{1/2} Q$, where $\Lambda_d$ is the diagonal matrix of $\Lambda$, with all but the $d$-largest eigenvalues on the diagonal set to zero. In other words, the 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, the layout from the strain model can be used as a good starting point for the stress model. Row "Classical MDS" of Figure 4.5 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 in 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 more time than the formation of the distance matrix. Therefore for very large graphs, the stress model is computationally expensive and prohibitive from a memory standpoint.

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 [32] improved the Kamada and Kwai [3] solution process by speeding up its convergence; Harel and Koren[33] 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 implementation, at the finest level, GRIP reverts back to the spring-electrical model of Fruchterman and Reingold [2], making it difficult to assess whether GRIP can indeed solve the stress model well.

LandmarkMDS [42] approximates the result of classical MDS by choosing $k << |V|$ vertices as landmarks, and calculating a layout of these vertices using 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 [43], 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 $CC^T$ are 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 [4,39].

The MaxEnt algorithm [44] is based on the following argument. The user only specifies the length of edges. The stress model, Equation 4.3, assumes that the unknown distance between nonneighboring 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 minimized the

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

$$\min \sum_{\{i,j\}\in E} w_{ij} \left(\|x_i - x_j\| - d_{ij}\right)^2 - \alpha \sum_{\{i,j\}\notin E} ln \|x_i - x_j\| \tag{4.6}$$

It turns out that a solution technique similar to stress-majorization can be employed, except that the matrices involved are all sparse, and the right-hand side 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.

There are at least two more recent attempts at a scalable stress model. Very briefly, the MARS algorithm [45] tried to approximate the full stress model by forming an approximate SVD of the off-diagonal matrix of the Laplacian, using a small set of selected columns of the matrix. The COAST [46] algorithms reformulate the stress model so that fast convex optimization techniques can be applied.

#### 4.2.1.3 Other graph layout algorithms for large graphs

**ACE.** Koren et al. [47] proposed an extremely fast algorithm for calculating the two extreme eigenvectors using a multilevel algorithm. The algorithm is called algebraic multigrid computation of eigenvectors (ACE). Using this algorithm, they were able to lay out graphs of millions of nodes in less than a minute. However, such eigenvector-based algorithms tend to give relatively poor graph layout.

**High-dimensional embedding.** The high-dimensional embedding (HDE) algorithm [48] 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 the $d$ 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.

HDE has many commonalities to PivotMDS. Due to its reliance on distances from the $k$-centers, HDE 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 4.5 gives drawings using HDE. It performs particularly badly on the `qh882` graph, with vertices close to each other, obscuring many details.

### 4.2.2 Software for large graph layout

While it is not difficult to write simple code to lay out small graphs, for large graphs, a lot of techniques must be used to make the code efficient and effective. Fortunately there are many free noncommercial software programs and frameworks for visualizing and drawing large graphs. The following is a non-exhaustive list that we are aware of:

- Cytoscape [49] is a Java-based software platform particularly popular with the biological community for visualizing molecular interaction networks and biological pathways.
- Gephi [10] 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 [9] 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.

- NetBioV [50] is an R package that allows the visualization of large network data in biology and medicine. The purpose of NetBioV is to enable an organized and reproducible visualization of networks by emphasizing or highlighting specific structural properties that are of biological relevance.

- OGDF [11] is a C++ class library for automatic layout of diagrams. Developed and supported by German and Australian researchers, it contains a spring-electrical model–based algorithm with fast multipole force approximation, as well as layered, orthogonal, and planar layout algorithms.

- Tulip [12] is a C++ framework from the University of Bordeaux I for development of 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.

There is other free software each with its own unique merits, but they may not be designed to work on very large graphs. For example,

- D3.js [51] 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.

- Dunnart [52] is a C++ diagram editor. Its unique feature is the ability to lay out diagrams with constraints, and it has features such as constraint-based geometric placement tools, automatic object-avoiding polyline connector routing, and continuous network layout. It is developed at Monash University, Australia.

An important resource for testing algorithms and systems is real-world network data. There are a number of websites that host large graph data from real-world applications. For example, the University of Florida Matrix Collection [36] contains over 2700 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 [53] hosts about 50 networks from social networks, web crawls, and others. The Koblenz Network Collection [54] has about 150 large networks. GraphArchive [55] contains many thousands of smaller graphs collected by the graph drawing community. House of Graphs [56] offers graphs that are considered interesting and relevant in the study of graph-theoretic problems.

## 4.3 Visual Abstraction of Large Multivariate Networks

In this section, we look at another class of large complex network visualization methods that do not display the entire graph in full detail like the graph drawing methods. Instead, the graph data has first undergone some kind of transformations into smaller and simpler

abstractions. These graph abstractions generally fit into the feasible set for classical graph drawing methods and can be displayed in readable visual forms. To study particular details on the visual abstraction, network interactions, such as various types of navigation methods, are often integrated to support the user-driven visual analysis.

The idea of assembling visual abstractions on networks is not new to the visualization community, and there are quite a few studies on this topic. Here we describe three of them based on graph clustering, graph simplification and multivariate network approaches, respectively. Interested readers are pointed to the related surveys [57–59] for more information.

### 4.3.1 Hierarchical clustering–based large network visualization

Clustering is a popular unsupervised data mining method to learn a partition of the data set. On graphs and networks, it generally works by first identifying a similarity measure over graph nodes, such as the distance-based [60] and topology-based measures [61]. Then these nodes are divided into a few groups by clustering algorithms. The visualization of these clustered graphs is promising for users in that they can reveal the high-level structure of the graph data. However, classical graph clustering algorithms (e.g., $k$-means [62]) create only one level of detail and do not allow the fine-grained navigation of these graphs. We describe the hierarchical interactive map HiMap [63] for large networks that builds over the hierarchical graph clustering algorithms and supports both flexible visual complexity control and interactive navigation over the visualization.

**Framework and algorithm.** Figure 4.7a shows the visualization pipeline for HiMap. It could be divided into two separate stages: offline data manipulation and online

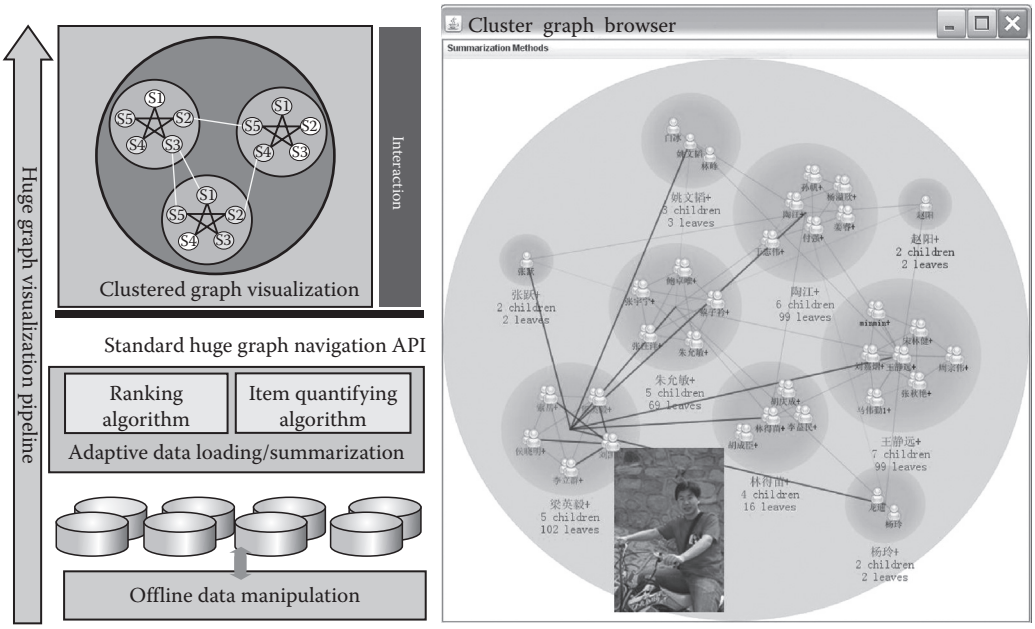



**FIGURE 4.7**: HiMap large network visualization approach: (a) the system pipeline; (b) the visual design in displaying the online social network of students in a university's department. (From Lei Shi et al., *Proceedings of the IEEE Pacific Visualization Symposium*, pages 41–48, 2009 [63].)

adaptive visualization. The offline data manipulation stage involves data collection, cleaning, and the hierarchical graph clustering. This stage prepares the network data required for the on-line visualization of large networks. The adaptive visualization stage involves data loading, graph layout, projection, and rendering. Different from the classical visualization methods that show all nodes of the large network in one view, the HiMap approach clusters the entire network into a hierarchical tree and only presents the nodes within a certain depth from the root node of the current view. This strategy greatly reduces the visual clutter commonly found in large network visualization design. Moreover, through a novel adaptive data loading method, the visual complexity of each network view is maintained within the capability of human perception. In this way, the amount of data required to load upon each view transition is also kept small by the screen's constraint, so that users do not find significant lag in the experience, but only the smooth animation designed to assist in network comprehension. Data caching is also maintained to store all the abstracted network data that users have accessed in their navigation history. When users switch back to one of the previous views, no extra time and cost is required for the repeated loading.

In particular, HiMap adopted the hierarchical graph clustering algorithm by Newman and Girvan [64], which can group all nodes in a graph into a binary tree in an agglomerative manner. To obtain the balanced hierarchical clustering structure, this algorithm is invoked recursively: (1) All the nodes are grouped into the binary tree until the termination condition of the algorithm is triggered (i.e., the delta Q value decreases below zero). The remaining groups become the clusters in the top hierarchy. (2) The subgraph of each such cluster is processed separately to obtain the subclusters in the next hierarchy. (3) This process works iteratively until the predefined maximal tree depth is reached. Finally, a static tree structure is generated over the large network.

On the clustering tree of large networks, we further apply summarization algorithms to generate the view for visualization. The objective is to maintain an interpretable visual density and readability in the final view based on the given screen real estate. The algorithm involves three steps. First, the nodes/clusters under the same father node are ranked by their customized importance. Two ranking algorithms are supported: one seeks to maximize the coverage on the subgraph given the top-ranked nodes/clusters. The second algorithm ranks the nodes/clusters according to their clustered betweenness centrality, so that the brokers connecting clusters together are displayed with higher priorities for network diagnosis purposes. The ranks with both algorithms are precomputed so that users could switch to a new configuration without any delay. Second, based on the current screen real estate and the user specification, the number of visual items to display in the entire view is computed. Third, a recursive allocation algorithm is run to determine the number and the set of displayed clusters/nodes in each hierarchy.

**Visual design.** Figure 4.7b shows the HiMap visualization of an online social network with more than 2000 users. These users are from the same department of a university. The entire view is composed of multiple filled circles enclosing each other. The largest circle in the view is drawn as the background canvas, which denotes the root node of the clustering tree, and the smaller circles indicate the first-level clusters in this tree. The fill color is selected in an ascending saturation from the clusters in the top hierarchy to the bottom hierarchy, so that the lower hierarchies of the graph are drawn in the foreground to display the fine details. Within the lowest-level hierarchy in the view, there are icons to represent the node/cluster within it. The group icon indicates the intermediate cluster in the tree, with the "+" sign following their node labels. The single icon indicates the leaf node in the tree. In the current figure, the view is set to a maximal visible depth of two.

By default, the edges between any two leaf nodes are drawn in the view by straight lines. To reduce the visual clutters common in the densely connected graph, two edge bundling methods are applied: geometric edge bundling and hierarchical edge bundling. Geometric

bundling implements the solution in [65]. Some control points are selected in the graph according to the topology, then all the edges are forced to traverse these points. The other method bundles all the edges between any two upper-hierarchy clusters together into one aggregated edge. The intra-cluster edges inside each cluster are not changed.

