Interactive Visual Summary of Major Communities in a Large Network

Yanhong Wu* Wenbin Wu† Sixiao Yang‡ Youliang Yan§ Huamin Qu¶

Hong Kong University of Science Technology
Huawei Technologies Co., Ltd.

ABSTRACT

In this paper, we introduce a novel visualization method which allows people to explore, compare and refine the major communities in a large network. We first detect major communities in a network using data mining and community analysis methods. Then, the statistics attributes of each community, the relational strength between communities, and the boundary nodes connecting those communities are computed and stored. We propose a novel method based on Voronoi treemap to encode each community with a polygon and the relative positions of polygons encode their relational strengths. Different community attributes can be encoded by polygon shapes, sizes and colors. A corner-cutting method is further introduced to adjust the smoothness of polygons based on certain community attribute. To accommodate the boundary nodes, the gaps between the polygons are widened by a polygon-shrinking algorithm such that the boundary nodes can be conveniently embedded into the newly created spaces. The method is very efficient, enabling users to test different community detection algorithms, fine tune the results, and explore the fuzzy relations between communities interactively. The case studies with two real data sets demonstrate that our approach can provide a visual summary of major communities in a large network, and help people better understand the characteristics of each community and inspect various relational patterns between communities.

1 INTRODUCTION

Relationships in complex systems are often represented as graphs and graph visualization enables people to intuitively observe graph structures and is thus widely used. Real-world graphs often have the community structures property, in which nodes are joined into tightly knit groups. Analyzing community structures is of great importance for people's understanding of graph structures. Previous researchers have proposed a variety of methods to detect and evaluate such structures [14, 27].

However, for large graphs, common visualization methods will encounter the scalability problem. For example, the node-link diagram may produce hairball-like results which obscures the community structures, preventing people from gaining insight in large graphs. As community structures widely exist in real world graphs, some visualization systems have been designed to reveal them [32, 33, 34]. However, existing visualization solutions usually target community attributes or hierarchical clustering structures while several issues still remain. First, the communities are often detected using data mining and community analysis algorithms. Different algorithms or different parameter settings will lead to different results. Users often want to know the quality of the communities generated by different methods. For overlapping or fuzzy clusters, there exist boundary nodes which cannot easily be put into any clusters. Besides, relation patterns among communities differ in a variety of ways. For example, one community may connect to others via many low-degree boundary nodes while another community may connect to others only through boundary nodes of a rather high degree. Therefore, developing an effective visualization method to show the major communities in a large network, their qualities, and the boundary nodes connecting them will be very desirable.

In this paper, we present a novel visualization approach to address these issues. We first use the Voronoi treemap [6], an effective space-filling technique that can support hierarchical data structures, to generate community boundaries. We then enlarge the gaps between polygons in order to create space for boundary nodes. Boundary nodes will be positioned in the gaps according to their relations with major communities. Finally, we provide the detailed view, statistic view and node-link view to reveal more detailed connection properties. Our method can provide an overview of a large network and allow users to evaluate, compare and refine the community structures in the network.

The major contributions of the work presented in this paper are:

- A graph layout based on MDS and the Voronoi treemap to position polygons of different areas representing major community structures in a large graph.
- An intuitive visualization scheme using various attributes of polygons such as smoothness to encode attributes of communities.
- A visualization scheme for boundary nodes.
- Rich user interactions that enable users to merge, split, and refine communities.

2 RELATED WORKS

Graph visualization has a rather long history [20]. Two main graph visualization methods are adjacency matrices and node-link diagrams. Adjacency matrices methods [19, 12] cannot intuitively reveal community structures. They also consume more space for large graphs and are not well suited for path-related work [18]. Node-link diagrams are often generated with force-directed layout algorithms, such as the Kamada-Kawai model [23] and the Fruchterman-Reingold model [15]. However, for large graphs, the results might become visually cluttered and the community structures cannot be easily revealed. Techniques like node coloring are widely used to highlight the community information in node-link diagrams [22]. Bubble Sets [10] uses a contiguous contours to encode the nodes from the same community. Riche and Dwyer [31] proposed a compact rectangle shape to draw each community. LineSets [3] and Kelp Diagram [11] draw continuous curves instead of the contours in BubbleSets. KelpFusion [28] provides a hybrid solution bridging the hull techniques like Bubble Sets and line-based techniques such as LineSets. These methods are good at illustrating overlapping communities but the quality of the communities as well as the boundary nodes cannot be easily revealed. In addition,

*e-mail: ywubk@ust.hk
†e-mail:wwbcarol@gmail.com
‡e-mail:yangsixiao@huawei.com
§e-mail:yanyouliang@huawei.com
¶e-mail:huamin@cse.ust.hk

they face a color overlapping problem when the number of communities increases. In comparison, our method provides an easy way to estimate the community size and quality and avoids the color overlapping problem by assigning the boundary nodes different locations rather than multiple colors.

