Discovering Parametric Clusters in Social Small-World Graphs

Jonathan McPherson Kwan-Liu Ma Michael Ogawa
Institute for Data Analysis and Visualization
University of California at Davis

ABSTRACT

We present a strategy for analyzing large, social small-world graphs, such as those formed by human networks. Our approach brings together ideas from a number of different research areas, including graph layout, graph clustering and partitioning, machine learning, and user interface design. It helps users explore the networks and develop insights concerning their members and structure that may be difficult or impossible to discover via traditional means, including existing graph visualization and/or statistical methods.

Categories and Subject Descriptors

I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Interaction Techniques*; H.5 [**Information Systems**]: Information Interfaces and Presentation; I.2.6 [**Artificial Intelligence**]: Learning

Keywords

Graph clustering, graph layout, histogram, information visualization, machine learning, self-organizing map, small-world graph, social networks, user interface design

1. INTRODUCTION

1.1 What is a small world graph?

The subject of small world graphs has received much recent treatment because graphs from a surprising number of disciplines have been discovered to have small world properties. When people say "it's a small world," they are expressing their surprise that they know a stranger through a short chain of acquaintances. In small world graphs, the average length of the path from one entity to another is small even though the graph is often quite large. The two main characteristic qualities of a small-world graph are small average path length and large clustering coefficient. [18]

The *clustering coefficient* c of a single node can be computed as follows. Construct a subgraph $G'(V', E')$ of the graph G that contains the node and all its neighbors. The clustering coefficient is the ratio of the number of edges that could exist in this graph to the number of edges that actually do exist in the graph. Thus:

$$c = \frac{|E'|}{(|V'| \times |V' - 1|)/2} \tag{1}$$

The clustering coefficient C of a graph $G(V, E)$ can be computed by letting $E = E'$ and $V = V'$ in the formula above. The *average path length* L of a graph is the average of the values given by an all-pairs shortest path algorithm. Now, let L_{rand} be the average path length of a random graph (a graph with randomly connected nodes), and C_{rand} be the clustering coefficient of a random graph. A "small world" graph is a graph in which the following hold true:

1. $L \leq L_{rand}$. In other words, the average path length L is at least as small, and probably smaller, than that of a random graph that has the same number of vertices and edges.
2. $C \gg C_{rand}$. In other words, the clustering coefficient C is much larger than that of a random graph.

1.2 Where are small world graphs found?

Small world graphs are found in a number of interesting—and unexpected—places. For instance, graph representations of the following have all been shown to have small world properties:

- The United States power grid, the collaboration graph for film actors, and the neural network of the nematode *C. elegans* [18]
- The World-Wide Web [3]
- Resource-sharing graphs [8]

2. THE PROBLEM

Since small world networks appear in so many areas, we would like to exploit their characteristic properties in order to answer questions about them. Some of these questions can be answered using tools that are purely mathematical, but others require a less rigid approach.

These are some of the things that we would like to be able to discover about a graph [4]:

1. What are the "social groups," or cliques, present in the graph? Small world networks tend to consist of a set of highly connected cliques or clusters, with a

SAC '05, March 13-17, 2005, Santa Fe, New Mexico, USA.

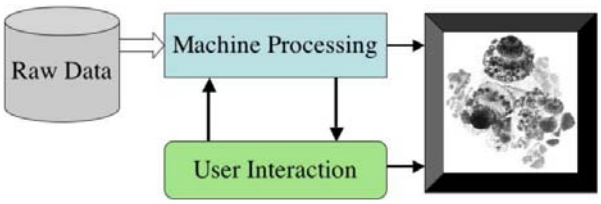

Figure 1: The basic process of visualization alternates between machine processing and human user processing. The goal is to arrive at a visualization of a segment of a small world graph which answers a specific question the user asks.

relatively small set of edges connecting them to the rest of the graph. In addition to *graph* clusters, there may also be *data* clusters of attribute or other scalar values. We use the term *parametric cluster* to refer to a cluster of data that shares some common set of properties, which may include membership in a graph clique. We will discuss this in detail later.

2. What are the properties of these groups? For each, what is its relative size? relative connectivity? topology?

3. How many of these groups are there? Do they have common properties?

4. How are these groups related to each other?

3. ABOUT THE SAMPLE DATA SET

For simplicity, the approach is first demonstrated with a simple, synthetic data set with 1,000 nodes and 10,000 edges. This data was generated by a group that is interested in the use of graphs to model relationships between entities related to national security—a node, for instance, might represent a terrorist attack, a terrorist, a meeting, and so forth. The generated data is very similar in graph structure to a real-world data set of such entities. Thus, the ability to discover insights about this synthetic data translates well to the ability to discover insights in the real-world data. The Albert-Barábasi [2] small-world graph model was used to generate this dataset.