**Interaction.** HiMap supports a suite of interactions to inspect the details of the large network. For example, as shown in Figure 4.7b, users can mouse over a node/cluster in the view to show the corresponding profile of the representative leaf node (in this data set, the photo of the user). All the neighboring nodes and edges are highlighted to facilitate the user's analysis. More importantly, HiMap provides navigation interactions to visually traverse the large complex network based on its clustering tree. The basic interaction method is the hierarchical drill-in/roll-up. By double-clicking on one cluster in the view, the subgraph under this cluster is summarized again and shown in the entire view, with more visual items for the detail. This is demonstrated in Figure 4.8. In the first stage of the interaction, the selected cluster in the blue frame is expanded geometrically, while the other clusters in the same hierarchy fade out gradually. In the second stage, this selected cluster becomes the root node and fills the entire canvas, while new clusters start to emerge under the selected cluster. In the third stage, subclusters/nodes appear within the new clusters to fill up the view and ensure the visual complexity requirement. Similarly, the view can be rolled up to go back to the upper hierarchy by double-clicking on the current canvas. Animated transitions are used during these interaction stages to provide a smooth viewing experience.

### 4.3.2 CompressedGraph: Large network visualization by graph simplifications

While graph clustering–based visualization is a good fit for large-scale social networks, the same method can be inadequate on other domains when the network does not necessarily

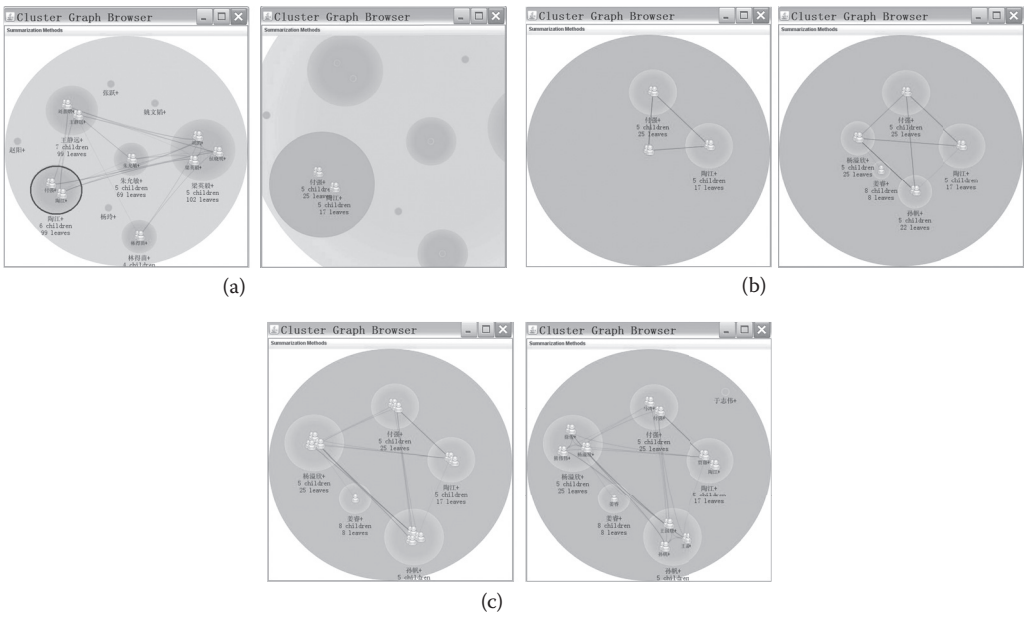



**FIGURE 4.8**: The drill-in interaction in HiMap; the circled frame in (a) indicates the selected cluster to drill-in: (a) Stage I; (b) Stage II; (c) Stage III. (From Lei Shi et al., *Proceedings of the IEEE Pacific Visualization Symposium*, pages 41–48, 2009 [63].)

have community structures. A typical case is the network traffic graph in a lab setting. Each host in the lab (the network node) can have more connections to the external network, for example, by surfing the web, than the internal connections. Figure 4.9b illustrates the visualization result using the clustering–based method over the original traffic network in Figure 4.9a. The clustered view does not reduce the visual complexity for user analysis, and even worse, the original network topology is destroyed to some extent. In this part, we describe a new graph simplification method, called the CompressedGraph [66], based on the concept of structural equivalence in social network analysis [67]. Rather than detecting communities, the structural equivalence method classifies the network nodes into categories by their positions taken in the network, depending only on the network topology. On large networks in domains such as the computer network, the visualization result can be much effective than the clustering–based method. As shown in Figure 4.9c, the 3460-node network traffic graph is reduced to 18 grouped nodes and 28 edges, while the same graph using modularity-based clustering [64] generates 50 intermediate clusters (with a maximal depth setting of four), 939 leaf nodes and 15, 438 edges in the top view.

We should note that the similar idea of visually compressing graphs by exploiting the topology redundancy has also been studied in a few other studies [68–70]. Similar visual effects can be achieved; however, they vary in the compression rate and the applicability to different domains.

**Framework and algorithm.** The main algorithm in the CompressedGraph is called structural equivalence grouping (SEG). SEG has a time complexity linear to the number of edges in graphs. Also, SEG enjoys an advantage in that it can support not only the undirected graphs, but also the directed, weighted graphs, and provide a fuzzy version of the algorithm to control the visual complexity after the compression.

Intuitively, SEG aggregates the graph nodes with the same neighbor set together into groups and constructs a new graph for visualization. For example, in the traffic graph of Figure 4.10a, the host "192.168.2.23" can be combined with the other three surrounding hosts with exactly the same connection pattern. The new graph after SEG processing (Figure 4.10b) is called the *compressed graph*. The compressed graph has two kinds of nodes: the *single-node* remaining from the original graph (drawn as hollow) and the *mega-node* grouped from multiple *subnodes* in the original graph (drawn as filled colors). In the following, the SEG algorithm is formally defined.

Let $G = (V, E)$ be a directed, weighted, and connected graph where $V = \{v_1, ..., v_n\}$ and $E = \{e_1, ..., e_m\}$ denote the node set and the link set. Let $W$ be the graph adjacency matrix where $w_{ij} > 0$ indicates a link from $v_i$ to $v_j$, with $w_{ij}$ denoting the link weight. In each row of $W$, $R_i = \{w_{i1}, ..., w_{in}\}$ denotes the row vector for node $v_i$, representing its connection pattern. The compressed graph after SEG is denoted as $G^* = (V^*, E^*)$. The compression rate is defined by $\Gamma = 1 - |V^*|/|V|$.

The basic SEG algorithm takes the graph as a simple, undirected, and unweighted one by setting $w_{ii} = 0$ and $w_{ij} = w_{ji} = 1$ for any $w_{ij} > 0$. On a graph $G$, for any collection of nodes with the same row vector (including the single outstanding node), SEG aggregates them into a new mega-node/single-node $Gv_i = \{v_{i_1}, ..., v_{i_k}\}$. All $Gv_i$ form the node set $V^*$ for the compressed graph $G^*$. Let $fv_i = v_{i_1}$ denote the first subnode in $Gv_i$. The link set $E^*$ in $G^*$ is generated by replacing all $fv_i$ with $Gv_i$ in the original link set, and removing all the links not incident to any $fv_i$. For directed graphs, the adjacency matrix $W$ is transformed to encode the two connection directions for each node. Each row vector $R_i (i = 1, ..., n)$ becomes $R_i = \{w_{i(-n)}, ..., w_{i(-1)}, w_{i1}, ..., w_{in}\}$ with $w_{i(-j)} = w_{ji}$ for $j = 1, ..., n$. For weighted graphs, the adjacency matrix $W$ is switched to the weighted one by mapping a numeric data attribute of the link $(i, j)$ (e.g., flow count in the traffic graph) to $w_{ij}$ in the matrix. To further increase the compression rate, the discretization of the link weight

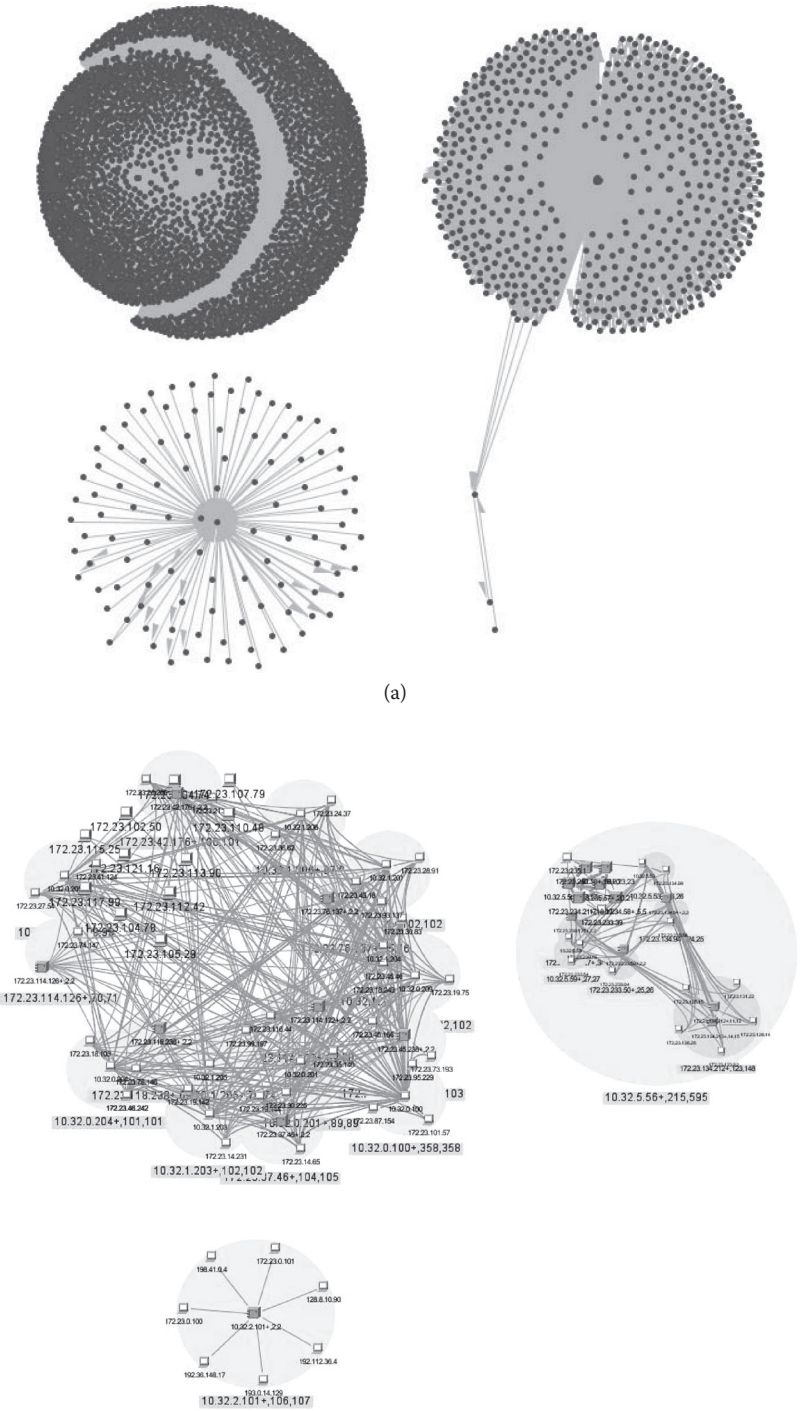
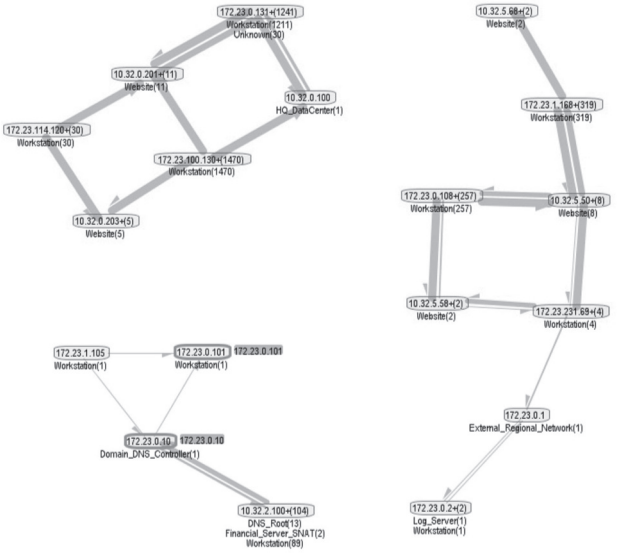
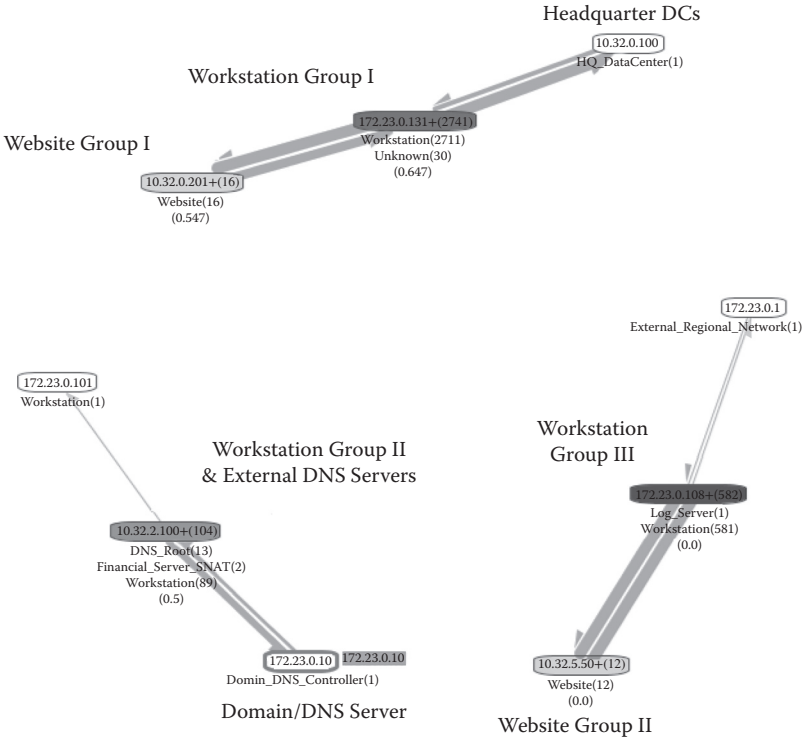