Many optimization techniques are introduced to reduce the visual clutter problem in large graphs. One widely used scheme is node aggregation. By using one abstract node to visually represent a cluster of nodes, the resulting node-link diagram is able to work for massive graphs. Muelder and Ma combined it with a treemap to generate a hierarchical graph layout that avoids visual clutter [29]. ASK-GraphView [2] is a node-link based visualization system that allows users to hierarchically explore large graphs. Archambault et al. proposed an interactive visualization system which allows users to explore and modify graph hierarchies in multiple possible ways [4]. Despite their flexibility in graph exploration, the relations between clusters are omitted. In contrast, our system is designed to focus on the cluster relations and the nodes which connect clusters. Vehlow et al. used the abstract node shape to encode the community fuzziness [34]. In this paper, we use a similar idea but the polygonal shape used in our system is quite different from the star shape used in their work in order to utilize space more efficiently.

Some works use glyphs or metaphors to convey clusters in a network. PIWI [36] uses vertex plots to describe node neighboring relations between communities but does not provide an intuitive topology overview for the whole graph. GMap [16] and its variants [17, 21] use a map metaphor and represents each community as a country, each node as a city, and each relationship as a road. Their approaches can intuitively show different clusters and their relationships. Our work focuses on conveying the community quality and the boundary nodes connecting communities.

3 Data Model and Design Tasks

In this section, we define the graph data model, introduce some terms used later, and present some design goals for our system.

3.1 Data Model and Term Definition

In this paper, we consider undirected graph model as $G = (V,E)$ with $n = |V|$ nodes and $m = |E|$ edges where $E \subseteq V \times V$, $(u,v) \in E \iff (v,u) \in E$. We model the community structures as a set of communities $\{C_1,...,C_K\}$ where K indicates the number of communities for $C_k \subseteq V$, $1 \leq k \leq K$. For each node $v \in V$, we define its community as $f_C(v) = c$ where $v \in c, c \in \{C_1,...,C_K\}$.

We then model the clustered graph as a weighted undirected graph $G^C = (V^C, E^C)$. The clustered graph G^C directly corresponds to an original undirected graph G. Each node $v^C \in V^C$ represents one cluster $c \in \{C_1,...,C_K\}$ and $|V^C| = K$. If there exists an edge between node v_i^C and v_j^C, we use $w_{e_{ij}}$ to indicate the relation strength between community C_i and C_j.

To simplify the presentation, we use the following terms in the rest of our paper:

- *Internal Degree*[14]: For node $v \in V$, its *internal degree* $d_I(v) = |\{(u,v) : u \in f_C(v), (u,v) \in E\}|$.
- *External Degree*[14]: For node $v \in V$, its *external degree* $d_E(v) = |\{(u,v) : u \notin f_C(v), (u,v) \in E\}|$.
- *Specific External Degree*: For node $v \in V$, its *specific external degree towards community* C_k $d_{E_k}(v) = |\{(u,v) : u \in C_k, (u,v) \in E\}|$
- *Internal Node*: An internal node is $v \in V$, where $d_E(v) = 0$.
- *Boundary Node*: A boundary node is $v \in V$, where $d_E(v) > 0$.
- *Internal Edge Number*: The number of internal edges in community C_k is $m_c = |\{(u,v) : u \in C_k, v \in C_k\}|$.
- *External Edge Number*: The number of external edges for community C_k is $e_c = |\{(u,v) : u \in C_k, v \notin C_k\}|$.
- *Cluster Polygon*: A Voronoi cell which visually represents a cluster.
- *Cluster Gap*: The gap between two *cluster polygons* where *boundary nodes* are positioned.

3.2 Design Tasks

We want to achieve the following design goals for our community visualization system.

T1. **Communities should be easily recognized and compared.** An effective community visualization method should clearly reveal major community clusters without visual clutter. In addition, cluster properties like cluster size and cluster quality need distinct representations in order to compare different communities. We accomplish this goal by grouping clustered nodes into several Voronoi polygons. Polygon shape, color and size are used to represent different cluster properties.

T2. **The relation strength between communities should be presented.** Evaluating the relation strength between communities is another important task for community analysis. Communities with stronger relations are more likely to generate new connections or to form a new united community. Having an overview of community relation strength will help people better understand interactions between communities. We put highly related communities geometrically closer to fulfill this purpose.

T3. **Boundary nodes between communities should be illustrated.** Apart from getting the overview of clusters statistical information, a visual representation of boundary nodes between communities is necessary for many applications [13, 24]. Giving a clear visual depiction of boundary nodes will help both evaluate these cluster measurements and understand how communities are connected. Boundary nodes are emphasized in our system by positioning them inside cluster gaps to achieve this goal.

