Visual Exploration of Complex Time-Varying Graphs

Gautam Kumar and Michael Garland

Abstract— Many graph drawing and visualization algorithms, such as force-directed layout and line-dot rendering, work very well on relatively small and sparse graphs. However, they often produce extremely tangled results and exhibit impractical running times for highly non-planar graphs with large edge density. And very few graph layout algorithms support dynamic time-varying graphs; applying them independently to each frame produces distracting temporally incoherent visualizations. We have developed a new visualization technique based on a novel approach to hierarchically structuring dense graphs via stratification. Using this structure, we formulate a hierarchical force-directed layout algorithm that is both efficient and produces quality graph layouts. The stratification of the graph also allows us to present views of the data that abstract away many small details of its structure. Rather than displaying all edges and nodes at once, resulting in a convoluted rendering, we present an interactive tool that filters edges and nodes using the graph hierarchy and allows users to drill down into the graph for details. Our layout algorithm also accommodates time-varying graphs in a natural way, producing a temporally coherent animation that can be used to analyze and extract trends from dynamic graph data. For example, we demonstrate the use of our method to explore financial correlation data for the U.S. stock market in the period from 1990 to 2005. The user can easily analyze the time-varying correlation graph of the market, uncovering information such as market sector trends, representative stocks for portfolio construction, and the interrelationship of stocks over time.

Index Terms—Graph and network visualization, financial data visualization, hierarchy visualization, time series data.

1 Introduction

Effectively visualizing large sets of relationships is a growing need in many fields. In contexts such as social networks, telecommunications, Internet networks, homeland security, and financial research, graph visualization is a standard form of extracting and conveying information. In all of these contexts, graphs are becoming increasingly complex, and in many cases, the graph structure changes over time. Force-directed layout algorithms, which attempt to find a minimal energy configuration, work well for visualizing relatively sparse static graphs. However, when applied to complex highly non-planar datasets, these spring-based methods are slow to converge and frequently produce extremely tangled results. When rendered with the conventional line-dot technique, the resulting graph is often so cluttered that the user is unable to recognize many important patterns within the complex data. Moreover, many graph visualization tools often overlook the handling of time-varying graphs and don't preserve temporal coherence. The resulting animation from laying out frames in dynamic graphs independently exhibits spurious movements from frame to frame, masking motion due to actual structural changes. As graphs become progressively more complex and dynamic, solving this problem will become vital to graph analysis.

We have developed an interactive visualization technique in which users explore a hierarchical representation of a complex graph, enabling rapid discovery of meaningful structure among the nodes. We construct a graph hierarchy by stratifying nodes into different levels so that central and representative nodes in the graph are emphasized. Nodes are then organized into interconnected groupings in a tree, enabling tree families to be placed close to each other in our layout algorithm. In contrast to prior work where graph hierarchies are built by repeated coarsening, we construct our hierarchy based on edge distribution. In many cases, complex non-planar graphs have vertex degrees distributed according to a power law, and our stratified hierarchy exposes the underlying structure of such graphs.

Based on the graph hierarchy, we also propose a new global/local layout scheme that recursively traverses the hierarchy and rapidly converges to an aesthetically pleasing end result. Our novel local layout algorithm uses a force-directed algorithm so that neighboring nodes are close to each other, while also using Lloyd relaxation to ensure the layout is well spaced. When extended to time-varying graphs, our layout produces clear animations of dynamic graphs that preserve coherence across frames and ensures that any motion reflects actual data changes. Users are thus able to easily perceive interesting structural trends over time.

Our rendering scheme takes advantage of modern graphics hardware by integrating user interaction with stylistic visual representations to abstract and explore graphs. Rather than displaying all edges and nodes at once, which can produce a convoluted image, we filter edges and nodes using the graph hierarchy and allow users to drill down into the graph for details. Our visualization emphasizes related clusters of nodes by clearly depicting cliques and families in the hierarchy. We improve on past methods by simplifying highly interconnected networks, enhancing the visual clarity of graph rendering, and incorporating time-varying systems.

As an example application, we utilize our tool to visualize price return correlations between stocks in the S&P 500. We construct a graph where two stocks are connected by an edge when their returns correlations are above a selected threshold. Such graphs are known to have power law degree distributions [3]. Analyzing the correlations amongst securities is central to Modern Portfolio Theory [18] where risk is managed through diversification of investments.

2 Related Work

Initially applied to relatively small and sparse graphs, early successful graph layout algorithms were typically force-directed. This approach was pioneered by Eades [8]. Kamada and Kawai [16] modeled a graph as a complete system of linear springs, and Fruchterman and Reingold [10] refined and simplified their force calculations. These methods are flexible and easily implemented, but their initial focus was on graphs of only up to 100 vertices. On larger, denser graphs (e.g., with power law edge distribution) they converge slowly, if at all [15], and the results were often cluttered and disorganized.

More recently, many interesting approaches to visualizing complex highly non-planar graphs have been developed. Harel and Koren [15] developed a multi-scale algorithm that can improve the running time of any force-directed method. Hachul and Junger [14] proposed a multi-level algorithm using potential fields that can achieve the same asymptotic running time as single-level methods. Andersen *et al.* [1] partitioned edges into local and global sets and used a force-directed method emphasizing local edges. Chan *et al.* [4],

- *Gautam Kumar is with the University of Illinois at Urbana-Champaign, E-mail: gvkumar@uiuc.edu.*
- *Michael Garland is with NVIDIA, E-mail: mjgarland@acm.org.*

Manuscript received 31 March 2006; accepted 1 August 2006; posted online 6 November 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