**FIGURE 4.9**: A comparison of large network visualization methods on the VAST 2012 Mini Challenge-II network traffic graph (3460 nodes, 48,599 edges): (a) Original view; (b) clustered view. (From Lei Shi et al., *IEEE International Conference on Big Data*, pages 606–612, 2013 [66].) *(Continued)*




(c)




(d)

**FIGURE 4.9**: (*Continued*) A comparison of large network visualization methods on the VAST 2012 Mini Challenge-II network traffic graph (3460 nodes, 48,599 edges): (c) SEG-compressed view; (d) manually-compressed view over SEG. (From Lei Shi et al., *IEEE International Conference on Big Data*, pages 606–612, 2013 [66].)

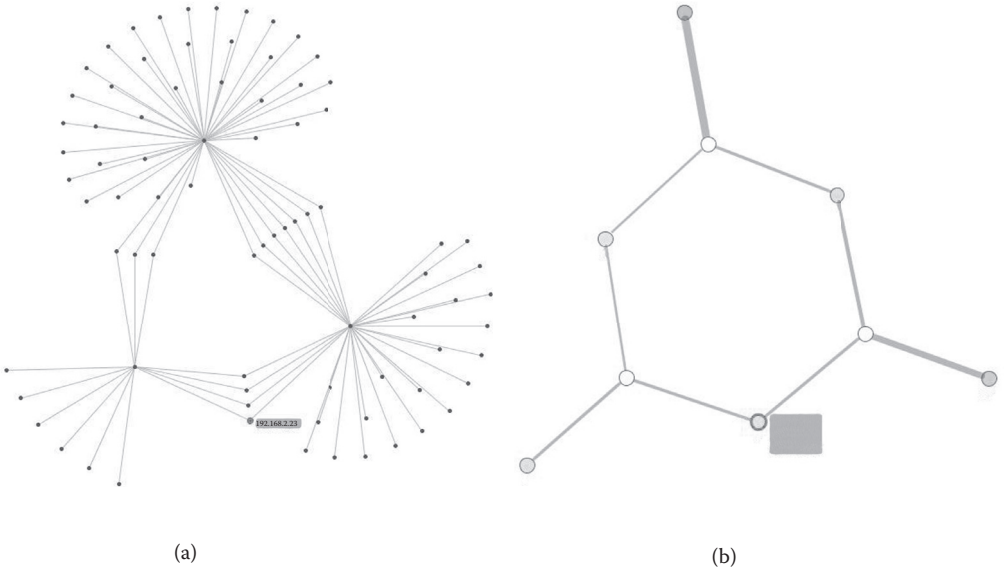



**FIGURE 4.10**: The result of the SEG algorithm on an undirected graph (a) original graph and (b) compressed graph. (From Lei Shi et al., *IEEE International Conference on Big Data*, pages 606–612, 2013 [66].)

is allowed: first, all link weights are transformed into $w_{ij} \in (0, 1]$ by either the linear or nonlinear normalization; second, a bin count $B(B \geq 1)$ is picked to regenerate link weights by $w_{ij} = \lceil w_{ij} \times B \rceil$.

SEG is a single-pass, deterministic algorithm in that for the same original graph, it always produces the same compressed graph. In real usage, the user would like to flexibly control the visual complexity after the compression. This can be achieved through the fuzzy SEG algorithm. The basic idea is to group nodes with not only the same, but also similar neighbor sets. The compression rate can be increased with a bounded compensation on the accuracy. The key is to define the pairwise similarity score between graph nodes. The standard Jaccard similarity can be adopted, that is, between two sample sets $A$ and $B$ we have $J(A, B) = \frac{|A \bigcap B|}{|A \bigcup B|}$. For the directed and weighted graphs, we introduce a unified Jaccard similarity computation between node $v_i$ and $v_j$ in the graph $G$ by $\rho = \frac{\sum_{\forall k} \min(w_{ik}, w_{jk})}{\sum_{\forall k} \max(w_{ik}, w_{jk})}$. Note that for the directed graph, $k = -n, ..., -1, 1, ..., n$. The fuzzy SEG is achieved by setting a similarity threshold $\xi$. The pair of nodes with $\rho \geq \xi$ is grouped together iteratively.

**Visual design.** The right panel of Figure 4.11 gives an example of the SEG-compressed graph visualization. As shown in the figure, the single-nodes have no fill color and the mega-nodes have solid fill colors, with the color saturation mapped to the number of subnodes in the group. The larger group is filled with the more saturated color. By default, the fill color hue is blue, for example, the node "192.168.1.10+" in the top-right of Figure 4.11. For the mega-nodes created by the fuzzy SEG, for example, the node "192.168.2.11+" on the right part of of Figure 4.11, the fill color is blue-green, indicating a pairwise similarity score below one. The fill color will gradually change from blue to green and finally to brown, as the similarity score drops from one to zero.

The labels on the mega-node are created by aggregating the labels of the subnodes in the original graph. Due to the space limitations of the graph view, an abstracted label is displayed on each mega-node as the node identifier. The full label will pop up upon a mouse hovering or a click action, for example, the one on the node "192.168.1.10+." The

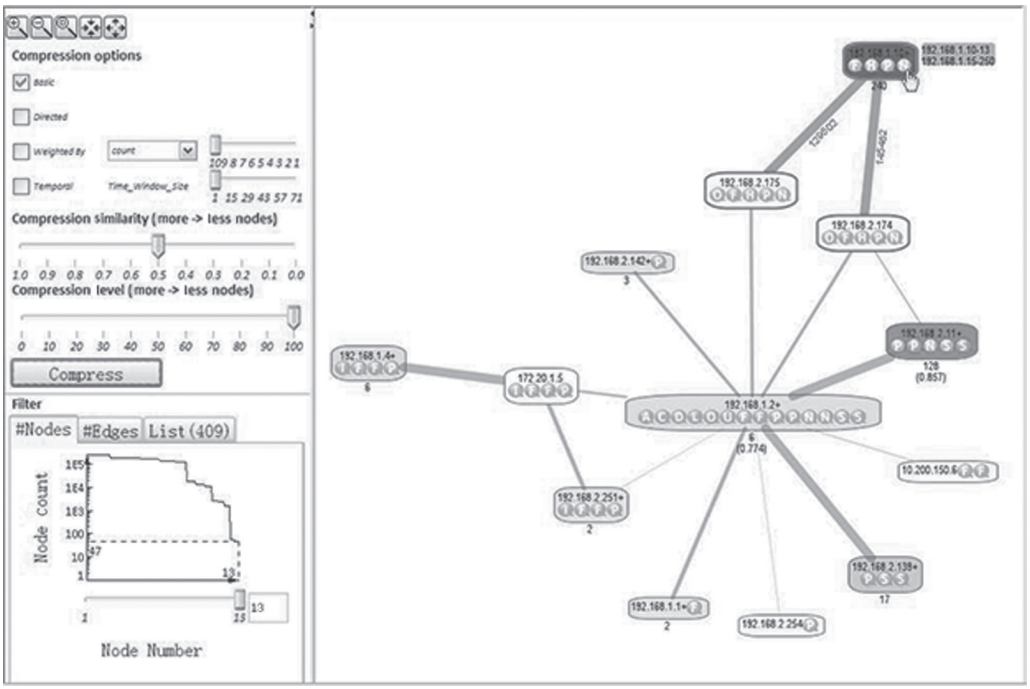



**FIGURE 4.11**: The compressed graph visualization interface. The main graph view is on the right, and the algorithm controller is on the left. The data set is from the VAST Challenge in 2011. (From Lei Shi et al., *IEEE International Conference on Big Data*, pages 606–612, 2013 [66].)

group size of each mega-node is drawn below the node, together with the within-group similarity score when it is below one. By default, straight lines are used to represent the links in the compressed graph, with the line thickness mapped to the sum of counts of all the corresponding links in the original graph. The node/edge attributes on compressed graphs can also be mapped onto the visuals. For example, in the case of the traffic graph in the security domain, the node label can be the host types (Figure 4.9c) or the alphabetical anomaly icons (Figure 4.11) indicating the detected type of anomalies on the host.

**Interaction.** The CompressedGraph visualization supports most basic graph interactions. In addition, it provides user control over the SEG algorithm settings. In the control panel of Figure 4.11, the "Compression level" section lists multiple checkboxes as switches for the undirected, directed, and weighted SEG. For the weighted SEG, the link weight mapped from the edge attribute of the graph can be specified. To use the normalized link weight, a bin number can be specified to discretize the weight. In the "Compression level" and "Compression similarity" sections, the compression rate can be tuned to generate smaller or larger graph abstractions through the fuzzy SEG algorithm.