T4. **Hierarchical structures and internal nodes.** A network sometimes has a hierarchical structure in which smaller communities can form a larger community. Revealing such substructures allows people to better understand a large network at different levels of detail. We adopt a treemap layout to support this requirement.

T5. **Interactions should be supported to enable dynamic community exploration and refinement.** Users should be able to dynamically merge clusters, rearrange the global layout, probe highly connected substructures, and highlight nodes of interest. In addition, a smooth transition during the changes of views should be provided to keep users' mental map. We support rich user interactions to satisfy these requirements.

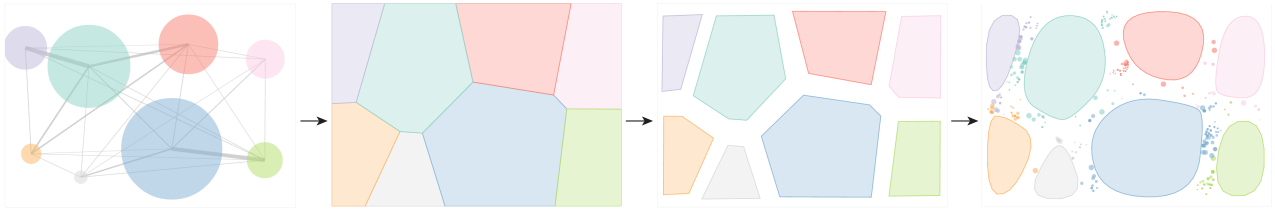
4 Visual Design

In this section, we first describe our visual encodings schemes, some alternative design choices we considered, and the interactions supported in our system. The layout generation process is summarized in Fig.1.

4.1 Cluster Polygons Encodings

Based on the design tasks listed above, we designed a visualization method to summarize major communities in a large network. To reveal community quality, size, and other attributes, we extend Voronoi treemap [6] to represent the clustered graph G^C.

In our system, we use subdivided Voronoi cells to represent different communities, which has several advantages. First, the distinct polygonal shape of individual Voronoi cells can help people easily recognize different communities. Second, compared with typical visualization methods which encode a community using a sized circle, our method uses screen space more efficiently as it is a space-filling method. In addition, some community detection algorithms result in a hierarchical structure which can be conveniently presented using our method. Furthermore, the edges between adjacent Voronoi cells can be utilized to demonstrate community relations. We use color as another visual channel to illustrate different



(a) (b) (c) (d)

Figure 1: The layout generation process: (a) Use MDS to gather strong connected cluster nodes together. (b) Original Voronoi treemap represents different clusters with the size attribute. (c) Shrink each Voronoi cell to form cluster polygons and cluster gaps. (d) Arrange external nodes along cluster gaps and adapt the corner-cutting algorithm for each cluster polygon.

communities, providing stronger visual cues to people for community recognition.

The size of a community can be intuitively encoded with the area of the corresponding Voronoi cell. Cluster quality is another piece of crucial information for quantitative evaluation and comparison of communities. The cluster quality can be computed by internal connectivity, external connectivity and their combinations in various ways [35]. In our design, we use the curve smoothness to encode cluster quality. The higher the cluster quality, the smoother the polygon boundary is. By observing the shape, color, size and curve smoothness of Voronoi cells, people can easily recognize different communities and visually compare their attributes. Using this method, design task T1 can be accomplished.

According to design task T2, apart from illustrating community properties, relation strengths between communities should be visualized too. From the Gestalt Law of Proximity [8], people perceive objects to have strong relations when they are positioned together. Based on this principle, we choose to position communities with strong relations geometrically close. In this way, we can get the relation strengths between communities by inspecting the Euclidean distance between communities.

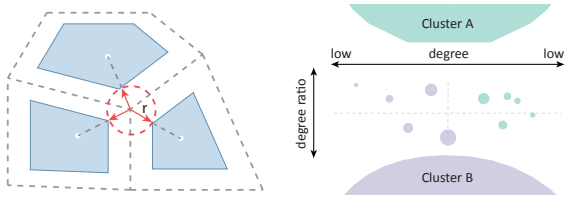
4.2 Boundary Nodes Encodings

As described in design task T3, boundary nodes are important to the community analysis. To create space for boundary nodes, a polygon-shrinking algorithm is developed and applied to each Voronoi cell such that a gap will be created between each pair of shrinked cluster polygons (see Fig. 2(a)). Boundary nodes which connect two or more communities are positioned along these gaps (see Fig. 2(b)).