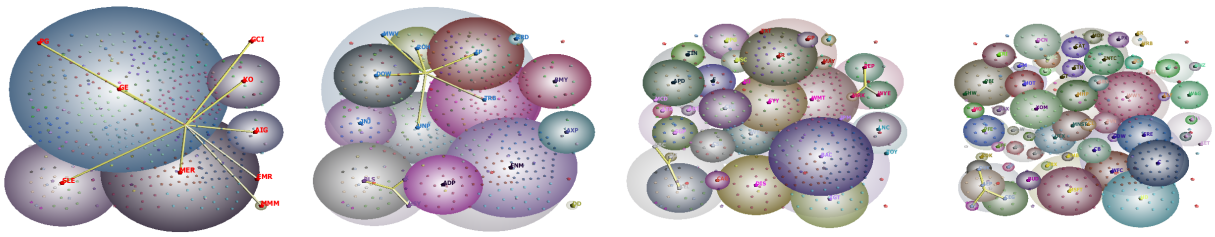

Fig. 1. Our visualization tool builds a stratified hierarchy from dense and non-planar graphs. In visualizing price correlations in the S&P 500 dataset, our tool automatically extracts the most highly correlated stocks and clusters related stocks together in the layout.

similar to our method, stratified the graph based on degree and used the Fruchterman-Reingold method on each layer. Rather than using a stratified hierarchy to accelerate layout, Gansner *et al.* [11] use a coarsening-based hierarchy to provide a topological fisheye view at multiple levels of detail. Similarly, van Ham and van Wijk [24] used an interactive fisheye scheme and spherical clustering to decompose the graph and smoothly interpolate between various levels of detail. Voronoi diagrams have also been integrated into graph drawing by recursively decomposing screen space [22] and as a post-processing step in laying out labeled or non-point nodes [6, 12].

Our stratification scheme makes use of prior work on node ranking in order to find the most central or hub nodes. Kleinberg [17] developed the HITS system of ranking by an authority measure. Newman [19] developed the Betweenness-Centrality measure which counts the number of shortest paths that pass through each node. Wu *et al.* [25] tested both of these measures, in addition to random sampling and degree ranking, when developing a data mining approach to simplify and cluster power law graphs using geodesic clustering. We develop our own method for ranking nodes using weights between every pair of vertices.

In contrast to static graphs, little research has been done on visualization of time-varying graphs. Most work on dynamic graph drawing is related to the online problem—only information about previous graphs is used for computing a layout. A prominent example for hierarchical directed acyclic graphs by North [20] incrementally updates a layout preserving the user's mental map. Gorg *et al.* [13] presented an offline approach (all graph changes are known beforehand) which also used mental map metrics.

Although these approaches to drawing complex graphs offer significant improvements particularly in running time, most still offer little interactivity, don't sufficiently abstract the complexity of the graph, and generally ignore integration with dynamic graphs. Our method extends very naturally to time-varying data by applying the same static force-directed algorithm on the differences between frames. We are also capable of rendering graphs using a structured hierarchy and intuitive visual symbols. User interaction is organized around an abstracted overview with zooming and filtering capabilities.

3 Constructing a Hierarchy

We use a graph hierarchy to achieve both faster convergence and better global positioning of nodes in our layout algorithm. The hierarchy also allows us to filter edges and nodes during rendering so that the user is not overwhelmed by too much complexity (see Figure 2, for example).

Several other graph visualization tools [15, 11], build hierarchies based on graph coarsening with the goal of preserving the structure of the complex graph. However, since force-directed methods do not work well on highly non-planar graphs, our goal is to break the complex structure and achieve much more planar graphs when viewing level by level. Therefore, our hierarchy is instead based on edge distribution and is built to reflect the underlying structure of complex graphs. These graphs, often with power law degree distributions, lend themselves well to a hierarchy since select nodes are highly connected while the majority of nodes are not. For example, in an airport network numerous regional airports can be grouped under the single major hub airport of the region. On the other hand, graphs with uniform edge distribution like meshes do not have an inherent hierarchy, and thus can be sufficiently positioned using early spring-based layout methods. Since our focus is on dense non-planar graphs, we stratify the nodes into separate layers and construct a hierarchical tree to group interconnected nodes together. Stratification into layers based on degree has been investigated by Chan *et al.* [4]; however, this method does not model the graph as a hierarchical tree and simply uses a constant number of levels, namely 3, for all graphs. A tree allows our layout algorithm to achieve better time complexity by applying the force-directed method on each individual tree family rather than the whole level. Also, our more sophisticated stratification emphasizes the authority of a node more than just degree and better supports the edge distributions of various data sets.

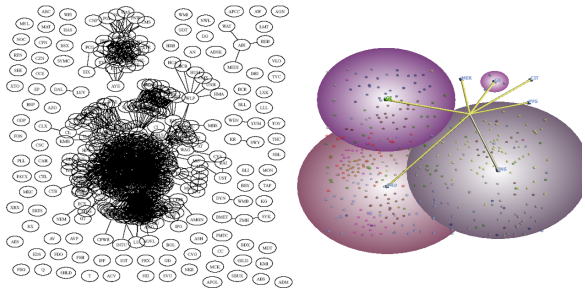

Fig. 2. Our system (right) produces an S&P 500 layout lacking the clutter typical of standard energy-based methods such as GraphViz.

3.1 Sorting the Nodes by Authority

We assume that we are given a graph $G = (V, E)$ with a weight w_{ij} assigned to each edge (i, j). The *authority* or *centrality* should reflect how representative it is of a group of nodes. We rank nodes by authority using the sorting factor

$$s_i = \sum_{j \in V} w_{ij}^2 \overline{w}_j \qquad (1)$$

where $\overline{w}_j$ is the mean weight of node j. This formulation is inspired by the HITS ranking system [17]. We use squared weights to give preference to nodes that are very representative of some nodes over those that are moderately representative of all nodes. The mean weight term ensures that the most central nodes are also representative of other less central nodes. By ranking the graph in this manner and visualizing the graph level-by-level, nodes are isolated with peer nodes of similar authority and the most authoritative nodes are immediately visible to the user.