Complementary to the automatic SEG operation, CompressedGraph introduces manual node grouping/splitting interaction. In many cases, the users have their own criteria towards a best graph abstraction. The user can either select a collection of nodes and click the "group" button in the navigation panel, or use drag-and-drop to group one node into another one at a time. In the drag-and-drop process, the pairwise similarities of the selected node with all the other subnodes in the candidate group are shown as the visual hint. On the other hand, the user can either select the mega-nodes and click the "split" button, or simply double-click on one mega-node. The mega-node grouped by the fuzzy SEG will first collapse

to mega-nodes by the deterministic SEG, and upon another double-click, collapses to the original subnodes. Figure 4.10a gives an example of the manually created abstraction over the SEG algorithm, which further reduces the network size to 9 grouped nodes and 11 edges.

The proposed SEG-based compressed graph visualization method has been integrated into a tool called network security and anomaly visualization (NSAV) to address the problem in the network security domain, which will be detailed in Section 4.4.

### 4.3.3 OnionGraph: Multivariate network visualization by topology+attribute abstractions

The aforementioned visual abstraction methods can be effective in displaying large-scale networks. However, the modern networks are often multivariate in as there are multiple attributes on the network nodes and edges. Some networks are even heterogeneous as their nodes and edges are of different types. For example, in a bibliographic information network, a node can be an author with an affiliation, a paper with a certain topic, or a venue (i.e., conference/journal) held in some location. An edge can represent the relationship of citations, authorizations, presentations, and so on. On these multivariate networks, except for visualizing the network topology, how to display the network node/edge attributes and the correlations between these attributes and the network topology emerges as a new problem. This is particularly challenging because in the graph drawing approaches, the node locations have traditionally been used to represent the network topology. While a few visual channels, for example, the node shape, fill color, and edge thickness are available to display the node/edge attributes, the constraints on human perception and the nature of high-dimensional information on networks make direct encoding methods infeasible for many scenarios. In this part, we describe OnionGraph [71], an integrated framework for the exploratory visual analysis of large multivariate networks. OnionGraph creates several hierarchies on the multivariate network by combining the topology and attribute information on these networks. Through visual abstractions based on the network hierarchies and interactions similar to the compressed graph, OnionGraph supports level-of-detail viewing on multivariate networks.

**Framework and algorithm.** OnionGraph introduced a stratified semantic+topological principle. At high levels, the large network is aggregated by a combination of semantic information (the node type and attributes). Interesting portions of the abstraction can be drilled down in situ by exploiting topological features. After steps of user navigation, a stratified network view with different levels of abstraction is created on demand to serve the complicated heterogeneous network analysis tasks. Sketched examples are given in Figure 4.12, which shows the five-level hierarchical structure of OnionGraph. The design features the semantic+topological principle: the semantic aggregations (SA) in level-I and level-II are purely semantic, the level-III RRE combines semantic and topological information, and the level-IV (SSE) is mostly topological. In the following, we formally present the suite of OnionGraph algorithms to create the hierarchical abstraction over multivariate networks.

*Heterogenous network.* Let $G = (V, E)$ be a directed and weighted multivariate network. $V = \{v_1, ..., v_n\}$ and $E = \{e_1, ..., e_m\}$ denote the node and link sets. $W$ denotes the adjacency matrix where $w_{ij} > 0$ indicates a link from $v_i$ to $v_j$, with $w_{ij}$ denoting the link weight. On each node $v_i$, $N^+(v_i) = \{j|w_{ij} > 0\}$ and $N^-(v_i) = \{j|w_{ji} > 0\}$ indicate the outbound and inbound neighborhood set, both representing its connection pattern. Let $D = \{d_1, ..., d_s\}$ be the type and attribute of network nodes in $G$, with $s$ dimensions in total. $D(v_i) = \{d_1(v_i), ..., d_s(v_i)\}$ denotes the type/attribute values of the node $v_i$, with $d_k(v_i)$ indicating the value in the $k$th dimension.

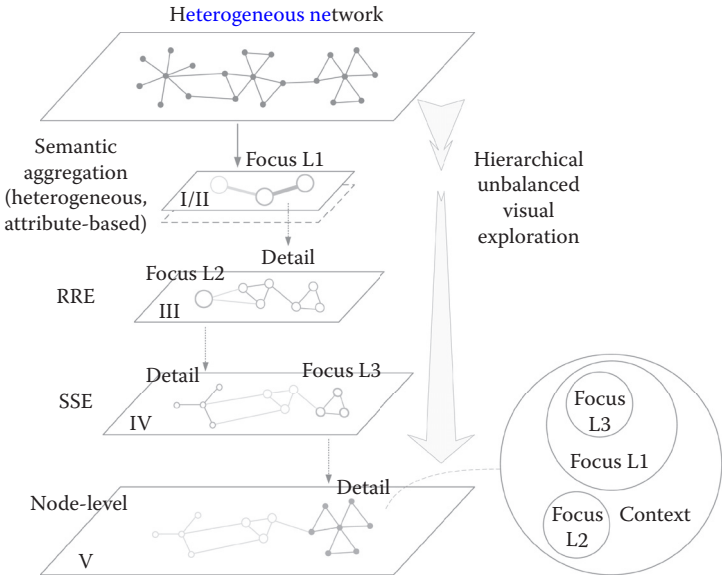



**FIGURE 4.12**: OnionGraph structure featuring five hierarchies below the original network: networks by semantic aggregations (SA) on the node type (heterogeneous abstraction) and the node attributes, relative regular equivalence (RRE), strong structural equivalence (SSE), and the single-node-level network in the finest granularity. In each hierarchy, the network can be expanded at certain points into their lower-hierarchy details. (From Lei Shi et al. *IEEE Pacific Visualization Symposium*, pages 89–96, 2014 [71].)

*Network partition.* Let $P : V \rightarrow \{1, 2, ..., t\}$ be a partition (role assignment, coloration, grouping, interchangeably) of the network $G$ into $t$ subgroups of nodes. $P(v_i)$ indicates the partition index of node $v_i$.

The OnionGraph algorithms to create a network abstraction is equivalent to finding a partition of the network according to the abstraction settings.

*SA* creates a partition of the network using a selected set of node types or attributes. Formally, for any nodes $v_i$ and $v_j$ in a network $G$, given the selected attribute set $\overline{D} \subseteq D$, the SA network partition $P$ satisfies:

$$\overline{D}(v_i) = \overline{D}(v_j) \Leftrightarrow P(v_i) = P(v_j) \tag{4.7}$$

Figure 4.13a illustrates a partition based on the node attribute having values "I" or "II."

*RRE* [72] is defined recursively on the network node by the same set of neighborhood roles. For any nodes $v_i$ and $v_j$ in a network $G$, a regular equivalence network partition $P$ satisfies:

$$P(v_i) = P(v_j) \Rightarrow P(N^+(v_i)) = P(N^+(v_j)) \;\; and \;\; P(N^-(v_i)) = P(N^-(v_j)) \tag{4.8}$$

However, directly applying the regular equivalence on a network will lead to many possible partitions. Figure 4.13b gives a particular case. In the extreme case, the identity partition (every node serves a different role) and the complete partition (every node serves the same role) are both regular. It is hard to find an appropriate regular equivalence partition in the real usage. OnionGraph introduces a practical solution to apply RRE, which can work on top of the existing SA partition. RRE explicitly derives the maximal regular equivalence

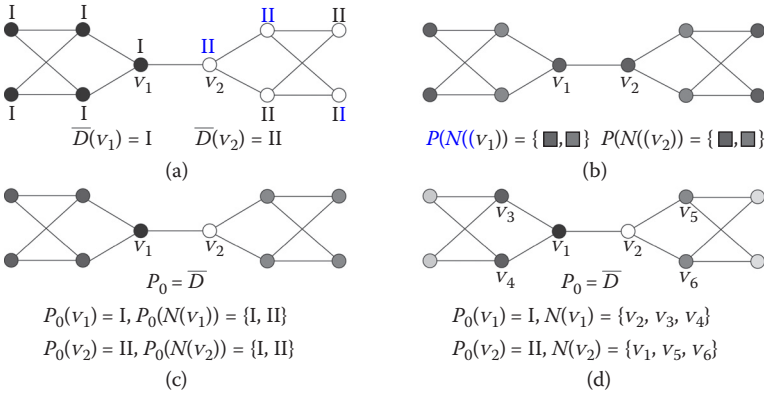



**FIGURE 4.13**: OnionGraph network partitions (undirected case). In each subfigure, the node fill color indicates the partition index: (a) semantic aggregation (the selected attribute value is labeled on the node); (b) a regular equivalence partition; (c) the regular equivalence relative to the semantic aggregation in (a); (d) SSE. (From Lei Shi et al. *IEEE Pacific Visualization Symposium*, pages 89–96, 2014 [71].)

partition by refining the semantic partition. Mathematically, the RRE partition $P$ over a semantic partition $P_0$ satisfies:

$$\begin{aligned} &P(v_i) = P(v_j) \Leftrightarrow P_0(v_i) = P_0(v_j) \quad and \\ &P_0(N^+(v_i)) = P_0(N^+(v_j)) \quad and \quad P_0(N^-(v_i)) = P_0(N^-(v_j)) \end{aligned} \tag{4.9}$$

Figure 4.13c gives an example of the RRE partition relative to the semantic partition in Figure 4.13a.

*SSE* partition [67] requires the network nodes to have exactly the same neighborhood set. For any nodes $v_i$ and $v_j$ in a network $G$, the SSE network partition $P$ over a RRE partition $P_0$ satisfies:

$$\begin{aligned} &P(v_i) = P(v_j) \Leftrightarrow P_0(v_i) = P_0(v_j) \quad and \\ &N^+(v_i) = N^+(v_j) \quad and \quad N^-(v_i) = N^-(v_j) \end{aligned} \tag{4.10}$$

Besides the standard definition, there are a few variations of the RRE/SSE partitions. The undirected RRE/SSE (Figure 4.13) considers the union of both inbound and outbound neighborhood sets, and the weighted RRE/SSE considers the number of role occurrences in the neighborhood (RRE) and the weight of the connecting edges (SSE). These options are included in the OnionGraph design and can be configured by users.

**Visual design.** A typical OnionGraph visualization is shown in Figure 4.14. Each colored node represents a group of the original nodes in the large network. The node size encodes the number of individual nodes in each group. The initial abstraction aggregates all the nodes by their types ("author," "paper," and "venue" in this figure), as indicated by the icon on the top-right part of each node. The node group in the top-level heterogeneous abstraction is displayed by a filled node, for example, the spring-green node in the center of Figure 4.14, representing all 9557 papers. All the other nodes in this figure are expanded from the top level. They are drawn by the "onion" metaphor composed of several concentric circles. The number of circles indicates the abstraction hierarchy: the SA has three circles (venue nodes in Figure 4.14), RRE has two circles (author nodes in Figure 4.14), and SSE has one circle. The individual node only remains a solid dot. Upon the top-down exploratory analysis, the visual complexity of each node group is reduced as the number of node groups increases, so as to balance the overall complexity of the OnionGraph view.