4. OUR APPROACH

The approach is based on an iterative process that is designed to allow the raw data to be manipulated naturally to arrive at a useful view. The process alternates between machine processing and human user processing. See Figure 1 for an overview of the process, in which some of the plausible steps are discussed as follows.

4.1 Initial layout

At the beginning, the user is presented with an initial image of the graph. It is quite difficult to generate a useful image without some knowledge of what the user is after, so our goal in generating an initial layout is simply to present some view of the graph that can be easily manipulated and observed. Displaying all nodes and all edges is not useful even for a small graph like our synthetic data set (See Figure 2(a)). Instead, a spanning tree of the graph is created, using a simple algorithm that aims to create a "bushy" tree. Edges must be discarded to present any worthwhile image of the graph, since any graph of appreciable size will have more edges than can be meaningfully displayed at once. Using a spanning tree permits preservation of the structure of the graph while eliminating the vast majority of the edges; however, the main goal in generating the tree is to give us an input to the layout algorithm.

A layout algorithm much like H3 [11] is used to generate the initial layout. The main difference is that H3 lays out nodes in hyperbolic space, whereas this algorithm uses Euclidean space. The decision to use Euclidean space was made primarily to preserve the user's "mental map" of the data. See Figure 2 (b) for an illustration of the layout mechanism of hierarchical hemispheres. Every parent hemisphere is made just large enough for its children to be placed on its surface. The dimensionality of the layout (3) is important because we wish our algorithm to be able to support graphs as large as possible, and 3D has a size advantage (and other advantages) over 2D when used to visualize information nets [17]. Since at this early stage there is more concern with presenting the largest amount of data possible than with making the data easy to interact with, it makes sense to trade interaction ease for sheer quantity of data [14]. Figure 2 (c) shows the initial appearance of the graph.

4.2 Derived attributes calculations

Some useful attributes are derivative; they need not be included in the raw data because they can be inferred from it. The clustering coefficient is a good example: it can be calculated by applying the simple formula discussed earlier.

Another useful calculation is clustering. For example, the Markov clustering algorithm MCL [15] can be applied to assign a "cluster id" to each vertex. MCL is a cluster detection algorithm that can detect graph clusters, which are groups of nodes that are very strongly connected to each other. These groups represent "cliques" in social networks—groups of people that all know each other but only have a few connections with the outside world. MCL can detect clusters at multiple levels of granularity; for instance, a large cluster might consist mostly of two smaller clusters that are somewhat well-connected to each other. This kind of group would be seen as one cluster at a low granularity level and two clusters at a high granularity level.

4.3 Graph exploration

Now that the user has an initial view of the graph, he needs some way of extracting information from it. Aside from the usual zooming, panning, and rotating facilities for looking at the graph's topology, two additional mechanisms are provided for graph exploration. They are designed to work in tandem.

4.3.1 Attribute mapping

This is simply the process of mapping *data values* to *graph attributes*. The user can choose any source of data in the graph and map it to any attribute in the graph, as long as the mapping is meaningful. For instance, a node might be colored or sized according to its degree, its "cluster id", its clustering coefficient, some node attribute present in the data set, and so forth. Figure 2 (d) displays the graph mapping color to cluster id.