3.2 Stratifying the Graph

To stratify the nodes into levels, we must first specify the desired depth of the tree. The user could provide this directly. However, it is typically more useful to automatically estimate an appropriate number of levels.

There is no truly optimal depth. Instead, we aim to find a depth that prevents levels with too many edges (which would yield visual clutter) while keeping the total number of levels low. Since many complex graphs have power law vertex degree distributions, we use this as the basis for our estimate.

We assume that the number of nodes of degree k is $Ck^{-\beta}$ for some constant C and $\beta > 1$. We can find C for a given graph by realizing that the total number of nodes n is proportional to the Riemann-Zeta function.

$$n = C\sum_{k=1}^{\infty} k^{-\beta} = C\zeta(\beta) \Longrightarrow C = \frac{n}{\zeta(\beta)} \tag{2}$$

Given the degree histogram of the graph, we can find the value of β that best fits the histogram, as shown in Figure 3.

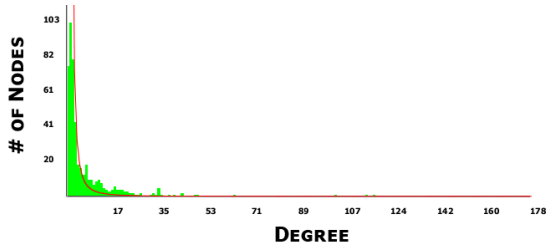

Fig. 3. A power law with exponent $\beta = 2$ (in red) provides the best fit for the degree histogram of the S&P 500 graph for August 1998.

Note that the useful range of β end where $\zeta(\beta)$ is very close to 1. Our system defaults to the range $1 < \beta \leq 14$ since $\zeta(14) \approx 1.000061$, but the user can optionally tighten this range to achieve a desirable number of nodes in the top level. Since higher values of β represent steeper power law curves, fewer levels are needed since fewer nodes are authoritative.

Given β, the depth of the tree will be $\log_\beta n$. To construct the tree, we iterate through nodes in order of authority s_i and place nodes in levels so that each level has an equal total degree.

3.3 Computing Families

Having sorted all nodes into levels, we must now pick the right parent for each node. We iterate through nodes in sorted order and for each node in level $l > 1$ we attempt to find the best parent from level $l-1$. In picking parents, we would like to avoid extremely imbalanced trees with large branching factors, as this will lead to clutter and poor performance during layout. This imbalance typically arises in power law graphs because the most central nodes are closely related to a majority of the graph, and thus appear to be good parents for most nodes. Enforcing the restriction that parents are in the immediately preceding level avoids this imbalance.

For a node p to be a good parent for node i, the node should have a high weight with both p and the neighbors of p. By neighbors of p we mean both graph neighbors (nodes sharing an edge with p) and tree neighbors (siblings and ancestors of p in the hierarchy). The number of common graph neighbors that a pair of nodes share is an important factor to consider [9]. Weight with tree neighbors is equally important to ensure that i is grouped with the correct family of nodes. Thus, we define the parent factor of child i with parent p to be an equally weighted sum of the weight between i and p, mean weight between i and p's graph neighbors, and mean weight between i and p's tree neighbors. We chose the parent that maximizes this measure.

This method is a fairly simple greedy algorithm and only takes into account the tree from the root to the current level. To produce better hierarchies, we apply an additional bottom-up relaxation phase that augments the parenting factor with the mean weight between p and i's children. The quality of this parent-child relationship will prove important since siblings will be positioned nearby during layout.

In the financial context, it has been noticed that building a hierarchy from price correlations tends to group stocks into industries [21].

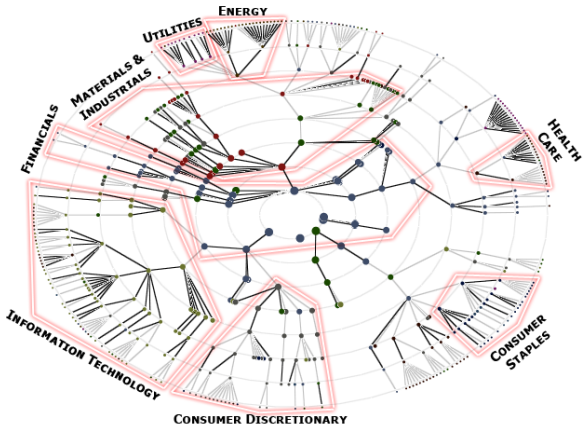

Fig. 4. The 2005 S&P 500 hierarchy with nodes colored by industry. Most siblings are in the same industry. Financials (light blue) dominate the top level due to high correlations with many other firms.

Our hierarchy construction preserves this property. Using the S&P 500 price correlation dataset in 2005, 77% of the average stock's tree siblings were in the same industry (see Figure 4).

4 Laying Out the Graph

Having constructed our graph hierarchy, we compute a planar layout of the graph using a hierarchical algorithm. Our goal is to quickly converge to a layout where related nodes are positioned close together. The hierarchy allows us to globally position entire subtrees and then locally beautify each group until convergence. Our algorithm combines a force-directed component to move connected nodes closer together and Lloyd relaxation to fairly utilize all the available screen space.

4.1 Allocating Screen Space

We want to ensure that all available screen space is used in layout, avoiding unnecessary congestion and graph shrinkage. To do this, we assign a desired area to each node before iterative layout. The total screen space is divided between top level nodes in proportion to the number of nodes in their subtrees. We then recursively divide each node's allotted space amongst its children in the same manner. From this fair hierarchical division of space, we can create weighted Voronoi diagrams, decompositions of space determined by distances to nodes, for each family in the tree.

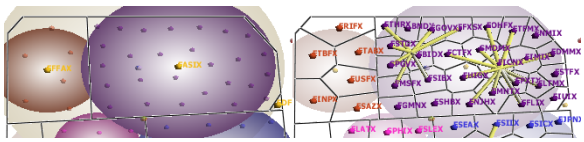

Fig. 6. Bottom levels of the Fidelity 2005 hierarchy. Each parent's weighted Voronoi cell (left) is divided among its children (right).