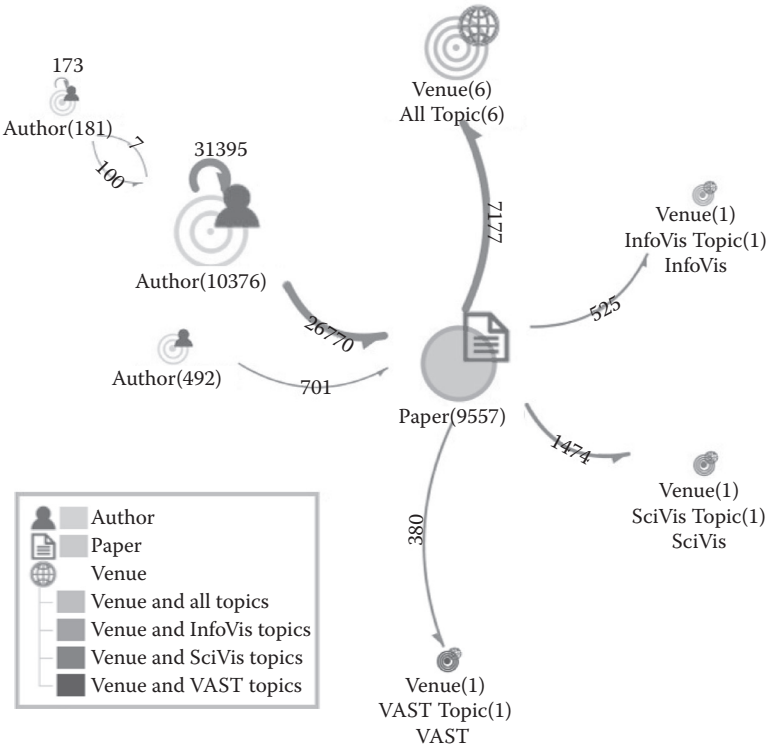



**FIGURE 4.14**: An OnionGraph network visualization of the author-paper-venue bibliographic network in the visualization community. Three author groups indicate the different connection patterns: normal authors with co-authors and publications, special authors who only write single-authored papers, and anomalous authors without a publication (potential errors in the data set). Four venue groups indicate the conferences/journals on different topics. (From Lei Shi et al. *IEEE Pacific Visualization Symposium*, pages 89–96, 2014 [71].)

Node color in OnionGraph is determined by the type/attribute values of each node group. In Figure 4.14, initially three colors (yellow, spring-green, indigo) are picked uniformly on the color hue. After the expansion of the venue node into subnodes, four new colors are assigned with the linear hue and saturation offsets from the indigo color, as shown by the legend in the bottom-left part of Figure 4.14. The node labels by default show the value of the currently selected node type/attributes. When a node group contains only one node, the detailed node title is also shown in the label. The selected nodes are drawn with dark-red outlines and labels, coupled with a "+/-" sign upon hovering to indicate the lower/upper level to explore. The neighborhood of the selected nodes and their connecting links are drawn in dark orange. The link thickness and the link label encode the number of individual links between the groups. Different from the simple graphs, OnionGraph usually has a loopback link on each node group, which indicates the internal connection. This is displayed by the arc above the node.

**Interaction.** The typical OnionGraph interaction works like this: the user selects a portion of interesting subgraphs in the network, specifies an abstraction profile, and finally executes the OnionGraph algorithm to display the finer/coarser-grained visual representation on these selected subgraphs. The network selection is supported in multiple ways, through single-clicks on network nodes, rubber band selection/deselection, and Ctrl plus single-click to select a node set with the same abstraction profile. The abstraction profile is

configured in the control panel on the left side of the interface. The control panel includes the abstraction level control, the attribute multiselection, and the switches indicating the abstraction settings, for example, directed, weighted, and fuzzy RRE/SSE algorithms. The selected subgraph is processed by clicking the abstract button. In another usage, users can double-click on the selected node set in the main network view to expand to the next level abstraction. When the context is set to the "collapse" mode, the selected node set is regrouped into the upper-level abstraction.

From the menu of the OnionGraph interface on the top, a few configurations such as the layout algorithm, and the network node/link visual encoding, can be specified. OnionGraph also allows the neighborhood charting mode. Each node group aggregated by RRE is drawn by a chart instead of the onion metaphor, showing the distribution of attribute values in the node's neighborhood. The bottom-left filter panel allows the user to plug in node attribute filters on the network. The OnionGraph abstraction is executed over the filtered network. On the network link, a simplified filtering mechanism is supported by a multi-checkbox interface. The rightmost part of the OnionGraph interface shows the network details upon visualizations and interactions. The top-right panel displays the node legend indicating the icon/color coding of each node group in the network. The center-right panel displays the list of nodes currently selected in the main view. Upon choosing one node in the list, the node attributes are displayed in a key-value table in the bottom-right panel.

## 4.4 Applications of Large Complex Network Visualization in Security

### 4.4.1 Compressed graph for identification of attacks

Large-scale network graph analysis and visualization has been applied and shown to be effective in computer security and network management. One example is NSAV [66]. It is challenging to visualize network traffic graph due to the sheer volume and large number of hosts and network connections among them. NSAV looks to reduce the number of nodes (and their connections), thus reducing the visual complexity for network administrators and operators to analyze their networks.

#### 4.4.1.1 Situation awareness

Consider John, the network operation lead of a large enterprise, who collects the Netflow firewall log on a daily basis; he is checking the corporate network status of the recent three days for noteworthy events. He starts by loading all of the network traffic in this period, as shown in Figure 4.15a. Because the view is too messy, he continues by creating a compressed traffic graph, as shown in Figure 4.15b. The comparison of the two graph visualizations is quite significant as the second graph is much smaller and easier to visualize.

#### 4.4.1.2 Port scan and OS security holes

Based on his network structure, John bypasses three server machines (1.2, 1.6, 1.14), which routinely communicate with all the hosts for DNS/data services. Also according to his knowledge, the suspicious behaviors of a hub node, for example, port scan, is often associated with OS security holes. Therefore, he clicks on this anomaly type and highlights all the hosts with such an anomaly on the graph. To drill-down to individual hosts, he splits the fuzzy group and locates 2.171–2.175 as the threats. He finds that 2.174/175 are more

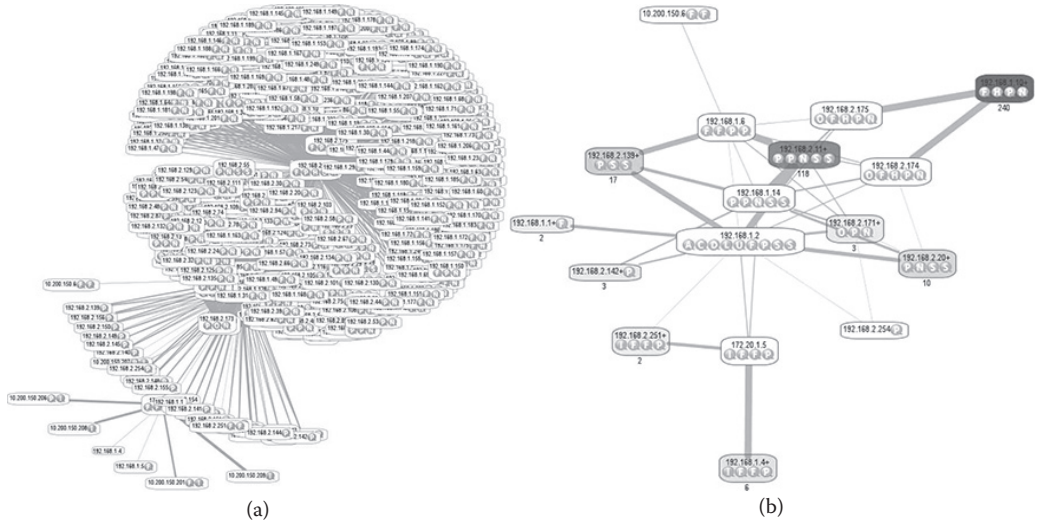



**FIGURE 4.15**: Corporate network traffic graphs using the NSAV compression tool: (a) Original and (b) compressed. (From Lei Shi et al., *IEEE International Conference on Big Data*, pages 606–612, 2013 [66].)

dangerous due to the higher port-scan rate (by the link thickness) and the cross-subnet floods to 1.10-250 where many hosts do not exist. The screenshot with anomaly views of 2.174/175 is given in Figure 4.16.

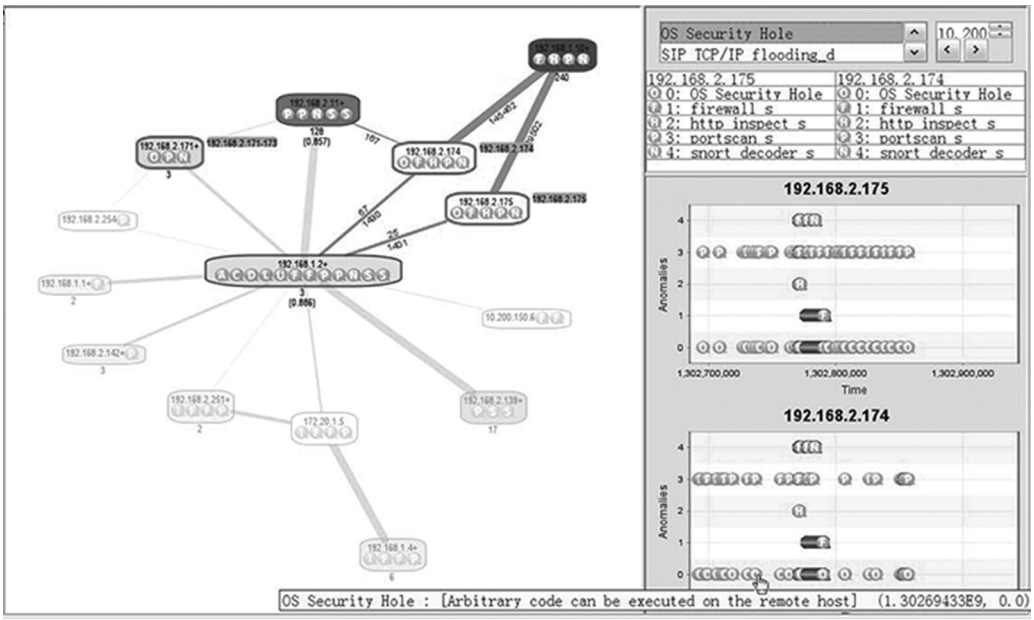



**FIGURE 4.16**: The hosts with security holes and the cross-subnet port scans from 192.168.2.174/175. (From Lei Shi et al., *IEEE International Conference on Big Data*, pages 606–612, 2013 [66].)

#### 4.4.1.3 DoS attacks

One critical machine that John examines is the corporate external web server (172.20.1.5). With the compressed graph, the server is easily located by its unique connection patterns. A single click on the host shows up a noteworthy anomaly icon (I) on the morning of the first day, suggesting that there could be Denial-of-Service (DoS) attacks through SIP. John further drills down to that period with the time range selector and highlights the web server's egocentric traffic graph. Figure 4.17 confirms the potential DoS attacks from the external hosts 10.200.150.201, 206–209, through the anomalies happening simultaneously with the web server.

#### 4.4.1.4 Botnet detection

In this example, NSAV is able to detect malware infection and botnet behavior via Internet Relay Chat (IRC) communication channels. In the first subgraph of the network traffic, as in Figure 4.18, it is identified that the $I$ and $M$ icons appeared frequently and almost in couples in reverse directions. A selection of the $IRC-Malware-Infection\_s$ anomaly (icon $I$) in the anomaly type list reveals three group of hosts highlighted in red in Figure 4.18. They are all workstations having enormous IRC connections to a portion of 12 websites (10.32.5.*), with the potential to be compromised botnet clients. Further selecting two typical workstations (172.23.123.105, 172.23.231.174) and websites (10.32.5.50, 10.32.5.52), their temporal anomaly distributions are plotted in the right panel of Figure 4.18.