One of the major issues with traditional graph attribute

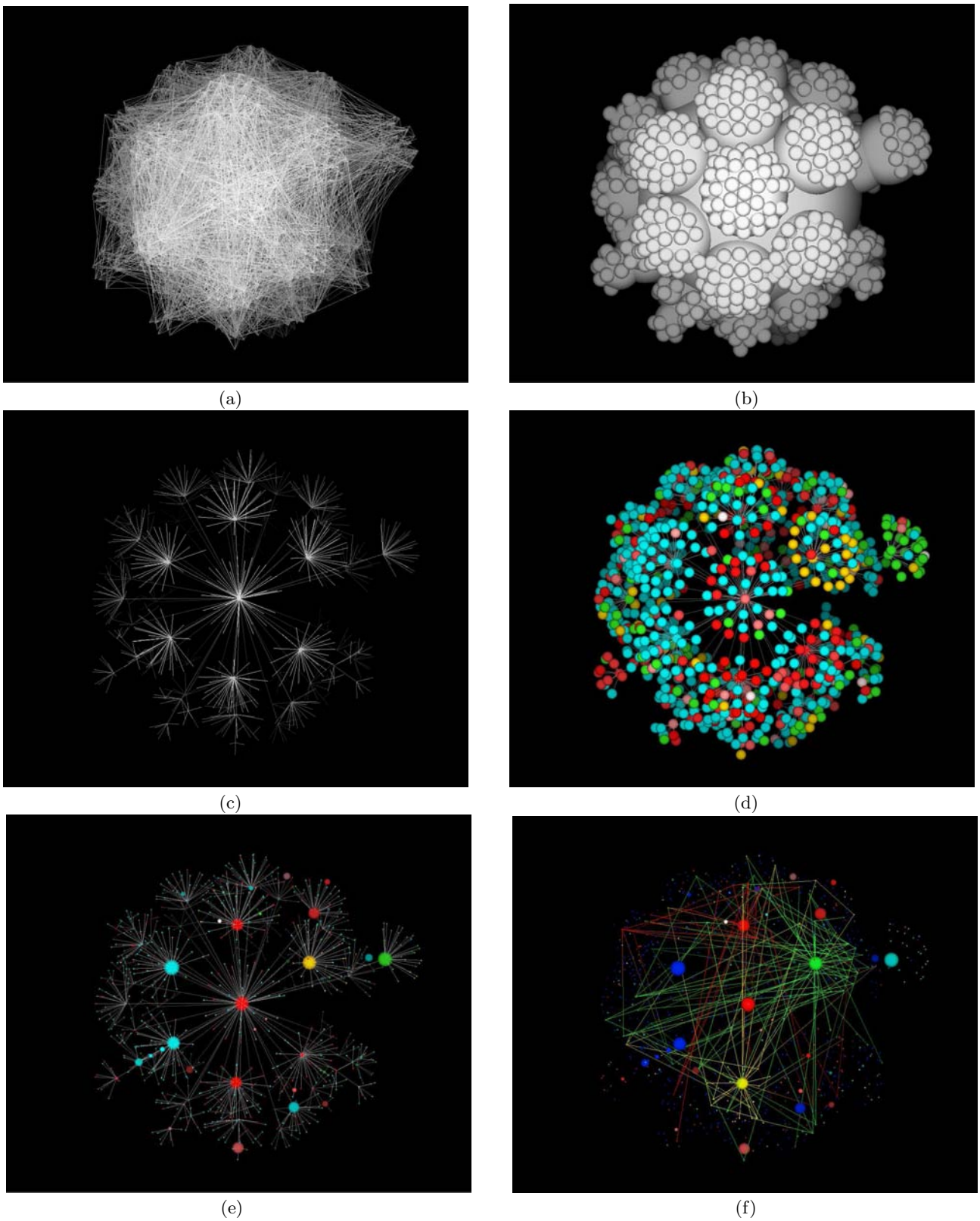

Figure 2: Visualization of the synthetic terrorism graph. (a) all nodes and all edges. (b) the layout mechanism of hierarchical hemispheres. (c) a spanning tree of the graph. (d) mapping color to cluster id. (e) mapping size to node degree. (f) subgraph selection.

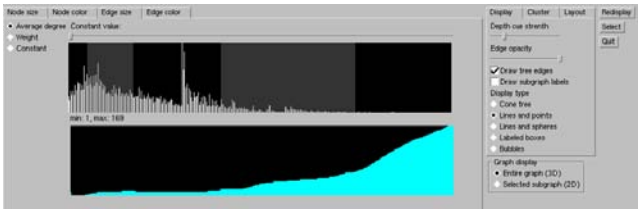

Figure 3: The user interface. Note the brushing histogram, transfer function for arbitrary size mapping, and choice of data sources.

mapping is its poor performance when presented with outlying values [6]. For instance, if a node's size represents its degree, and one node has a degree much larger than that of the others, a linear mapping of degree to size will fail to draw much distinction among the N-1 nodes in the graph.

To resolve this issue and to allow the emphasis of any parts of the data ranges the user desires, *user parameterization of all attribute mappings* is implemented. When a user selects a data source for mapping, he is presented with a histogram that helps him identify the distribution of that data. The next step depends on the type of graph attribute being mapped to:

- If the user is mapping a data source to a *color*, he is presented with a gradient editor that corresponds to and has the same width as the histogram. The gradient editor supports an arbitrary number of arbitrarily colored control points, so that the user can create an arbitrarily complex mapping from the data values to colors. By displaying the gradient editor in combination with the histogram, we encourage the user to explore various "peaks" and "valleys" in the data frequencies by mapping them to smoothly interpolated color values.

- If the user is mapping a data source to a *size*, he is presented with a transfer function editor. This transfer function, like the gradient editor, permits any level of size to be mapped to any range of data.

Figure 2 (e) contains an image of the synthetic graph with attribute mappings. We selected a color mapping that maps red, blue, green and yellow to different graph clusters, as detected by MCL. A *size map* was also created for nodes that corresponds to node degree. All of these mappings were defined and applied in real time. Figure 3 contains a screen capture of the user interface used to make the mappings and to select subgraphs. It is also helpful to create a size map for nodes that corresponds to clustering coefficient or for edges that corresponds to the average degree of the nodes they connect.

4.3.2 Subgraph selection

It is very unlikely that the user will be able to gather many useful insights from the initial layout of the graph, for at least two reasons: (1) the initial layout is not optimized toward any particular visualization goal—and cannot be, since there is no initial knowledge of what the user is looking for—and (2) the insight the user is interested in is likely to require only certain parts of the graph.