Figure 6 shows how a parent's Voronoi cell is divided among its children based on the number of descendants. These Voronoi diagrams are then used to integrate a kind of Lloyd relaxation [7]—nodes are moved to the center of their Voronoi cells—into the layout algorithm. In addition to improving space utilization, this also brings stability to potentially unstable force-directed algorithms and encourages faster convergence.

4.2 Iterative Layout

We combine Lloyd relaxation with a force-directed method in our iterative layout algorithm. Using force-directed algorithms on multi-level

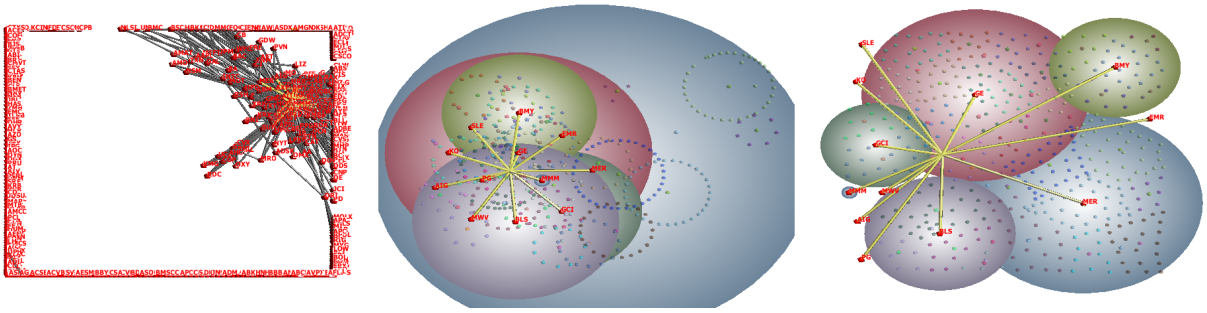

Fig. 5. The July 1995 S&P 500 graph laid out with force-directed minimization (left), plus stratification (center), and plus Lloyd-based screen allocation (right). Running times are 416, 3, and 16 seconds, respectively.

graphs is common in visualizing large graphs [15, 14, 1, 4] since they are easy to implement and can be quickly computed in a hierarchical tree.

Our method involves recursively traversing the hierarchy and computing layouts for each family as we move down the tree. The initial position for each top level node is a random position in the screen space. Since the most central nodes have the greatest effect on the layout, we perform our iterative local layout algorithm on the top level nodes alone until no node moves more than a small distance. With nodes initially placed in a random position in the parent's Voronoi cell, we repeat this iterative process for each subtree. Thus, we are able to achieve a fast and easily-implemented global/local layout scheme. Figure 5 compares the results of our method using force-directed minimization alone and hierarchical relaxation without Lloyd relaxation. Our hierarchy provides clear benefits in speed and aesthetics. Although the Voronoi computation takes time, Lloyd relaxation results in a better organization of the graph.

Our iterative relaxation scheme uses a scheduled weighting of force-directed and Lloyd terms where early iterations consist primarily of force-directed layout, smoothly transitioning to emphasize fair allocation in later iterations. For a node i with a set of siblings S_i, we compute the force vector acting on node i during iteration k as

$$f_i^k = (1-\alpha^k)v_i + \alpha^k \sum_{j \in S_i} f_{ij} \tag{3}$$

where v_i is the vector to the Voronoi centroid of node i, f_{ij} is the force exerted by node j on i. The transition constant α can be determined by the user, but our experience shows that a value of $\alpha = 0.95$ works well for all datasets we have tried.

We calculate forces between nodes using the Fruchterman-Reingold model [10] modified to account for edge weights. This model has the advantages of both speed and ease of implementation. However, this choice is not central to our method and other force-directed methods could be chosen instead.

Attractive forces along an edge serve to avoid long edges and edge intersections, and repulsive forces keep nodes from being too close together. If A_p is the area of the Voronoi cell for a node p with n_p children, the ideal distance between two children of p is

$$k = \sqrt{A_p / n_p} \tag{4}$$

The magnitudes of the attractive and repulsive forces between nodes at a distance d_{ij} will be

$$f_{ij}^a = d_{ij}^2 / k \qquad f_{ij}^r = -k^2 / d_{ij} \tag{5}$$

The force vector between two nodes is thus

$$f_{ij} = \begin{cases} u_{ij}(1-w_{ij})f_{ij}^r + u_{ij}w_{ij}f_{ij}^a & \text{if } (i,j) \text{ is an edge,} \\ u_{ij}(1-w_{ij})f_{ij}^r & \text{otherwise.} \end{cases} \tag{6}$$

where u_{ij} is the unit vector from i to j. Finally, we also enforce a maximum displacement limit on each node based on its parent's allocated space.

4.3 Extending the Layout for Time-Varying Graphs

Our iterative layout algorithm easily extends to create animations for time-varying graphs. In many applications, graph analysis does not occur only once, but several times over a time period. Studying changes in a network is equally, if not more, important than analyzing the structure of a static network. A central feature of our visualization tool is the ability to use our rich graph exploration capabilities at any point in time. Inputting a series of graphs into our program, the user can simply slide to different time periods to explore the graph or play an animation to discover trends.

In the static layout case, a good layout is able to position related nodes together, maintain an aesthetically pleasing and uncomplicated layout, and converge to a final layout quickly. In the dynamic case, we must add another metric: preserve temporal coherence across frames to avoid unnecessary motion caused by little or no change in the graph relationships. To do this, we choose an offline approach in our layout algorithm where all graph changes are known beforehand. This allows us to emphasize trends that persist over time and de-emphasize those that are due to minute momentary data changes.