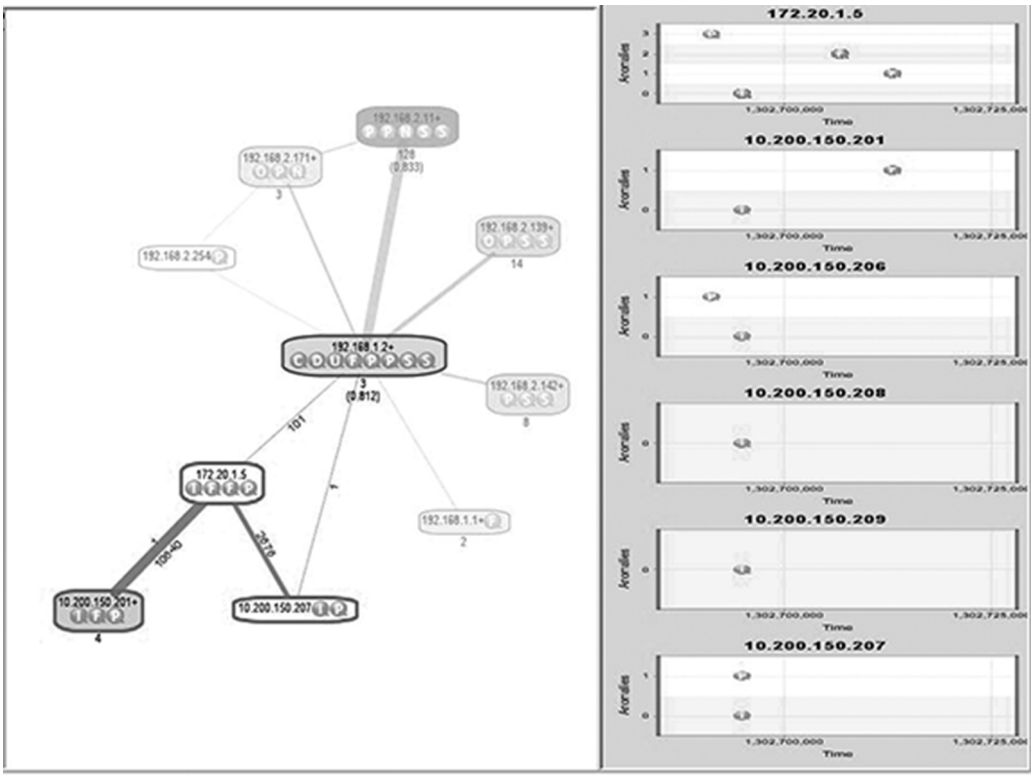



**FIGURE 4.17**: DoS attacks against 172.20.1.5 (corporate web server) from 10.200.150.201, 206–209. (From Lei Shi et al., *IEEE International Conference on Big Data*, pages 606–612, 2013 [66].)

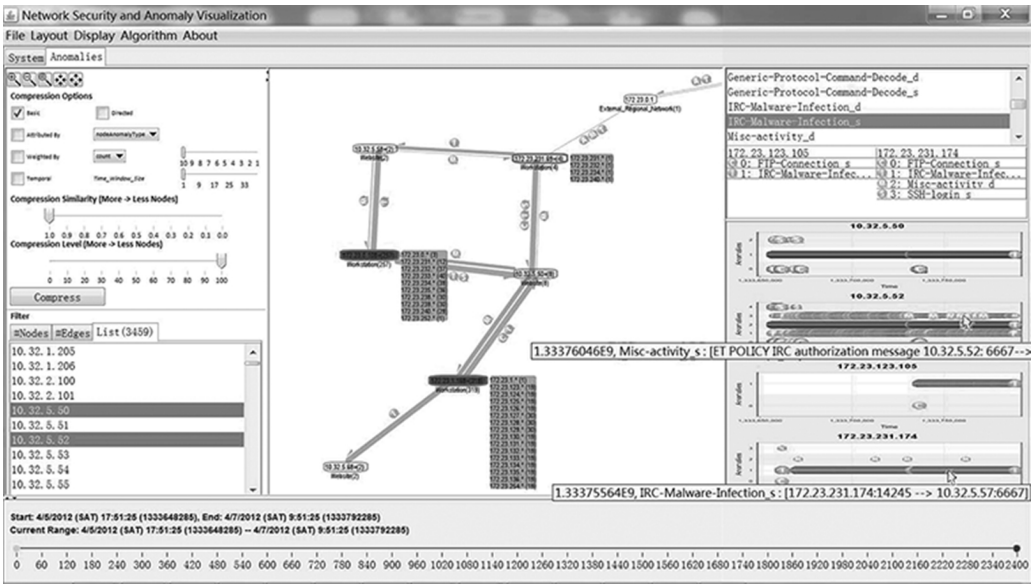



**FIGURE 4.18**: Three groups of machines in heavy IRC traffic with the websites through port 6667 suggesting suspicious botnet infections. (From Lei Shi et al., *IEEE International Conference on Big Data*, pages 606–612, 2013 [66].)

It is shown that the IRC traffic exchanged with the websites overwhelm in the whole inspected period. Note that nine of the websites (10.32.5.51-59) reply with the IRC authorization message (icon $M$), indicating the establishment of potential botnet server-client connections.

The host 172.23.231.174 (all-time IRC client) shows fine-grained patterns: the connections are composed of two temporal stages, indicated by a small gap in the middle of the anomaly panel. A drill-down analysis at this gap shows that the first stage ends up with a large port (43325) and the second stage starts with a small port (1185). After checking the anomaly file, we conclude that the IRC traffic from the workstations are programmed, with sequentially enumerated source ports. It verifies the hypothesis that these hosts have been compromised as botnet clients.

### 4.4.2 ENAVis for enterprise network security management

Enterprise Network Activities Visualization (ENAVis) [73] is a tool for visualizing the network activities among hosts (domains), users and applications (HUA) graph model. The network connectivity graphs can be built based on the various combinations of states (H, U, A, HU, HA, UA, HUA). The hierarchical, heterogeneous graph representation of HUA data, enabled by the local context information collected from enterprise networks, captures the dynamic relationship and interaction between machines and user applications. Figure 4.19 illustrates such HUA graph visualization in enterprise network security management.

#### 4.4.2.1 User and application-level policy compliance

In this example, suppose an enterprise management team needs to know whether their employees have complied with the company's network usage policy and whether the current mechanisms are adequate for enforcing the policy. A financial intranet server's access policy

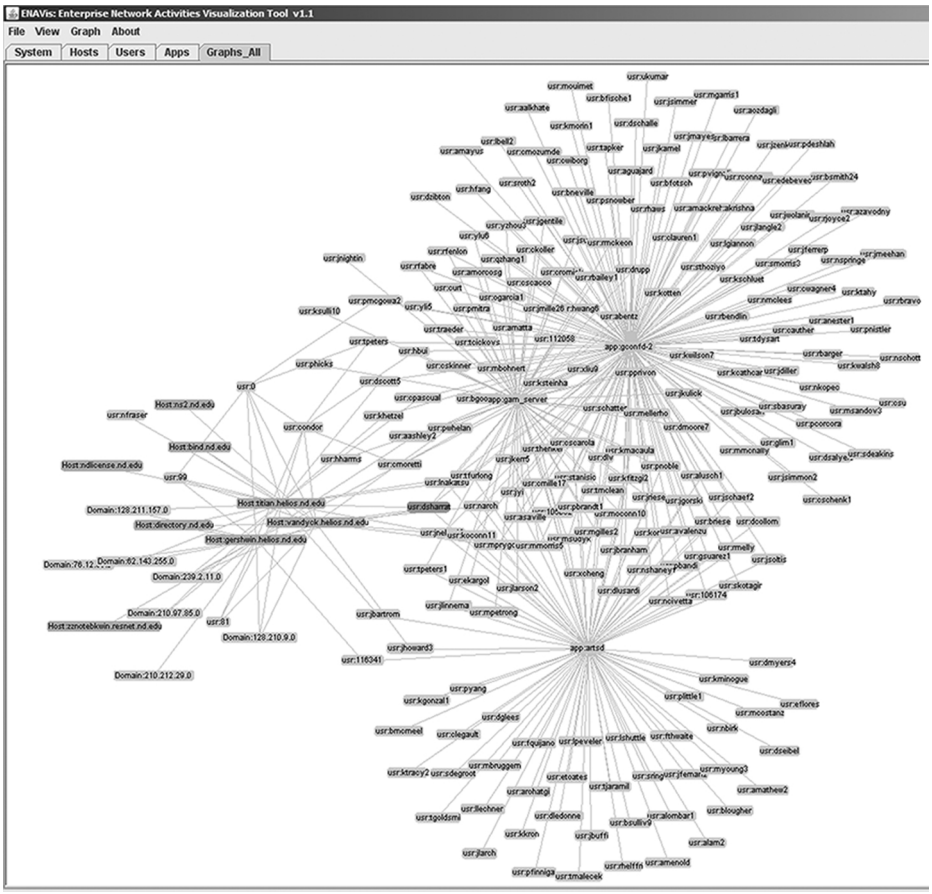



**FIGURE 4.19**: Hierarchical host-user-application (HUA) graph model in the *ENAVis* tool. (From Qi Liao et al., *Proceedings of the USENIX 22nd Large Installation System Administration Conference (LISA '08)*, pages 59-74, San Diego, CA, November 9–14 2008 [73].)

is defined such that only authorized users can access or even see the financial system. To that end, a set of host-based firewall rules are put in place on the finance server (host name *finance*) with restrictions to the hosts (host name *concert*) of authorized finance personnel on the company campus.

Since host to user mapping is dynamic, the notion of host as identity quickly breaks down (see Figure 4.20). Suppose in the same environment that *ssh* connectivity was enabled on the network. In the scenario of Figure 4.20, an unauthorized user *qliao* connects from *IrishFB.nd.edu* with X11 forwarding to *concert.cse.nd.edu* and launches an instance of *firefox* to access the finance web server. In a multiuser environment, where multiple users are logged on to the same machine and make network connections, other tools have no way of differentiating those connections because the connections all have the same source IP. Similarly, the legitimate user *striegel* may carelessly connect from a *Starbucks* shop to his office desktop *concert* in order to access a financial account just for convenience. Neither of these two cases is desirable and is a violation of the policy because the original intent of the policy was that anyone not part of the finance department should not be able to access the finance host.

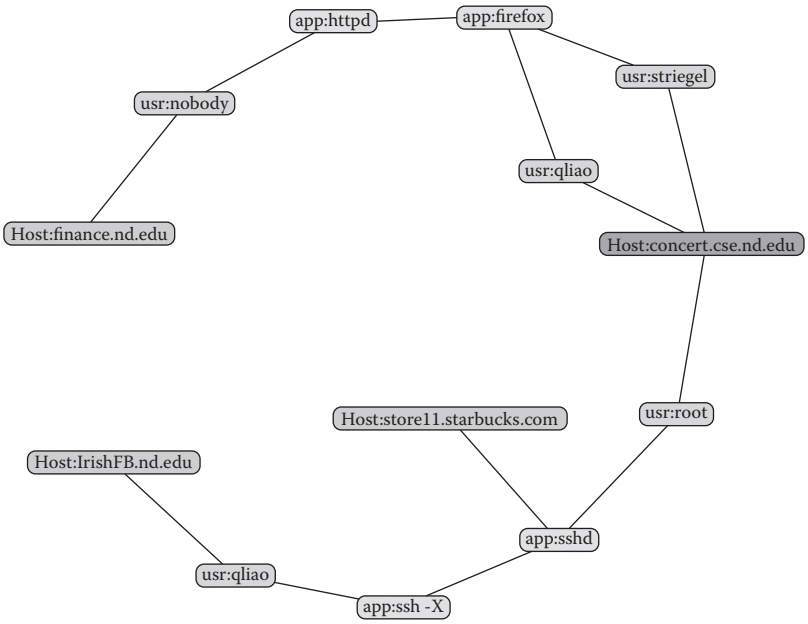



**FIGURE 4.20**: HUA graph captures two possible host IP ACL policy violations caused by the unintentional configuration on host *concert* without `ssh` restriction. (From Qi Liao et al., *Proceedings of the USENIX 22nd Large Installation System Administration Conference (LISA '08)*, pages 59–74, San Diego, CA, November 9–14 2008 [73].)