To better reveal the relation patterns between communities, we design a degree-sensitive layout algorithm for boundary node layout. As illustrated in Fig. 2(b), there are two kinds of nodes positioned along the gap between clusters A and B: 1) cluster A's boundary nodes that linked with nodes from cluster B. 2) cluster B's boundary nodes that linked with nodes from cluster A. For each cluster, we make a central axis from this cluster towards the other cluster and its boundary nodes are always allocated on the right side of this axis. The larger the node degree is, the closer it is positioned near the central axis. Besides, for node v in a specific gap, we calculate the ratio of its internal degree and external degree towards the other community k: $r = d_I(v)/d_{E_k}(v)$. Nodes with the same ratio are allocated in the same parallel axis. For nodes with $r = 1$, they are arranged in the middle axis. Otherwise, nodes with more edges connected to a community are positioned closer to the corresponding cluster polygon. By adapting this degree-sensitive layout, nodes with similar degrees from one community are grouped together. To visually highlight nodes with high degrees, we use the node radius as the second visual channel to encode node degree.

Next we discuss several issues to be addressed for our visual design:

Gap determination. For a node connected to three or more clusters, we need to determine which gap the node should be allocated



(a) (b)

Figure 2: (a) Voronoi cell shrinking; (b) Boundary node encoding.

to. There are two possible solutions: 1) Make a duplication of that node in each gap if necessary; 2) Allocate the node in the gap between its cluster and the cluster with which it has the largest number of connected edges. Considering that it might cause misunderstanding when users want to count the number of high-degree boundary nodes of one community if we duplicate several copies of one node in an overview graph, we choose the second solution as our default setting. However, users are able to switch to the first solution in our system at any time.

Gap length difference. Due to the limitation of the Voronoi treemap generation algorithm, the generated Voronoi cell edges might have quite different lengths. Therefore, the gaps between various pairs of communities are also different which prevents people from comparing the boundary node distributions in different gaps. To solve this problem, we propose a detailed view for boundary nodes at cluster gaps. Users can select two adjacent cluster polygons in the overview graph, and then the corresponding detailed view for that gap will be generated. These detailed views are of the same size and scale which helps people explore and precisely compare the boundary nodes between different communities.

Boundary node overlapping Assume that there are several boundary nodes of a very high degree between two communities. Even in the detailed view, all the other nodes with low degrees may overlap and obscure these high degree nodes since they are all positioned near the border based on the degree-sensitive layout algorithm. We propose two solutions for this problem: 1) Apart from providing a linear scaling method for degree-position mapping, we also provide a log-based scaling method to separate the overlapping nodes. In addition, we support a zooming interaction in the detailed view to further differentiate those nodes.

4.3 Interactions

Our visualization system supports rich user interactions to achieve T5 in the design goals.

Cluster merging. Users are able to select any number of clusters and group them together to make a merged cluster. The selected clusters are regarded as sub-clusters while their size and color attributes are kept.

Cluster splitting. By selecting the merged cluster and clicking the splitting button, users are allowed to split aggregated clusters.

Cluster rearranging. In our initial layout algorithm, cluster polygons with strong relations are located geometrically adjacent. However, weak relations between cluster polygons positioned a

long distance from each other will be difficult to examine. We provide an interaction to overcome this drawback. Users can select any two cluster polygons and click the Rearranging button on the tool bar, and then a new Voronoi layout with these two clusters adjacent will be regenerated.

Cluster dividing. As described in design task T4, our visualization method should explore the substructures in a community. Taking advantage of the hierarchical structures in the Voronoi treemap, users can show the next level Voronoi diagram by selecting a cluster polygon of interest and clicking the Substructure button. In addition, our system can draw a force-directed layout to show the topological structure for internal nodes within a community when the number of nodes is reduced to a manageable level.

Node highlighting. We adopt a node highlight technique for users to further explore a specific node of interest. Users can hover over the node they want to inspect and a tooltip will pop up to display the summary information of that node.

For cluster merging, cluster splitting, and cluster rearranging, the MDS algorithm will be re-executed in order to get the new community layout from the updated community relations. Since MDS results may change totally even if there is only a small modification in the similarity/dissimilarity matrix, the community might be transformed to another location after performing one of the above three interactions. To better preserve users' mental map, we add animations to illustrate the transformation process. For the cluster splitting operation, the split clusters are located around the former aggregated cluster in our implementation.

4.4 Alternative Designs

A key function of our design is to demonstrate community quality. Though we use the curve smoothness to encode the community quality, we considered other options such as color, blur, enhanced border. Fig. 3 shows four major encoding methods. We briefly compare these methods and a formal user evaluation is presented in Section 6.2.

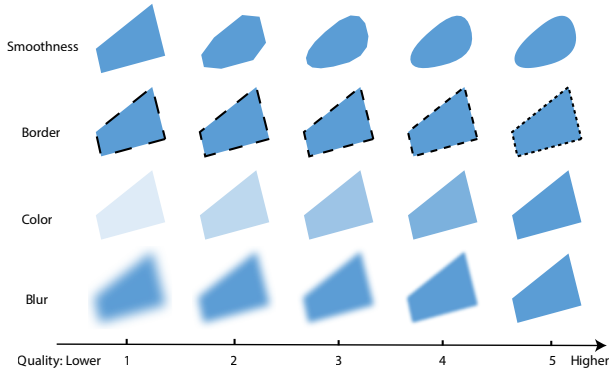


Figure 3: Four major encoding methods for community quality using curve smoothness, border stroke, color saturation, and blur. The community quality improves from left to right.