The mechanism we use for subgraph selection is a *painting histogram*. The same histogram used for attribute mapping is used for subgraph selection. It allows the user to "click and drag" to select a range of data values. The user can select multiple ranges in the same histogram. Contrast this with simpler interfaces, e.g. the single-range selection mechanisms in [1]. We believe that multiple-range techniques are necessary for the achievement of some visualization goals; for instance, imagine that the user wanted to answer the question "how are the vertices of high degree and vertices of low degree related in the graph?" This question can be answered only by selecting *both* the vertices of high and low degree in the graph. The subgraph can often be thought of as a skeleton of the graph built according to a user-controlled metric, as briefly described in [7].

The use of the histogram is important because it strongly ties the subgraph selection process to the attribute mapping process. The idea is that the first will influence the second; if the user maps some part of a data source to a red color, for instance, it becomes quite easy to select the parts of the graph that are red.

Note that we include subgraph selection in the same step as attribute mapping. This is because these processes are not necessarily sequential. The first try at subgraph selection may lead to a realization that the attribute mappings previously chosen are not useful for discerning appropriate parts of the graph for selection, or vice versa. The process may need to be repeated iteratively until the user is pleased with the subgraph he has selected and the attribute mappings he has chosen.

When a subgraph is selected, all the edges in that subgraph are displayed, rather than just the spanning-tree edges, as shown in Figure 2 (f). We selected only the nodes actually in the red, green, and yellow clusters to be part of the subgraph. All their edges are shown. The edge confusion will be resolved by the next stage of visualization, but some structure is already apparent here. The tree edges may optionally be turned off in order to get a clearer picture of what the subgraph looks like. When the user is certain that he has the right subgraph, he can move toward a more useful layout.

Note that subgraph selection can also be done based on topological structure, such as proximity [12] and edge strength [4]. However, this is not discussed in this paper due to the space constraint.

4.4 Automatic self-organizing layout

There are many algorithms for discovering cliques, and of course finding "data cliques" of nodes with similar values for some attribute is a trivially easy task. But what we are after is the discovery of higher-level cliques. For example, imagine that we are looking at the United States social network and would like our algorithm to find cliques of "people who know each other well and are paid about the same amount." It is easy to find groups of people who are paid the same amount; it is much harder but still possible to find groups of people who know each other well. No existing techniques can directly and automatically discover clusters whose components have both data and graph nearness.

Our system allows the user to conveniently perform clustering of nodes that possess a combination of *data nearness*; that is, values that are nearer to each other than to other nodes in the graph, and *graph nearness*; that is, locations

in the graph such that the length of the path from one node to another is lower than to other nodes in the graph. This is achieved by using the cluster id numbers discussed earlier. By using the cluster ids, we move the domain of graph nearness into the domain of data nearness.

4.4.1 Selection of the feature vector

A feature vector represents each node. Selection of attributes for the feature vector must be left to the user, because the selection of any attribute causes it to be used to potentially split clusters; therefore, only attributes the user is interested in ought to be used in the feature vector.

4.4.2 Kohonen maps

A technique called a Kohonen map [9] is used to generate a layout that automatically finds these kinds of clusters [10]. Kohonen maps—also known as self-organizing maps, or *SOMs*—organize high-dimensional data onto a 2-dimensional grid while preserving the data clusters in higher dimensions. In a placement of nodes generated by a SOM, nearness implies similarity, but distance does not imply difference. Suppose that each node has a d-dimensional feature vector (normalized so that each dimension is in the range $[0, 1]$). The data is organized by the SOM as follows:

1. A grid of hexagonally connected points is created. Each point on the grid is given a random d-dimensional vector.

2. An input vector i is compared with each vector g on the grid. The point on the grid with the lowest Euclidean distance $|g - i|$ is chosen.

3. The chosen point, and all the points that are near it on the grid (according to the current neighborhood size, n) have their vectors made more like that of the input vector; the amount of adjustment is controlled by a learning parameter, α. To speed computation, we use a bubble function rather than the traditional Gaussian function to relate the level of adjustment to the distance from the center of the neighborhood.

4. The previous two steps are repeated for each input vector. n and α are linearly reduced as the map approaches its final form.

5. The input vectors are presented once more to the SOM for final placement.

Smaller grids tend to generalize data better than larger grids, whereas larger grids tend to generate better detection of fine features. The user interface permits small adjustments to the grid size, and to the learning parameter α.

The main differences between traditional SOM applications and this application are (1) we are applying the layout to a graph rather than a set of independent data points, and (2) by mapping graph clusters found by MCL to scalar values, we are permitting the use of an algorithm designed to discover clusters of scalar values to "discover" the encoded graph clusters as well. Figure 4 shows an example clustering of the synthetic terrorism data set.