#### 4.4.2.2 Machine compromise and attack analysis

In this example, a phishing e-mail pretending to be from the IT department in an enterprise network claims the system has been upgraded and requires all users to send in their passwords, or their accounts will be suspended. Some users are suspected to be victims of the scam and may have their system compromised.

An administrator generates HUA network graphs and highlights the problem user node (*jdoe*) (as in Figure 4.21). It is straightforward to see which hosts the user has touched during the time frame and what applications the user used. The file access information logged by `lsof` is kept in the master database and a single query would reveal all files that user ID has touched among all the hosts. In this case, a visual HUA graph is helpful to see what hosts and users that a compromised user account has contacted and which applications it has attempted to launch. This helps expedite significantly such an investigation should it occur.

### 4.4.3 OnionGraph for heterogeneous networks

OnionGraph [71] allows a hierarchical grouping of nodes based on not only topological structure, but node attributes as well. The graph model is useful in a network management setting. Similar to the earlier mentioned HUA graph model, OnionGraphs can be applied to visually analyze HUA network graphs. Suppose a network administrator needs to analyze log data collected from his or her managed network. Since there must be a starting point, he or she starts with the most concise HUA network visualization in Figure 4.22. Not overwhelmed by the number of nodes and connections, from this top-level graph, the administrator easily finds that there were 128 users logged on 601 internal hosts running 298 unique applications, which connected to either internal hosts or 2802 external domains.

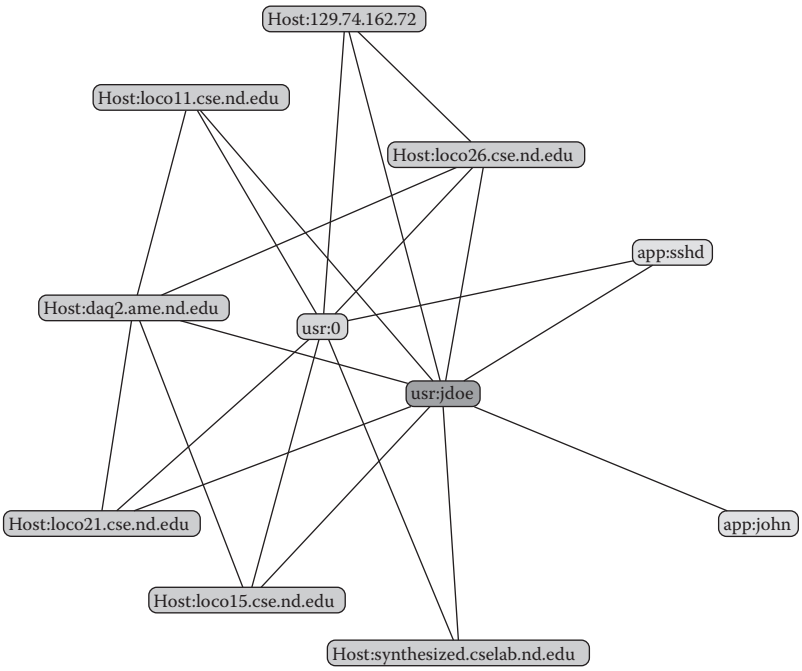



**FIGURE 4.21**: The HUA network graph reveals a highlighted user (jdoe) has logged on seven machines via `ssh` and has used the application John the Ripper to crack password files on those machines. (From Qi Liao et al., *Proceedings of the USENIX 22nd Large Installation System Administration Conference (LISA '08)*, pages 59–74, San Diego, CA, November 9–14 2008 [73].)

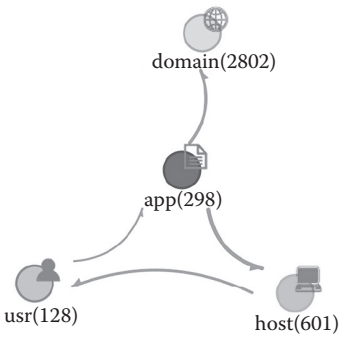



**FIGURE 4.22**: A top-level HUA OnionGraph gives the most concise and scalable visualization on large enterprise activity networks. (From Lei Shi et al. *IEEE Pacific Visualization Symposium*, pages 89–96, 2014 [71].)

The administrator makes a few interesting observations when moving from the initial "heterogeneous groups" to "RRE groups," on each node type. First, the *app* nodes are split into five subgroups, as shown in Figure 4.23, displayed by neighborhood charts: (1) the majority of applications [apps (217) node] are connected to only internal hosts by users (focused node in the graph); (2) six applications [apps (6) node] are connected to only external domains by users; (3) 69 applications [apps (69) node] are connected to both internal hosts and external domains by users; (4) five applications [apps (5) node] do not

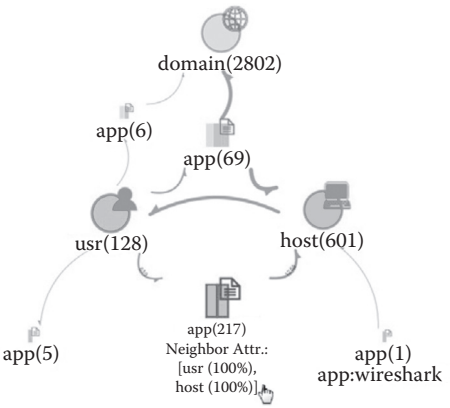



**FIGURE 4.23**: Expanded view on application nodes in OnionGraphs reveals a suspicious application (wireshark) possibly sniffing packets on the network by a malicious user. (From Lei Shi et al. *IEEE Pacific Visualization Symposium*, pages 89–96, 2014 [71].)

make network connections; and (5) one application [app (1) app: wireshark] run by an unknown user talked to a few internal hosts. Type-1 apps contain predominantly scientific computing programs while Type-2 and Type-3 have significantly more generic network applications such as ssh, firefox, ftp, and so on. In particular, the Type-5 node containing only one app (wireshark) is clearly suspicious, possibly leveraged by a malicious user to sniff packets on the network.

In addition to expansion of application nodes, the administrator may further divide the user nodes into two groups (Figure 4.24): 127 users that had run applications to connect to other computers; and the only user who does not run applications to make network connections. The Type-1 users are primarily enterprise users who are allowed to run scientific

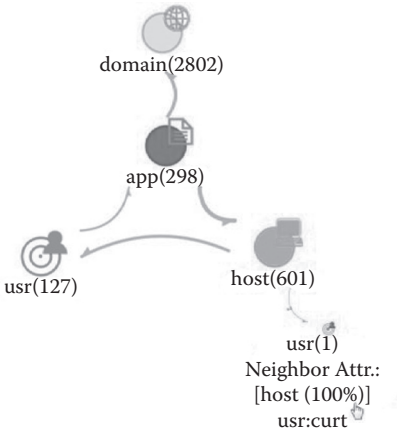



**FIGURE 4.24**: Expanded view on user nodes in OnionGraphs separates enterprise users with privileged users in network activities. (From Lei Shi et al. *IEEE Pacific Visualization Symposium*, pages 89–96, 2014 [71].)

programs. The Type-2 user is the system administrator. As policy compliance auditing, it is clear that normal users and privileged users have distinguished activity patterns.

### 4.4.4 OntoVis for terrorism networks

OntoVis [74] prunes a large, heterogeneous network according to both structural abstraction and semantic abstraction, thus making large networks manageable and reducing visual analytic complexity. Graph visualization has applications in social networks as well as for national security, such as terrorism domains. The original graph of a terrorism network (Figure 4.25a) is too large to analyze effectively while the ontology graph of the same network (Figure 4.25b) reduces such a network into only nine node types including terrorist organization, classification, terrorist, legal case, country/area, attack, attack target, and weapon and tactic.

Al-Qaeda is one of the most dominant organizations in the network and is related to most terrorist organizations, which can be further clustered. Terrorist organizations are consistent with their locations. For example, Al-Qaeda connects to the cluster at Gaza Strip, West Bank, and another cluster at Kashmir, India, and Pakistan. By investigating the attributes of related organizations, the researchers find that Al-Qaeda and most organizations related to it are classified as religious and nationalist/separatist organizations. In Figure 4.26a, organization clusters with the same classifications tend to connect to each other. Organizations that are located in the same areas tend to be of the same type.

In order to characterize the terrorist attacks, the researchers use attacking targets, tactics, and weapons to describe attacks. In Figure 4.26(b), the green nodes are attacks, the light blue ones are targets, the purple ones are weapons, and the pink ones are tactics. Most attacks by Al-Qaeda are bombing attacks using explosive weapons. The attacking targets vary from business, airport, and government to diplomatic objects. The 9/11 attacks can also be classified as unconventional attacks.

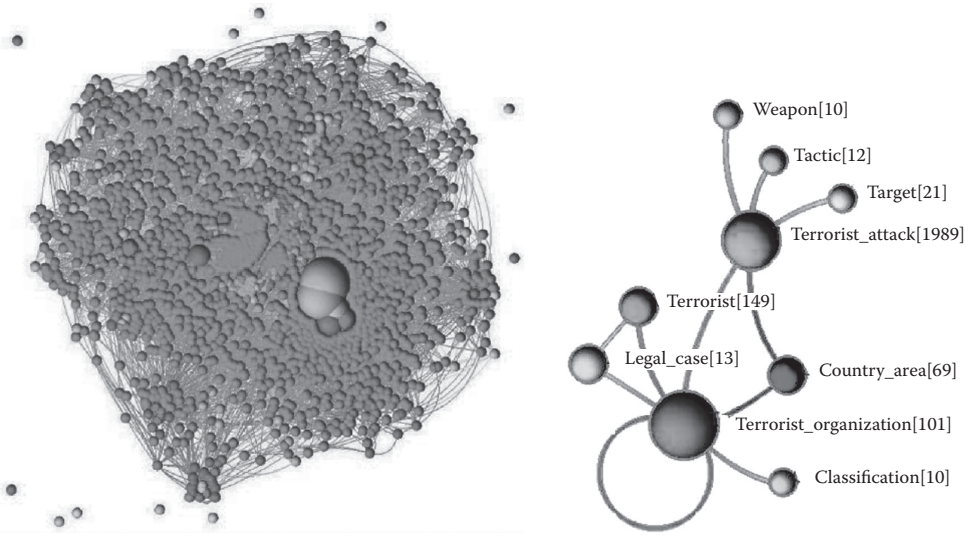



**FIGURE 4.25**: Ontology graph of the terrorism network. (a) Graphs of over 2000 nodes and 8000 edges creates an overwhelming visual complexity; (b) Onto-graph of only nine nodes. (From Zeqian Shen et al., *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1427–1439, 2006. [74].)