Our financial dataset demonstrates a perfect application for visualizing a graph over time. In this context, we store a separate stock correlation array for each month in the past several years. We then use each of these arrays as a "keyframe" in an animation. We statically draw a graph for the first array in the time series using the iterative static layout algorithm described in the previous section. For subsequent frames, the initial layout is derived from the final layout of the preceding frame.

We must also modify our hierarchy construction algorithm so as to avoid drastic (and misleading) changes in layout due to small changes in correlation. When computing the sorting factor, we average the weights w_{ij} over a 3 frame window. Similarly, we average parent factors used in hierarchy creation.

Finally, we modify the weights used in our local layout algorithm to reduce meaningless movements. The static layout algorithm scales the force between two nodes by the weight w_{ij}. In the dynamic case, we instead scale by the *change* in weight w_{ij} from the previous frame. This very simple change drastically reduces spurious inter-frame movement, and allows the user to identify interesting changes in data much more reliably.

Figure 7 compares the results of simple per-frame static layout (top) with our temporally coherent layout (bottom). As expected, laying out every frame separately results in large amounts of spurious movement even though the weights are changing relatively little. Though we smooth the layout, we also do not want to hide movement caused by significant weight changes. We have found that our gradual smoothing approach preserves these "shocks", such as the market crash of 2001 where dramatic movement occurs from frame to frame.

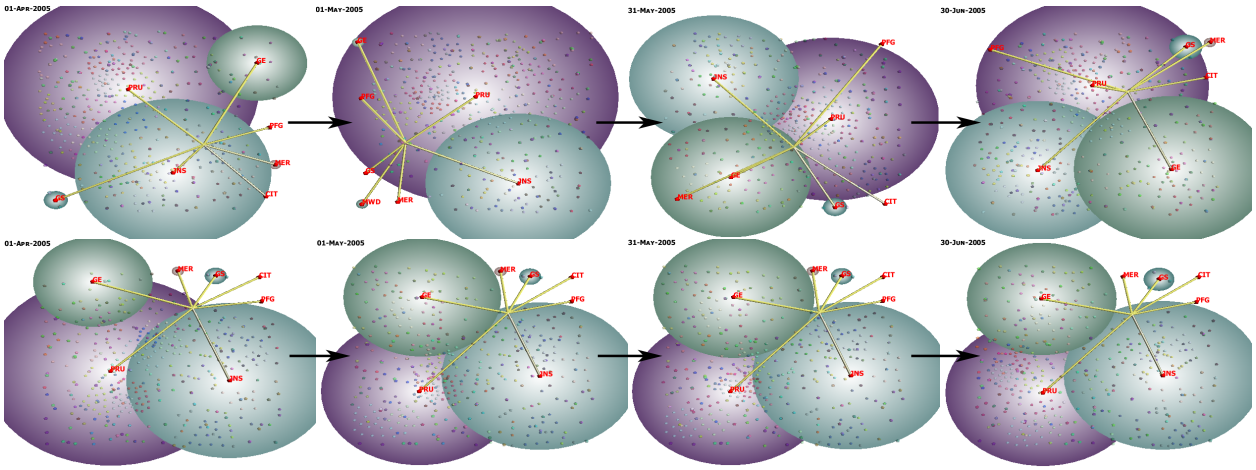

Fig. 7. Frames from animation of S&P 500 during 2005 both with (bottom) and without (top) temporal coherence.

5 Abstracted Rendering

In addition to supporting efficient layout, we also use the graph hierarchy to abstract away detail during rendering. And while most visualization tools use very simple rendering models, we are able to achieve dramatically clearer results by using modern graphics capabilities like 3D rendering, shading, and alpha-compositing.

Visualizing every edge in a large graph overwhelms the user. Most successful visualizations simplify the graph, say by contracting or filtering edges. We propose hierarchy-based filtering of edges, which users click through to explore. Level-by-level views allow users to view relationships between nodes of comparable authority while avoiding edge overload as demonstrated in Figure 15.

Color-coded circles surrounding a group of nodes represent sibling sets so that clusters become immediately evident to the user. Because the layout is well-spaced, these circles will also be distributed well and the regions of overlap are minimal. The hierarchical position of nodes within these overlapping regions is still clear through color-coding. Sibling nodes with the same parent and the circles surrounding the sibling set are colored alike to clearly segment the graph using color. The user may also color nodes by industry, making industry clusters visible. An edge is only shown if the user clicks in the sibling circle of one of the nodes of the edge. Using alpha-compositing of these circles based on level, the user also has the perception of focusing on different levels as in a microscope and may even zoom in to explore a family of nodes.

Drawing the hierarchical Voronoi diagram is also a viable alternative, although it adds considerable expense to the frame rendering time. More importantly, the Voronoi cells change much more significantly between frames than the circles we use, leading to disturbing visual "popping" artifacts during animation.

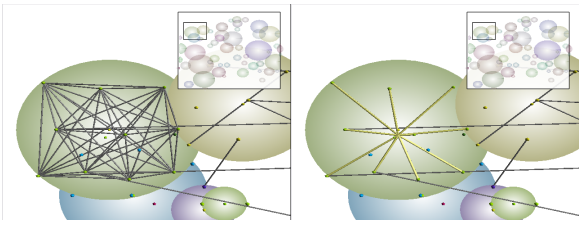

Fig. 8. Drawing cliques with simple visual symbols greatly reduces visual complexity, while zooming makes local structure apparent.

In addition to filtering, we attempt to minimize the number of edges drawn in two ways: clique simplification and forked edges. Cliques are a major source of entanglement in graph drawings. Therefore, we simplify cliques, as illustrated in Figure 8, by using star edge glyphs. This avoids clutter while still allowing users to pinpoint clusters of highly connected nodes. Inspired by confluent diagrams [5], we use forked edges that combine all edges from a single source node to several targets that are part of another sibling set. This can dramatically simplify the display, as seen in Figure 9.