Curve smoothness Edge shape is an intuitive visual channel for human beings to recognize. Vehlow [34] encodes cluster fuzziness as a star-shape polygon. However their method is not suitable for our tasks as the star-shape design does not use space efficiently and thus affects the boundary node layout in the cluster gaps. To intuitively represent the community quality and keep the boundary nodes layout at the same time, we propose a curve smoothness encoding scheme.

Border We can make use of a polygon boundary to show community quality. The polygon border can be used in two different ways. First, the width of a boundary can clearly represent a continuous scalar value. Second, we may also use dashed boundaries. Different degrees of dash and gap length can represent different levels of quality. However, a short edge with a thick border or a wide gap dash may lead to ambiguity compared with long edges. Due to the varied length of Voronoi cell edges, these two methods may not suit our visual design.

Color It is possible to use color saturation, opacity or brightness to indicate community quality. Human beings have a good perception of color, however the color channel is better for indicating categorical data. From task T1, we need to show different communities and it is better to reserve colors to represent different communities. Reusing the color channel for quality may confuse users.

Other visual channels We also try to encode community quality with other visual channels like blur, sharpness and texture. Nonetheless, these visual channels cannot provide aesthetic displays. Furthermore, these designs cannot be precisely perceived by people and people have difficulty in estimating the values these designs represent.

5 System Overview and Implementation

In this section, we introduce our system architecture and overall visualization process. First, in a *data processing* stage, raw graph datasets of different formats are imported, filtered and organized into a unified form. This stage allows using kinds of clustering algorithms to detect possible communities for dataset without labeled community information. For each dataset, a clustered graph is generated so that weighted-nodes represent communities of different size and weighted-edges represent the edges between connected communities. Both the original graph's and clustered graph's topology information are recorded. Network attributes like node degree and modularity are pre-calculated for further usage. Fig.4 illustrates the pipeline of our visualization process.

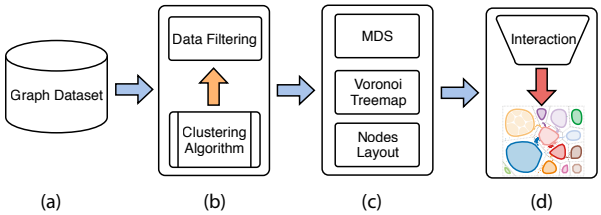


Figure 4: Overview of our visualization process pipeline. (a) Raw graph datasets; (b) Data processing stage; (c) Layout optimization stage; (d) User interaction stage.

The user interface of our system consists of multiple components as illustrated in Fig. 5: The dataset selection menu and the interaction toolbar are on the top of the system. From the main view, we can see a Voronoi treemap based visualization that summarizes major communities in a large network. On the right top corner, there is a detailed view which is used to explore and compare boundary nodes in polygon gaps. And on the right bottom corner, we put a statistics view to illustrate quantitative attributes of different communities and the node-link view to show the topological structure of communities. We also have a visualization control menu to allow users to dynamically switch between different visual encoding schemes.

The entire system was mainly implemented in Flask[1] and D3[2]. Next, we describe the implementation details of the *layout optimization* stage.

Multidimensional scaling(MDS). An intuitive way to display community relations is to position communities with strong relation geometrically close. Clearly, there exist many methods to define community relation strength. Here, we are primarily concerned with the number of edges between communities, which is a simple but effective way to measure community relations. But our method is not limited to this relation strength definition, any other relation

[1]Website: http://flask.pocoo.org/
[2]Website: http://d3js.org/

criterion can be adopted. We use Multidimensional Scaling(MDS) [25] to achieve this goal. Since our purpose is to put communities with strong relations together, the dissimilarity matrix used in MDS is computed based on the community relation weight matrix W as discussed in Section 3.1.

Voronoi treemap layout. As mentioned before, we use the Voronoi treemap to represent the hierarchical structure of community in our design [7]. We adopt the Voronoi treemap generation algorithm of Nocaj and Brandes [30] by using Aurenhammer's algorithm [5] for weighted Voronoi diagrams. The generation algorithm is an interactive process which works as follows:

- Initialize each site with the precomputed position and weight.
- Compute an initial weighted Voronoi Diagram.
- Iterate until the iteration number achieves the predefined iteration limit or the result is satisfactory:
  - Move sites to the centroid and reduce the corresponding weights if necessary.
  - Recompute the weighted Voronoi diagram.
  - Adapt weights to better match Voronoi cell area to ideals.
  - Recompute the weighted Voronoi diagram.