4.5 User-controlled force-directed layout

At this point, we have a layout that has been generated by the Kohonen map. But:

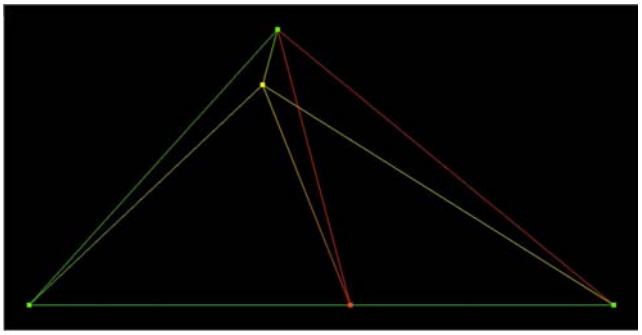

Figure 4: Automatic clustering. The self-organizing map has put each node into one of five "parametric clusters" partitioned by graph nearness and node degree. We selected a feature vector consisting of node degree and "cluster id." The few nodes represented in this clustering are actually groups of nodes that have been partitioned according to differences in their cluster ids and degree values.

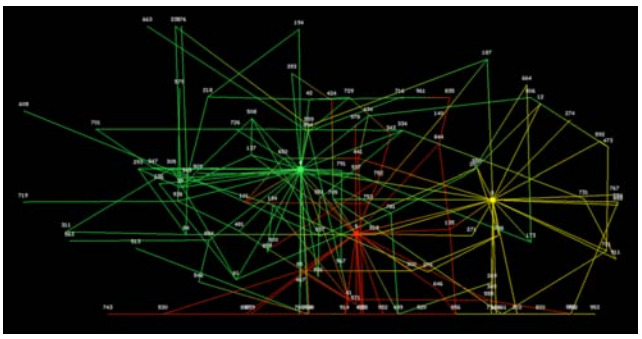

Figure 5: Parametric clusters have been pushed apart by the force-directed layout algorithm after several iterations.

1. while it may be a fine layout of data *points*, it is not a terribly good layout for a *graph*; and

2. the self-organizing map may place two or more points at the same location, making it impossible to see how many data points are actually present and what the relationship among them is. (Figure 4 illustrates this.)

Traditional graph layout algorithms are not appropriate for use here, for they will disturb the clustering that the Kohonen map created in their attempt to create an evenly spaced layout. Nor can the user be forced to manually move things about until he is happy with the result—this is awkward and likely too time-consuming a task.

To resolve the issue, a hybrid algorithm is used: a *user-controlled* force-directed layout. The force-directed algorithm is based heavily on the one described by Fruchterman in [5]. At any point, the user can do either of the following:

1. *perform small number of iterations of a force-directed algorithm with limited displacement.* This has the effect of pushing apart nodes that are placed close together and attracting groups of nodes that have a lot of edges between them. It moves the system closer to a state of even distribution of nodes in the visualization

space. Force-directed layouts of grouped nodes are an established practice for effective visualization [13].

2. *move a group of nodes to anywhere in the visualization space.* The user can select one or more nodes and drag them to a new space. Dragging nodes outside the space merely enlarges the space and re-scales it, so a group of nodes may be dragged "outside" the space. Moving nodes about also provides an important pre-attentive cue: motion. By selecting a group of nodes and moving them around a bit in small, quick increments, the relationship between the nodes and anything else in the graph moves and therefore "pops out."

After some number of applications of the above two steps, the user will likely have reached a final visualization of the graph that contains the insight he was seeking (or, even better, an insight he was not seeking).

Observe the final view of the graph achieved in Figure 5. This figure was generated by clustering both on "cluster id" and on node degree. We can make some interesting observations from it. First, Each of the three clusters (red, yellow, and green) seems to have a "master node" that is of much higher degree than the others, and is connected to the other "master nodes". Second, the yellow cluster talks to the red cluster almost entirely through its "master node"; there are virtually no connections between the yellow and red groups.

5. CASE STUDY: KAZAA FILE-SHARING GRAPH

This data set consists of a graph of activity on the Kazaa file-sharing network [8], containing about 2,400 nodes (users) and 13,300 edges (connections between users). Like many socially-based networks, it exhibits small-world properties. Figure 6 shows some of the visualization results. We found that there are several tightly connected cliques, and that big neighborhoods have the similar clique structure as smaller neighborhoods. .

The images in figures 7 and 8 represent a potential user's exploration of the dataset. While it could conceivably be possible to get answers to many of the user's questions using other techniques, the key here is that the user is able to quickly adjust the visualization to find answers. It would take programming, statistics collection, and analysis by a skilled user to answer these questions mathematically; here, an unskilled user can answer quickly and with minimal knowledge about the data.