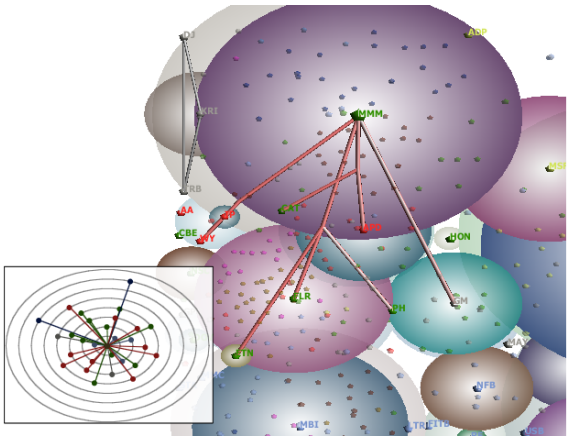

Fig. 9. Selecting a node highlights its neighbors in a subgraph. Clicking a sibling set displays edges within that set, and edges to other sibling sets are forked.

Our visualization tool also includes several interactive features to extract important aspects of the graph. As mentioned above, users may zoom in to investigate particular families in the graph tree and zoom out to examine relationships between families. A subgraph, highlighted in Figure 9 is used to visualize details of selected nodes. When users interactively select nodes, the subgraph shows all neighbors of the selected nodes organized into levels through concentric circles. When extended to dynamic graphs, an animation slider allows users to easily play the entire animation or explore particular frames in as rich a manner as a static graph.

6 Results

Using our visualization tool, we were able to quickly extract valuable information from the financial data we looked at. For example, we were able to automatically organize securities by industry and discover price relationships between industries simply by noticing the

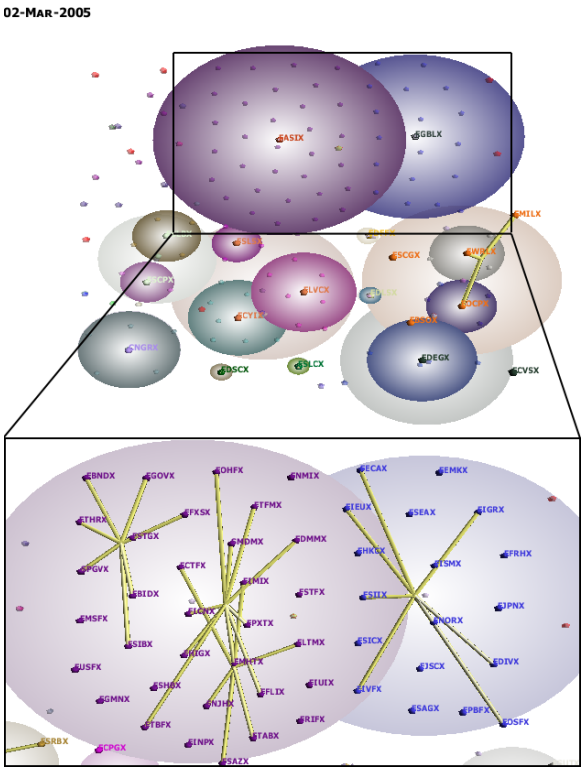

Fig. 10. Correlation graph for Fidelity fund family (2005). Subtrees for FASIX and FGBLX are highlighted.

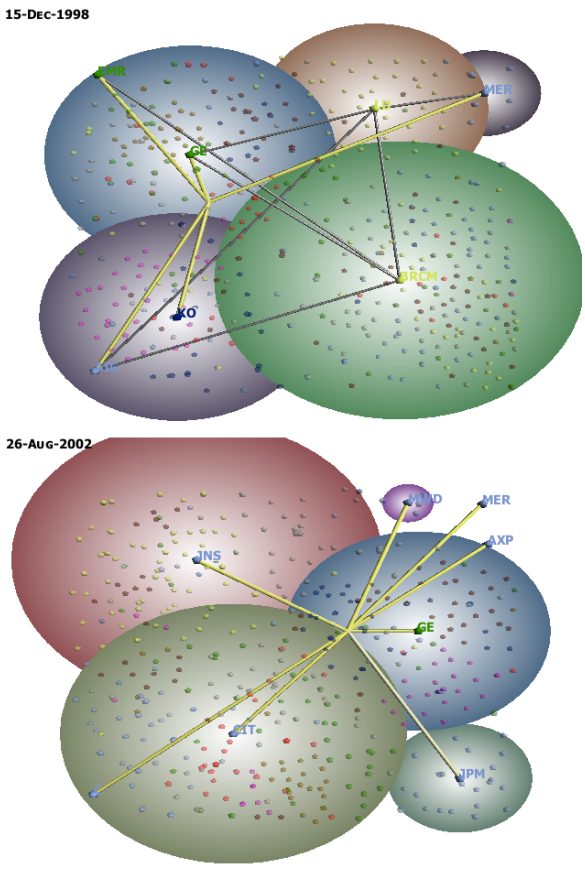

Fig. 11. The rise of financials (light blue) is seen in Dec. 1998 and Aug. 2002. Janus Capital forms the root of the IT (yellow) subtree.

coordinate placements of industries and the edges between them. For portfolio management, we can quickly determine which stocks and industries would best diversify our portfolio by looking at which areas of the graph we have few assets in. We are able to view stocks that are the most highly correlated to others and notice that these are the large-cap and financial services stocks since they are more likely to follow general market trends. We can immediately identify very highly correlated clusters of stocks using clique symbols.