Though the final site positions are very likely to change after the above iterative process, the relative positions of the corresponding Voronoi cells have been kept. Thus, the relative distance between cluster polygons can be used to encode community relation strength. We use the community site positions calculated by MDS as the initial Voronoi cell site positions. As described in Section 4.1, we use the area of each Voronoi cell to encode the corresponding community size. The initial site weights are directly initialized to be sizes of communities too.

Cluster polygon shrinking. According to the definition of Voronoi diagram, all the points in one Voronoi cell are closer to its Voronoi site than to any other sites. As Fig. 2(a) illustrates, taking Cluster A as an example, the cluster polygon shrinking algorithm works as follows: Suppose its Voronoi site's coordinates are $p_A(x_A, y_A)$, for each polygon vertex $p_i(x_i, y_i)$, we define vector $\overrightarrow{v_i} = (x_i - x_A, y_i - y_A)$ and the shrinked vector $\overrightarrow{u_i} = (|\overrightarrow{v_i}| - C_{Scale}) \cdot \overrightarrow{v_i}/|\overrightarrow{v_i}|$ where C_{Scale} is a constant shrinking scale value. Then, the shrinked polygon vertices coordinates are changed to $p' = (x_i + u_x, y_i + u_y)$. Another approach is to define $\overrightarrow{u_i} = \overrightarrow{v_i} * C_{Scale}$. Although this approach can scale down each Voronoi cell, this may result in very short gaps around small polygons.

Corner-cutting and cluster quality. By default, the edge curve smoothness encodes cluster quality. To differentiate varying degrees of smoothness, we use Chaikin's corner-cutting algorithm [9]. After considering various community quality criterias, we decide to make *conductance* as our default quality criterion. Conductance is a simple but useful measurement for community quality. It represents the ratio between the number of edges inside the community and the number of edges leaving the community [24].

6 Experiments and Evaluations

In this section, we first present the performance of our system, then we evaluate four different community quality encoding schemes with a controlled user study. Finally, we select two example datasets for case studies to illustrate how our design can help reveal major communities in a large network.

6.1 Performance Evaluation

There are three major steps in the layout optimization stage: MDS layout, Voronoi treemap layout, and boundary nodes layout. The time complexity of MDS layout algorithm is $O(n^2)$. By adapting the methods proposed by Nocaj and Brandes[30], the Voronoi treemap layout algorithm is $O(nlogn)$ for each iteration. Since our system is designed to support dozens of communities in each layer, the time complexity is acceptable. The last step is an O(n) time complexity algorithm. As described in Section 5, we implemented our system as a web service. We use a MacBook Pro with 2.6GHz Intel Core i5 CPU with 8GB memory as the Web server, and conduct the case studies in the Chrome Web browser (version 38). The system can perform interactively after the data processing stage.

6.2 User Study: Community Quality

To evaluate the design options for encoding community quality, we designed an experiment which compares four different methods shown in Fig. 3. We recruited 22 unpaid undergraduate or graduate students (18 male, 4 female) from a variety of majors for the experiment. The experiment environment was built as an interactive Web system using JavaScript and Python. Each subject needs to answer 40 questions (10 for each method). In each question, subjects were given a polygon and asked to specify its quality. The quality ranges from 1 to 5 and is encoded by one of the four methods. Before the test, the subjects can practice test questions with answers so that they can be familiar with the encoding methods before the test. For each question in the test, the subjects were instructed to click on a start button which shows the question and triggers the timer. The timer would stop once the subjects answer the question. The response accuracy and response time were recorded. The response time was recorded in milliseconds.

We computed the average response time of participants categorized by the four methods and found that the participants spent 3.76 seconds in the color saturation condition(SD = 3.53), 5.92 seconds in the border stroke condition(SD = 6.60), 5.21 seconds in the curve smoothness condition(SD = 4.01), and 4.63 seconds in the blur condition(SD = 4.25). Regarding the response accuracy for the questions of each method, on average, the participants answered 90% questions correctly in the color saturation condition (SD = 10%), 50% questions correctly in the border stroke condition (SD = 24.7%), 73.6% questions correctly in the curve smoothness condition (SD = 12.3%), and 59.5% questions correctly in the color blur condition (SD = 16.4%).

We used the Representations Repeated Measure Analysis of Variance test to analyze the response time and accuracy. It revealed a significant effect for Representations in response time ($F(3,63) = 10.38, p < 0.001$) and accuracy ($F(3,63) = 22.17, p < 0.001$). Pair-wise comparisons showed that the response time for the border stroke method is shorter than for the color saturation method ($p < 0.001$) and the response time for the curve smoothness method is shorter than for the color saturation method too ($p < 0.001$). From the pair-wise comparisons, we can also see that the response accuracy difference between the blur method and the border stroke method is not significant($p > 0.05$), while other method pair combinations have a significant difference in response accuracy ($p < 0.05$).