6. COMPUTATIONAL COMPLEXITY

The self-organizing and force-directed layouts are the two most computationally intensive tasks that are performed while the user is interacting with the system. For a graph of size n and a self-organizing map of size m (the size of the map is user-controlled), these take time $O(n^2)$ and time $O(nm)$, respectively. In practice, these run on modern hardware (a 2GHz Pentium) in under a second on subgraphs of 500 nodes.

When the graph description is first loaded, by far the longest computational cost is incurred in the MCL algorithm. See [15] for a detailed discussion of MCL's computational complexity. It is about $O(kn^2)$, where k is a parameter to MCL that specifies the number of nonzero entries per stochastic sparse matrix column (typically 500–1000). In practice, MCL takes about an hour to cluster a 30,000 node graph, but under a minute to cluster a 3,000 node graph.

7. CONCLUSION

To design our approach, we first considered the sub-tasks involved, and experimented with some of the existing techniques to solve each, adapting or modifying those techniques to suit the design goals. The resulting set of algorithms and methods seems complex, but it reflects what we believe will be an increasing trend in the design of information visualization systems: the collaborative effort of many pieces of already-discovered methods into something greater than the sum of its parts. The information visualization field is young, but already there exists a rich selection of methods that accomplish specific tasks. The study of how to apply this explosion of discovery in real-world systems and situations deserves more treatment than it has been given.

We have described a cohesive, step-by-step process for exploring small-world graphs, along the way we have presented several novel concepts, including specific methods of subgraph selection and attribute mapping; a unification of data and graph closeness; and a simple layout method that takes advantage of neural networks, physically-inspired automatic layouts, and user interaction. Our study forms a good basis for further study in this area.

8. FUTURE WORK

We will continue refining our system design by incorporating more effective clustering algorithms [12] and interactive focus+context techniques for visualizing small-world graphs [16]. Other specific directions for future work are summarized as follows.

8.1 Better graph clustering algorithm

The method of generating identical values for every member of a cluster as located by a single run of MCL at a pre-set granularity level could use improvement. The issues associated with this method include the following:

1. clusters can only be detected at a single, pre-set level of granularity;
2. the nearness in the "cluster ids" created has nothing to do with their graph nearness; and
3. the clustering can take quite some time for a large graph, even on modern hardware.

To solve these problems, we would like to find a fast algorithm to generate some vector D_k for each node k, such that the following relationship holds:

$$\forall i, j \in V : |D_j - D_i| \propto C(i, j) \tag{2}$$

where $C(i, j)$ returns the "graph nearness" of i and j; that is, the length of the shortest path from i to j. We want this property so that we can use the D_k values in our self-organizing map to allow it to automatically find graph clusters. It seems unlikely that a scalar value for D_k could be found for every node in the graph to satisfy the above condition, but what about a two-dimensional (or three-dimensional) vector value? If an N-dimensional ($N \ll |V|$) vector can represent D_k values that meet the above inequality, how can we find N for some $G = (V, E)$ and derive all the D_k values in a reasonable time?