We are even able to easily discover interesting changes in the graph over time, a largely overlooked feature in current visualization tools. For example, we produced a temporally-coherent animation of the S&P 500 from January 1990 to June 2005 using every month as a keyframe. We noticed dramatic movements in the graph from 1999 to 2002. This corresponded to the internet bubble, a very dynamic period in the stock market. A sustained bull market followed by a recession caused many changes in correlations. A large part of this movement was due to the rise of financial firms to the top levels of the graph, which was immediately evident in our animation. Financial services companies, who are most affected by market changes, started to gain correlation with many companies during this time, boosting them to the top level. As seen in Figure 11, in December 1998, Merrill Lynch (MER) and AIG (AIG) were the only financial nodes in the top level, but by April 2002, General Electric (GE) was the only non-financial top level node. Along with financial services, the most highly affected industry during the bubble was obviously information technology. By November 2001, IT nodes fell under the umbrella of Janus Capital Group. Janus, which had amazing growth in 1999 due to its holdings in tech companies like Amazon and Priceline, suffered a drastic melt down in the March 2000 crash. Thus, the price correlation between Janus and the IT nodes becomes evident.

We were also able to discover other interesting market events. For example, the rise and fall of Lucent Technologies (LU) is one major change seen through our animation. Lucent, a major player in the 1999 technology market, grew almost 1000% in the late 90s, however in early 2000, its stock dropped even more incredibly, greatly outpacing the average drop of the crash. In the animation, Lucent was just a leaf node in October 1997, but by November 1998, it was a top level node and soon became the parent of the IT sector, as seen in Figure 12. By June 2000, Lucent was a leaf node again. After the recession, another central node emerged: Prudential (PRU). Since late 2002, Prudential stocks saw steady growth, representative of the rest of the market. By January 2003, it was part of the top level and remained the parent of many nodes. Another interesting example is American International Group (AIG), who was a central node in the graph since 1993. However, in early 2005, AIG was charged with using reinsurance strategies to hide poor performance on the balance sheet. This cost the CEO his job and caused the stock to tumble to a two-year low. Because of this hammering, AIG, whose parent was the consistently growing Prudential, became more correlated with the stagnating GE. Figure 13 shows the graph at the end of January 2005.

In Figure 10 we see our visualization tool applied to the funds of the Fidelity mutual fund family. Among other interesting observations, we notice that the conservative allocation fund, Fidelity Asset Manager Income (FASIX), was highly correlated with the municipal and government bonds because of its similar holdings. We also note that Fidelity Global Balanced (FGBLX), which holds investments throughout the world, is the parent, and thus representative of, funds for Asian, European, and emerging market equities.

Although we have focused on financial applications, our visualization tool is general enough to produce quick layouts for any application requiring the analysis of complex and dynamic relationships. For example, we produced a visualization of a protein-protein interaction graph shown in Figure 14 (1846 nodes, 2203 edges) with a layout time of 29 seconds. We also visualized a high energy physics publication network shown in Figure 15 (4841 nodes, 24587 edges), in which our

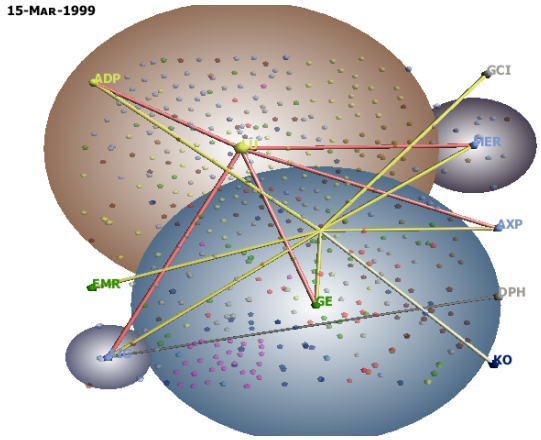

Fig. 12. At the peak of the boom in March 1999, Lucent Technologies (LU) is the parent of the IT sector.

layout algorithm finished in 31 seconds. By comparison, our S&P 500 animation converged to a final layout in a median time of 13 seconds per keyframe. All running times were computed on a 1.86 GHz Pentium M processor.

7 Conclusion and Future Work

Although we believe our visualization tool to be very useful for most data, the quality of the hierarchy created is still very data-dependent. The hierarchy imposed on the graph may not be as meaningful for graphs not matching a power law distribution of edges. Also, cases where two nodes are equally qualified to be the parent of another node can cause dramatic movements in an animation due to small changes in weight. Although we tried to minimize spurious movement by taking an offline approach to maintain temporal coherence and by incorporating several metrics in choosing parents, we realize that a standard tree hierarchy may not be appropriate for some data. Stratifying or building the hierarchy in a more flexible manner (perhaps using a tree where nodes can have more than one parent) may be promising future work. Additionally, since the current space complexity of our tool is $O(n^2)$, we believe that we can improve on this to support very large sparse graphs in the future. Finally, a different hierarchical visualization technique may also improve the system. Although we tried to limit regions in which sibling circles occlude each other, a quasi-Voronoi cell approach with good temporal coherence could address this issue.

Despite these limitations, our case studies on financial data show that our system is still very useful in analyzing graphs and extracting trends from complex data. We believe a major advantage of our method is the ability to analyze dynamic graphs, an often-overlooked aspect of graph layout research. Our method of structuring complex graph data and our interactive approach to graph layout allows us to efficiently express key data trends while still allowing an endless amount of information to be explored and analyzed.