The results show that the color saturation method outperforms other methods in both response time and accuracy. The curve smoothness method has a better response accuracy compared with the border stroke method and the blur method while it has no significant response time benefit. As discussed in Sec. 4.4, the color channel is better for indicating categorical data. People may also make mistakes when perceiving and comparing the saturation of different colors. Thus we select the curve smoothness method as our default solution for encoding community quality. However, users are free to switch between these four methods in our system.

6.3 Case Study I: DBLP

Our first dataset is extracted from DBLP [3]. In this dataset, each node represents a paper while each edge connecting two nodes means these two papers have at least one common author. Conferences are regarded as communities and each paper belongs to the conference it is presented at. For this dataset, we want to achieve

[3] http://dblp.uni-trier.de/db/

the following goals: 1) **G1**: Visually summarize the number of papers presented at different conferences and the relevancy of these conferences. 2) **G2**: Discover potential relations between conferences in different domains. 3) **G3**: Demonstrate that some papers presented at one conference can be highly related to other conferences at the same time.

We first select eleven conferences in the sub domain of computer science including Computer Networks, Operating System, Computer Architecture and Programming Languages, and the like. This dataset comprises 1032 papers published from 2003 to 2005. The results are shown in Fig. 5. From the figure we can see that these eleven conferences are positioned and grouped based on their relevancy. In Fig.5 (a) four conferences in the Programming Language field are grouped on the left-hand side. Note that ASPLOS is a conference not only targeting at the area of Programming Language but also the Computer Architecture technology. Thus, it is positioned near the ISCA, which is one of the top conferences in Computer Architecture. In Fig. 5 (b), two conferences in the field of Computer Networks are grouped on the top right corner. Another two operating system related conferences stand on the right-hand side as illustrated in Fig. 5 (c). The area of each Voronoi cell shows the number of papers published at the corresponding conference. Taking ISCA and PPoPP as an example, it is obvious that the area of ISCA Voronoi cell is nearly twice the area of PPoPP. It turns out that PPoPP accepts nearly 30 papers every year while ISCA accepts more than 50 papers annually. This result shows that our system can accomplish goal G1.

Our default cluster quality evaluation is based on conductance which exposes the ratio between internal edge number and external edge number for one community. In Fig. 5, we can easily see that ISCA's quality is better than ASPLOS's quality and HPCA's quality. Both the boundary nodes between HPCA and ISCA or the boundary nodes between ISCA and ASPLOS are closer to ISCA. This phenomenon means these boundary nodes have more connections with ISCA than HPCA or ASPLOS. For each pair of communities, though their boundary nodes have equal number of external degrees, ISCA has a larger internal edge number. This is why ISCA has better quality.

To achieve goal G2, we need to pay attention to the boundary nodes. For example, there are many boundary nodes along the gap between PLDI and OOPSLA which implies a strong relation between them as both conferences are in similar fields. On the contrary, FAST, a conference targeting at file systems and storage technology has a weak relation with conferences on Computer Architectures like ISCA and HPCA. As file system field is related to the operating system field to a certain extent, we want to explore the relation between FAST and HotOS. For the detailed view of the boundary nodes in Fig. 5 (d), we can see that there is a relatively strong relation between FAST and HotOS. An unexpected finding is that almost all the boundary nodes are laid on the FAST side. This suggests that the authors who presented at both HotOS and FAST are likely to publish more papers at FAST.

USENIX ATC is a comprehensive conference focusing on a wide range of topics. From the overview, we can see that many boundary nodes of USENIX ATC are positioned closer to FAST, HotOS, and SIGCOMM. Fig. 5 (e) shows the detailed view of the gap between USENIX ATC and FAST. One paper presented at USENIX ATC is laid near the middle of the cluster gap. From the tooltip we know that this paper has common authors with twelve papers presented at FAST. The title of this paper, "A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers", also validates that it is highly relevant to the field of files systems. From this example, we can see the goal G3 can be achieved in our visualization method.

These explorations suggest how our visual design can be utilized to summarize information about conference papers and reveal the connection patterns between conferences based on the coauthorship relations.

6.4 Case Study II: JDK Dependency Network

We use the software class dependency network data of the JDK 1.6.0.7 framework [1], collected from the Koblenz Network Collection (KONECT) [26], as our second example. Nodes in this dataset represent classes, and edges between them indicate that one class depends on the other.