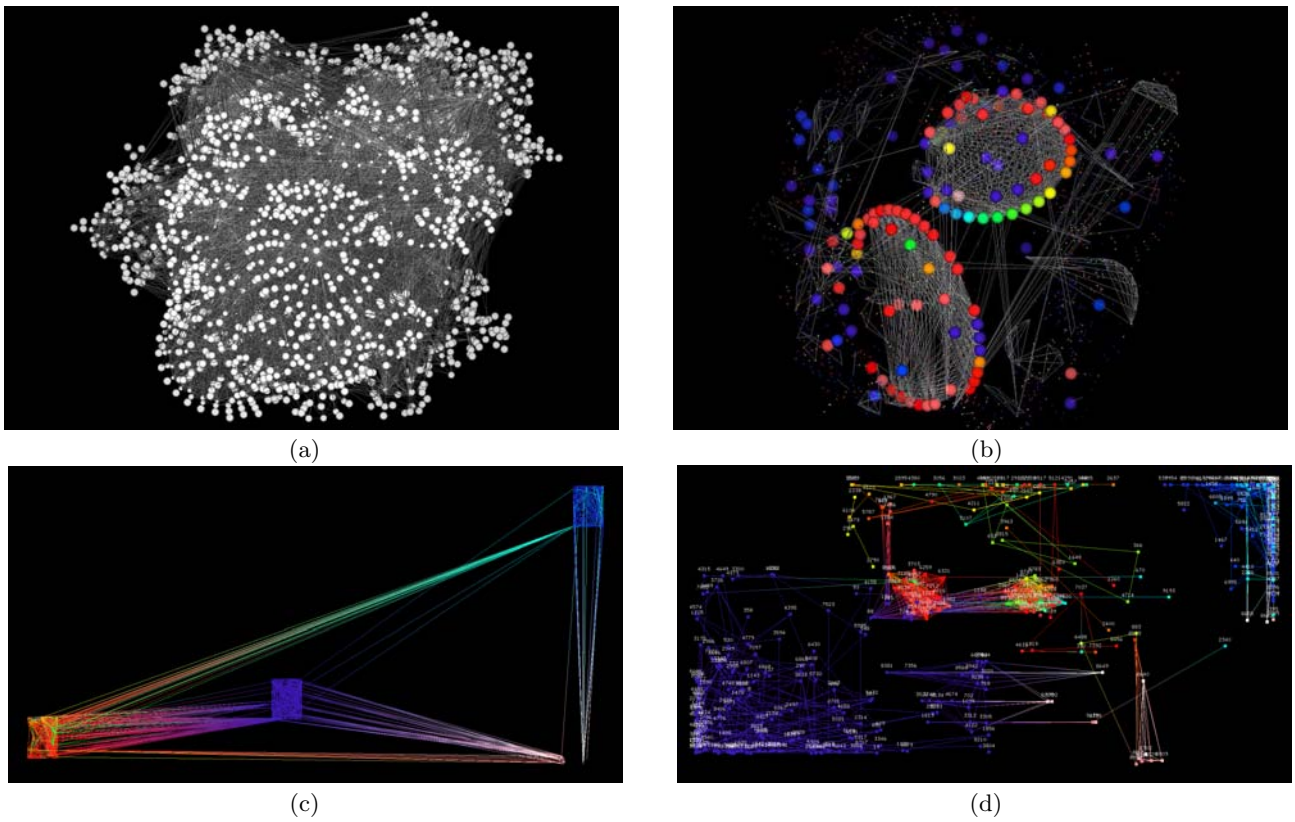

Figure 6: Visualization of the Kazza dataset. (a) displaying all nodes and all edges with uniform size and color. (b) each node has been assigned a color based on its cluster id and a size based on its degree. Only edges connecting nodes with high clustering coefficients are displayed. These are tight neighborhoods of file-sharers. (c) result of applying a SOM and one iteration of the force-directed layout. (d) after more iterations of the force-directed layout.

8.2 SOM improvements

8.2.1 Better SOM implementation

The current SOM implementation has two shortcomings, both of which were intentional sacrifices of quality for speed: (1) it only goes through the input data once, which means that the earlier input vectors get an unfair emphasis as N and α decline, and (2) it uses the "bubble function" rather than a Gaussian to regulate the strength of vector adjustment as distance from the epicenter increases. We should be able to remove these shortcomings without significantly impacting the interactivity of the system.

8.2.2 Weighted feature-vector components

Currently, when SOM is clustering the data, it assigns an equal weight to each component of the feature vector; that is, if we include "node degree" and "cluster id" in our feature vector, a group is just as likely to be split over either. We would like to find weights $w_1, w_2, \ldots, w_n$ for each of the n dimensions of the feature vector, such that the weight w_i represents the importance of separating groups based on attribute i.

9. ACKNOWLEDGMENTS

This work has been sponsored in part by the U.S. National Science Foundation under contracts ACI 9983641 (PECASE award), ACI 0325934 (ITR), ACI 0222991, and CMS-9980063; and Department of Energy under Memorandum Agreements No. DE-FC02-01ER41202 (SciDAC), No. B523578 (ASCI VIEWS), and No. B537770. The authors would like to thank Jens Schneider for his helpful comments. The authors would like to thank Edmond Chow at Lawrence Livermore National Laboratory for generating the sample data set, Adriana Iamnitchi at the University of Chicago for her helpful suggestions and for providing the Kazaa data set, and Stijn van Dongen for the use of his MCL implementation.

10. REFERENCES

[1] Ahlberg, C., and Shneiderman, B. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Human Factors in Computing Systems. Conference Proceedings CHI'94* (1994), pp. 313–317.

[2] Albert, R., and Barabási, A.-L. Statistical mechanics of complex networks. *Rev. Mod. Phys. 74* (2002), 47–97.

[3] Albert, R., Jeong, H., and Barábasi, A.-L. Diameter of the world wide web. *Nature 401* (1999), 130–131.