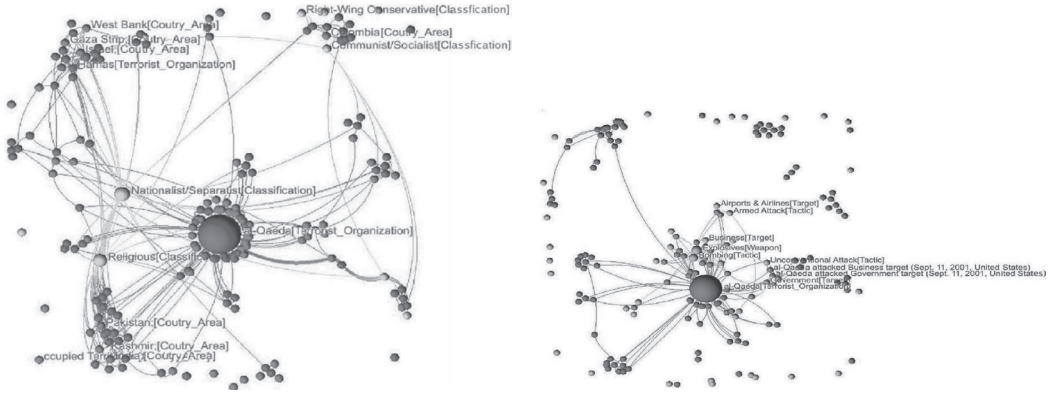



**FIGURE 4.26**: Visualization of various components of terrorist networks and their relationships. (a) Terrorist organizations, their locations, and classifications; (b) Terrorist attacks, their targets, weapons used, and attack tactics. (From Zeqian Shen et al., *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1427–1439, 2006. [74].)

## 4.5 Summary and Future Directions

In this chapter, we have discussed the state-of-the-art visualization methods to display large complex networks. The main challenge lies in the unique characteristics of these networks collected in the Big Data era: the huge volume in both network size and complex connections; the increasing variety on the data source, network topology, and the node/edge attributes and the changing dynamics of the network structure. We focus on the first two challenges and describe two kinds of visualization approaches: the large graph drawing method, and the visual abstraction method on large multivariate networks. The graph drawing methods propose new algorithms to support the layout of large graphs with up to millions of nodes. The visual abstraction methods study the algorithms to cluster, compress, and aggregate large graphs into smaller abstractions. These abstractions are visualized by new metaphor designs and manipulated by users through novel interaction models. We also introduce one application domain in security where the large complex network visualization methods are successfully applied to support the visual analysis from end users.

Despite the many existing works in this area, we are far from perfection in connecting the dots between large complex networks and domain users through visualizations. In the future, we foresee some important topic threads where many questions remain unanswered, though the area is largely open and new efforts should not be limited to the following list.

**Theory.** While the Big Data surge keeps on going, how people visually understand the result on large complex networks remains open. In particular, given that the graph drawing algorithms can not scale to an ultra-large network size and visual abstraction is inevitable, what is the best method and visual design for the abstraction to meet the goal of effectively analyzing big complex networks?

**Algorithm.** Rather than focusing on the layout algorithm to place nodes for the large network, new transformation and mining algorithms on graphs and networks can be more important in the Big Data context. Compared to the general-purpose graph mining algorithms, the focus on visualization calls for novel mining algorithms to detect interpretable patterns from the large complex network. In this perspective, the joint optimization of both the data mining objective and the visualization objective can be an interesting direction.

**Framework and system.** In this area of the application research, the design and implementation of software frameworks and systems to visualize the incoming large complex newtork, presumptively in a web context, are in most cases more visible than the theoretical and algorithmic studies. We need online systems that allow users to upload their large networks for visualization and/or to visually navigate the existing large network collections for valuable insights.

## References


1. Peter Eades. A heuristic for graph drawing. In *Congressus Numerantium*, volume 42, pages 149–160, 1984.
2. Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software, Practice & Experience*, 21(11):1129–1164, 1991.
3. Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.
4. Emden R. Gansner, Yehuda Koren, and Stephen North. Graph drawing by stress majorization. In *Proceedings of International Symposium on Graph Drawing*, pages 239–250, 2004.
5. Colin Ware. *Information visualization: Perception for design*. Elsevier, Amsterdam, the Netherlands, 2012.
6. Mohammad Ghoniem, Jean-Daniel Fekete, and Philippe Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. In *IEEE Symposium on Information Visualization*, pages 17–24, 2004.
7. Hyunmo Kang, Catherine Plaisant, Bongshin Lee, and Benjamin B. Bederson. Netlens: Iterative exploration of content-actor network data. *Information Visualization*, 6(1):18–31, 2007.
8. Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall PTR, Upper Saddle River, NJ, 1998.
9. Emden R. Gansner and Stephen North. An open graph visualization system and its applications to software engineering. *Software, Practice & Experience*, 30:1203–1233, 2000.
10. Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *ICWSM*, 8:361–362, 2009.
11. OGDF. Open graph drawing framework, http://www.ogdf.net/.
12. David Auber. Tulip—a huge graph visualization framework. In *Graph Drawing Software*, pages 105–126, 2004.
13. Fabian Beck, Michael Burch, Stephan Diehl, and Daniel Weiskopf. The state of the art in visualizing dynamic graphs. In *Euro-Vis'14 State-of-the-Art Reports*, pages 83–103, 2014.

14. 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-Verlag, Berlin, Germany, 2004.

15. I. Bruss and A. Frick. Fast interactive 3-D graph visualization. *LNCS*, 1027:99–11, 1995.

16. Yifan Hu. Efficient and high quality force-directed graph drawing. *Mathematica Journal*, 10:37–71, 2005.

17. J. Barnes and P. Hut. A hierarchical O(NlogN) force-calculation algorithm. *Nature*, 324:446–449, 1986.

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

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

20. S. Pfalzner and P. Gibbon. *Many-body tree methods in physics*. Cambridge University Press, Cambridge, 1996.

21. 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.

22. L. F. Greengard. *The rapid evaluation of potential fields in particle systems*. The MIT Press, Cambridge, Massachusetts, 1988.

23. 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-Verlag, Berlin, Germany, 2004.

24. 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.

25. Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, New York, NY, USA, 1995. ACM.

26. 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.

27. Y. Hu and J. A. Scott. A multilevel algorithm for wavefront reduction. *SIAM Journal on Scientific Computing*, 23:1352–1375, 2001.

28. I. Safro, D. Ron, and A. Brandt. Multilevel algorithms for linear ordering problems. *J. Exp. Algorithmics*, 13:1.4–1.20, 2009.

29. C. Walshaw. A multilevel approach to the travelling salesman problem. *Oper. Res.*, 50:862–877, 2002.

30. C. Walshaw. Multilevel refinement for combinatorial optimisation problems. *Annals of Operations Research*, pages 325–372, 2004.

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. R. Hadany and D. Harel. A multi-scale algorithm for drawing graphs nicely. *Discrete Applied Mathematics*, 113:3–21, 2001.

33. D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. *Journal of graph algorithms and applications*, 6:179–202, 2002.

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

35. Y. Hu. A gallery of large graphs. http://research.att.com/yifanhu/GALLERY/GRAPHS/index.html.

36. T. A. Davis and Y. Hu. University of Florida Sparse Matrix Collection. *ACM Transaction on Mathematical Software*, 38:1–18, 2011.

37. J. B. Kruskal. Multidimensioal scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29:1–27, 1964.

38. 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.

39. 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, Berlin, Germany, 2009.

40. W. S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17:401–419, 1952.

41. G. Young and A. S. Householder. Discussion of a set of points in terms of their mutual distances. *Psychometrica*, 3:19–22, 1938.

42. 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, Cambridge, MA, 2003.

43. Ulrik Brandes and Christian 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.

44. Emden R. Gansner, Yifan Hu, and Stephen C. North. A maxent-stress model for graph layout. *IEEE Trans. Vis. Comput. Graph.*, 19(6):927–940, 2013.

45. Marc Khoury, Yifan Hu, Shankar Krishnan, and Carlos Eduardo Scheidegger. Drawing large graphs by low-rank stress majorization. *Comput. Graph. Forum*, 31(3):975–984, 2012.

46. Emden R. Gansner, Yifan Hu, and Shankar Krishnan. Coast: A convex optimization approach to stress-based embedding. In Stephen Wismath and Alexander Wolff, editors, *Graph Drawing*, volume 8242 of *Lecture Notes in Computer Science*, pages 268–279. Springer-Verlag, Berlin, Germany, 2013.

47. 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.

48. D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. *LNCS*, pages 207–219, 2002.

49. Cytoscape. An open source platform for complex network analysis and visualization. //http://www.cytoscape.org/.

50. Shailesh Tripathi, Matthias Dehmer, and Frank Emmert-Streib. NetBioV: an R package for visualizing large network data in biology and medicine. *Bioinformatics (Oxford, England)*, 30(19):2834–2836, October 2014.

51. D3.js. Data-driven documents. http://d3js.org.

52. Tim Dwyer, Kim Marriott, and Michael Wybrow. Dunnart: A constraint-based network diagram authoring tool. In IoannisG. Tollis and Maurizio Patrignani, editors, *Graph Drawing*, volume 5417 of *Lecture Notes in Computer Science*, pages 420–431. Springer, Berlin, Heidelberg, 2009.

53. SNAP. Stanford large network dataset collection. http://snap.stanford.edu/data.

54. Koblenz. The Koblenz network collection. http://konect.uni-koblenz.de.

55. GraphArchive. Exchange and archive system for graphs. http://www.graph-archive.org/.

56. House of Graphs. A database of interesting graphs. https://hog.grinvin.org/.

57. Ivan Herman, Guy Melancon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

58. 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.

59. Andreas Kerren, Helen Purchase, and Matthew O. Ward. *Multivariate Network Visualization (Proc. Dagstuhl Seminar 13201)*. Springer-Verlag, Berlin, Germany, 2014.

60. Douglas Brent West. *Introduction to graph theory*, volume 2. Prentice Hall, Upper Saddle River, NJ 2001.

61. Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 538–543, 2002.

62. Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: concepts and techniques*. Elsevier, Amsterdam, the Netherlands, 2011.

63. Lei Shi, Nan Cao, Shixia Liu, Weihong Qian, Li Tan, Guodong Wang, Jimeng Sun, and Ching-Yung Lin. HiMap: Adaptive visualization of large-scale online social networks. In *Proceedings of the IEEE Pacific Visualization Symposium*, pages 41–48, 2009.

64. M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, 2004.

65. Weiwei Cui, Hong Zhou, Huamin Qu, Pak Chung Wong, and Xiaoming Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1277–1284, 2008.

66. Lei Shi, Qi Liao, Xiaohua Sun, Yarui Chen, and Chuang Lin. Scalable network traffic visualization using compressed graphs. In *IEEE International Conference on Big Data*, pages 606–612, 2013.

67. Francois Lorrain and Harrison C. White. Structural equivalence of individuals in social networks. *The Journal of Mathematical Sociology*, 1(1):49–80, 1971.

68. Charis Papadopoulos and Costas Voglis. Drawing graphs using modular decomposition. *Journal of Graph Algorithms and Applications*, 11(2):481–511, 2007.

69. Cody Dunne and Ben Shneiderman. Motif simplification: Improving network visualization readability with fan and parallel glyphs. In *Proceedings of the International Conference on Human Factors in Computing Systems (CHI'13)*, pages 3247–3256, 2013.

70. Tim Dwyer, Nathalie Henry Riche, Kim Marriott, and Christopher Mears. Edge compression techniques for visualization of dense directed graphs. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2596–2605, 2013.

71. Lei Shi, Qi Liao, Hanghang Tong, Yifan Hu, Yue Zhao, and Chuang Lin. Hierarchical focus+context heterogeneous network visualization. In *IEEE Pacific Visualization Symposium*, pages 89–96, 2014.

72. Douglas R. White and Karl P. Reitz. Graph and semigroup homomorphisms on networks of relations. *Social Networks*, 5(2):193–234, 1983.

73. Qi Liao, Andrew Blaich, Aaron Striegel, and Douglas Thain. ENAVis: Enterprise network activities visualization. In *Proceedings of the USENIX 22nd Large Installation System Administration Conference (LISA '08)*, pages 59–74, San Diego, CA, November 9-14 2008.

74. Zeqian Shen, Kwan-Liu Ma, and Tina Eliassi-Rad. Visual analysis of large heterogeneous social networks by semantic and structure. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1427–1439, 2006.