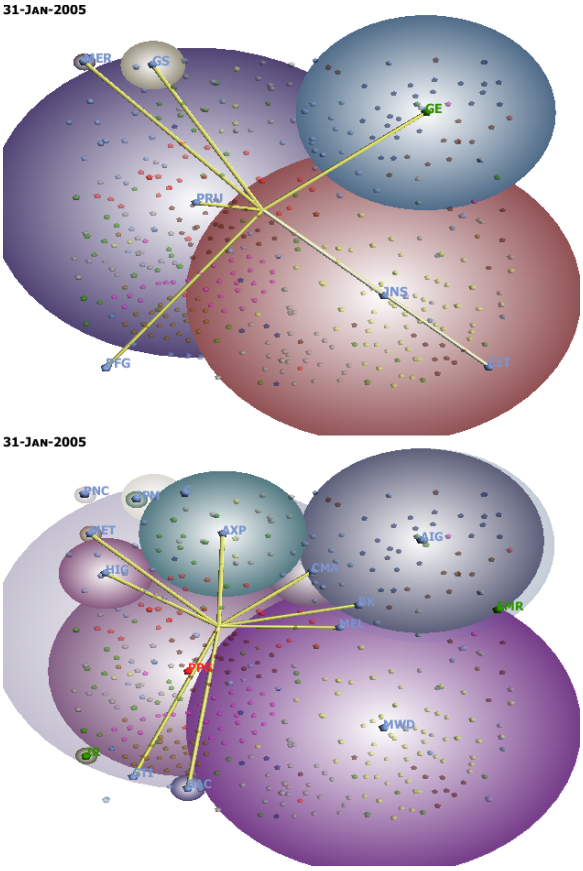

Fig. 13. The first (top) and second levels of the S&P 500 at the end of January 2005. After a significant decline, AIG becomes a child of the stagnating General Electric.

References

[1] R. Andersen, F. Chung, and L. Lu. Drawing power law graphs using a local/global decomposition, 2004.

[2] V. Boginski, S. Butenko, and P. M. Pardalos. On structural properties of the market graph. *Innovations in Financial and Economic Networks*, pages 29–45, 2003.

[3] V. Boginski, S. Butenko, and P. M. Pardalos. Statistical analysis of financial networks. *Computational Statistics and Data Analysis*, 48(2):431–443, 2005.

[4] D. S. Chan, K. S. Chua, C. Leckie, and A. Parhar. Visualisation of power-law network topologies. In *Proc. of the 11th IEEE Intl. Conf. on Networks*, pages 69–74, 2003.

[5] M. Dickerson, D. Eppstein, M. T. Goodrich, and J. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. In *Proc. 11th Int'l Symp. on Graph Drawing*, pages 1–12, 2002.

[6] D. P. Dobkin, A. Hausner, E. R. Gansner, and S. C. North. Uncluttering force-directed graph layouts. In *Proc. of the 15th Symp. on Computational Geometry*, pages 425–426, 1999.

[7] Q. Du, V. Faber, and M. Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Review*, 41(4):637–676, 1999.

[8] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[9] L. C. Freeman. Visualizing social networks. *Journal of Social Structure*, 1(1), 2000.

[10] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.

[11] E. Gansner, Y. Koren, and S. North. Topological fisheye views for visualizing large graphs. In *Proc. of the IEEE Symp. on Information Visual-*

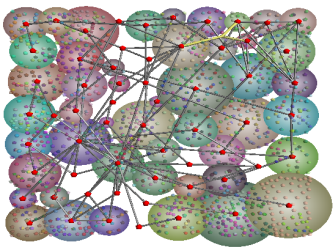

Fig. 14. Biological graph with edges between interacting proteins.

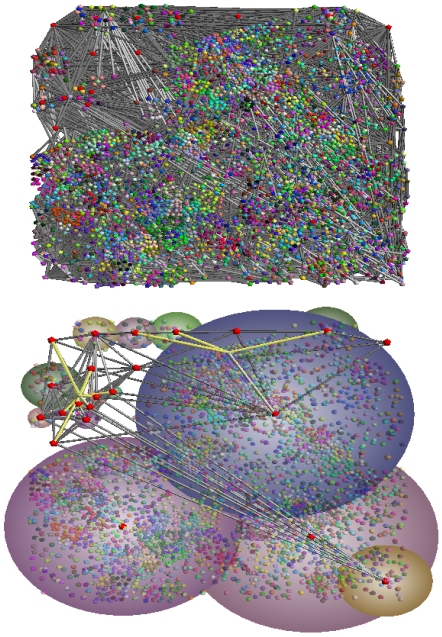

Fig. 15. Graph of physics research papers with edges representing citations. Viewed all at once (top), the graph is inscrutable whereas our filtering techniques (bottom) reveal identifiable structure.

ization, pages 175–182, 2004.

[12] E. R. Gansner and S. C. North. Improved force-directed layouts. In *Proc. 6th Int'l Symp. on Graph Drawing*, pages 364–373, 1998.

[13] C. Gorg, P. Birke, M. Pohl, and S. Diehl. Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In *Proc. 12th Int'l Symp. on Graph Drawing*, pages 228–238, 2004.

[14] S. Hachul and M. Junger. Drawing large graphs with a potential-field-based multilevel algorithm. In *Proc. of the 12th Intl. Symp. on Graph Drawing*, pages 285–295, 2004.

[15] D. Harel and Y. Koren. A fast multi-scale method for drawing large graphs. In *Proc. 8th Int'l Symp. on Graph Drawing*, pages 183–196, 2000.

[16] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[17] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[18] H. Markowitz. Portfolio selection. *J. of Finance*, 7(1):77–91, 1952.

[19] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(1):066133, 2004.

[20] S. C. North. Incremental layout in dynadag. In *Proc. of Graph Drawing '95*, volume 1027, pages 409–418, 1996.

[21] J-P Onnela, A. Chakraborti, K. Kaski, J. Kertesz, and A. Kanto. Dynamics of market correlations: Taxonomy and portfolio analysis. *Physical Review E*, 68:056110, 2003.

[22] K. J. Pulo. Recursive space decompositions in force-directed graph drawing algorithms. In *Proc. of Australian Symp. on Information Visualisation*, pages 95–102, 2001.

[23] B. Schneiderman. The eyes have it: A task by data type taxonomy for information visualization. In *Proc. for IEEE Symp. on Visual Languages*, pages 336–343, 1996.

[24] F. van Ham and J. J. van Wijk. Interactive visualization of small world graphs. In *Proc. of the IEEE Symp. on Information Visualization*, pages 199–206, 2004.

[25] A. Y. Wu, M. Garland, and J. Han. Mining scale-free networks using geodesic clustering. In *Proc. of the 10th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 719–724, 2004.