[4] Auber, D., Chiricota, Y., Jourdan, F., and Melancon, G. Multiscale visualization of small world

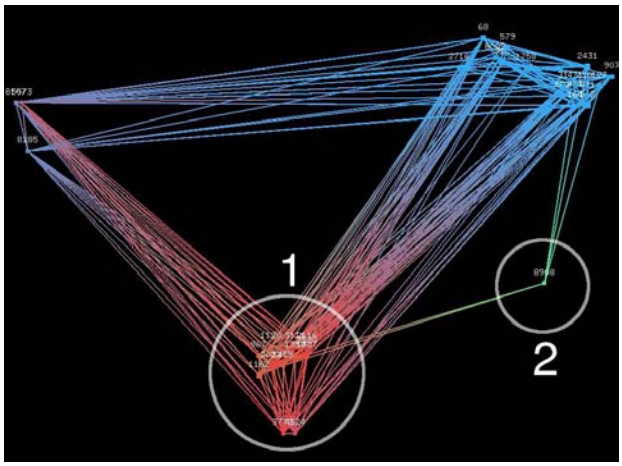

Figure 7: Vertices of high degree were selected. Large clusters are blue, average clusters are red, and small clusters are green. The question is: "How are users who are very well-connected to others distributed in the neighborhoods?" The visualization makes it clear that they are roughly equally distributed. Note, however: (1) a neighborhood size that tends to attract a disproportionate number of well-connected users, and (2) one user who has a very large number of connections for someone from such a small neighborhood.

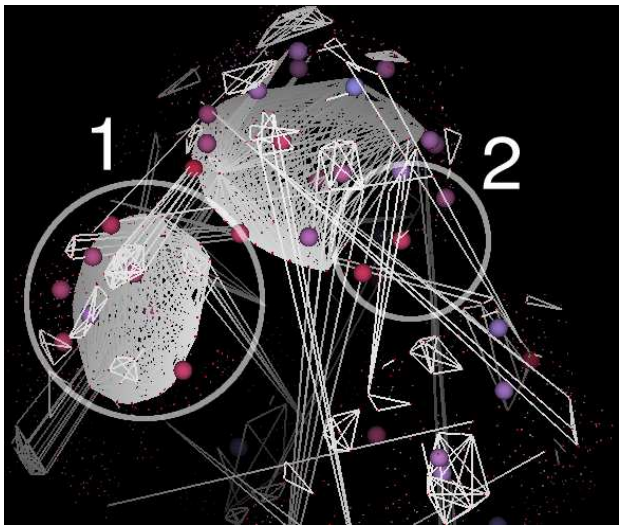

Figure 8: The question is: "Do users who have a lot of connections tend to be part of tightly connected neighborhoods?" The answer was found by applying a size map to the nodes, so users with lots of connections showed up as large spheres—e.g. the trio in (2)—and selecting a subgraph consisting of nodes with very high clustering coefficient so that tightly connected neighborhoods—e.g. (1)—were shown. It is clear that the highly connected users are rarely part of the tightest-knit neighborhoods, since the spheres and clusters do not coincide.

networks. In *Proceedings of the 2003 IEEE Symposium on Information Visualization* (2003), pp. 75–81.

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

[6] Herman, I., Marshall, M. S., and Melancon, G. Density functions for visual attributes and effective partitioning in graph visualization. In *INFOVIS* (2000), pp. 49–56.

[7] Herman, I., Marshall, M. S., Melancon, G., Duke, D. J., Delest, M., and Domenger, J.-P. Skeletal images as visual cues in graph visualization. In *Data Visualization '99*, E. Gröller, H. Löffelmann, and W. Ribarsky, Eds. Springer-Verlag Wien, 1999, pp. 13–22.

[8] Ianmitchi, A. *Resource Discovery in Large Resource-Sharing Environments*. PhD thesis, The University of Chicago, 2003.

[9] Kohonen, T. *Self-Organization and Associative Memory*, 3rd ed. Springer-Verlag, Berlin, 1989.

[10] Meyer, B. Self-organizing graphs: A neural network perspective of graph layout. In *Proceedings of the 6th International Symposium on Graph Drawing* (1998), pp. 246–262.

[11] Munzer, T. Exploring large graphs in 3d hyperbolic space. In *IEEE Computer Graphics and Applications, Vol. 18, No. 4* (1998), pp. 18–23.

[12] Noack, A. An energy model for visual graph clustering. In *Proceedings of the 11th International Symposium on Graph Drawing* (2003), pp. 425–436.

[13] Six, J. M., and Tollis, I. G. Effective graph visualization via node grouping. *Proceedings of the 2001 IEEE Symposium on Information Visualization* (2001), 51–59.

[14] Stasko, J. T., and Wehrli, J. F. Three-dimensional computation visualization. In *Proc. IEEE Symp. Visual Languages, VL* (24–27 1993), E. P. Glinert and K. A. Olsen, Eds., IEEE Computer Society, pp. 100–107.

[15] van Dongen, S. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.

[16] van Ham, F., and van Wijk, J. J. Interactive visualization of samll world graphs. In *Proceedings of the 2004 IEEE Symposium on Information Visualization* (2004), pp. 199–206.

[17] Ware, C., and Franck, G. Evaluating stereo and motion cues for visualizing information nets in three dimensions. *ACM Transactions on Graphics 15*, 2 (1996), 121–140.

[18] Watts, D., and Strogatz, S. Collective dynamics of small-world networks. *Nature 393* (1998), 440–442.