As illustrated in Fig.6, we first examine the classes within the java.* package which have 2,379 classes (vertices) and 14,619 dependencies (edges). First, we found that the Voronoi cell of package java.lang is surrounded by lots of boundary nodes. Most of the boundary nodes belong to other packages like java.util, java.awt and java.text, and the like. The majority of these boundary nodes are located away from the middle axis, which means these boundary node degrees are generally small. This phenomenon illustrates that many other classes have dependencies on classes in the java.lang packages. The Java.lang package is the package that contains core classes related to language functionality and runtime system. This explains why so many classes have dependencies on the java.lang packages. By exploring the boundary nodes between java.lang and java.awt, we find an unexpected pattern that the java.lang package also has some dependencies on the java.io package. This is because many classes in java.lang need to import Exception classes and IO classes into the java.io package.

Next we examine the relationship between Java.awt and Java.util. From the figure we can see that lots of boundary nodes from java.awt package form an almost straight line. To solve the clutter problem due to the large number of nodes, we hide the boundary nodes from the java.util package and expand the width of boundary nodes from the java.awt package. This phenomenon does not only show that many classes in the java.awt package have dependencies on the java.util package. In addition, all these classes only depend on a few classes in the java.util package. From the detailed view, we can see that the classes with a high dependence degree are data structure classes like Collection and Set. The layout classes in java.awt need to use these data structures to manage GUI items. A few handler classes are necessary for classes in the java.awt package as well.

In addition, we find that there are several nodes with very large degrees laid on the central axis of the gap between java.util and java.lang. These nodes include java.lang.Class, java.lang.String and java.lang.Object. All the classes are inherited from the java.lang.Object class by default. This case study suggests that using our visualization methods can help users explore the highly dependent classes in a software package.

7 Discussion

The case studies have clearly demonstrated that our method can reveal the major communities, their aggregation attributes and hierarchical structures, and their relationships in a large network. The visualization can also support rich user interactions for users to explore boundary nodes, refine and compare clusters, and zoom into each community for more detailed information. The basic framework is extensible and can incorporate other graph visualization techniques such as edge bundling and other visual encoding schemes for community attributes.

Our method also suffers several weaknesses. First, our method enables people to compare the size of communities by observing the areas of Voronoi cells. However, it is difficult for people to accurately estimate the area of a polygon. Thus, the method is only good for qualitative analysis but not quantitative analysis. Second, similar to the MDS method, our method can use Euclidean distance to encode the relation strengths of two communities. However, there is no guarantee that all the relationships can be preserved in the

2D layout. This is a fundamental problem for many point-based approaches that use distance to encode relationships.

8 Conclusion and Future Work

In this paper, we present an interactive visualization method based on the Voronoi treemap to reveal the major communities and their relationships in a large network. It provides a novel approach to help people understand community characteristics and investigate various relation patterns between communities. Compared with existing visualization methods, our approach gives an effective way to depict community attributes like size and quality. In addition, boundary nodes that connecting different communities are illustrated in a degree-sensitive way. Users are allowed to explore and compare boundary node patterns with the detailed views. Interactive community adjustment including moving, merging and splitting are provided with animated transitions. Two case studies with the DBLP co-authorship data and JDK class dependency data have demonstrated the effectiveness of our methods. Our future work includes filtering techniques to solve the problem of boundary node overlapping. We also plan to improve our method to illustrate more internal node attributes for each community.

9 Acknowledgments

We would like to thank Panpan Xu and Conglei Shi for valuable suggestions, and anonymous reviewers for constructive comments. This work is partially sponsored by Shannon Lab, Huawei Co.Ltd and we thank for their help and suggestions.

References

[1] JDK dependency network dataset – KONECT. `http://konect.uni-koblenz.de/networks/subelj_jdk`, 2014.

[2] J. Abello, F. Van Ham, and N. Krishnan. Ask-graphview: A large scale graph visualization system. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):669–676, 2006.

[3] B. Alper, N. H. Riche, G. Ramos, and M. Czerwinski. Design study of linesets, a novel set visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2259–2267, 2011.

[4] D. Archambault, T. Munzner, and D. Auber. Grouseflocks: Steerable exploration of graph hierarchy space. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):900–913, 2008.

[5] F. Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM Journal on Computing*, 16(1):78–96, 1987.

[6] M. Balzer and O. Deussen. Voronoi treemaps. In *IEEE Symposium on Information Visualization*, 2005.

[7] M. Balzer and O. Deussen. Level-of-detail visualization of clustered graph layouts. In *IEEE Pacific Visualization Symposium*, pages 133–140, 2007.

[8] U. Brandes, B. Nick, B. Rockstroh, and A. Steffebibn. Gestaltlines. *Computer Graphics Forum*, 32:171–180, 2013.

[9] G. M. Chaikin. An algorithm for high-speed curve generation. *Computer Graphics and Image Processing*, 3(4):346–349, 1974.

[10] C. Collins, G. Penn, and S. Carpendale. Bubble sets: Revealing set relations with isocontours over existing visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1009–1016, 2009.

[11] K. Dinkla, M. J. van Kreveld, B. Speckmann, and M. A. Westenberg. Kelp diagrams: Point set membership visualization. In *Computer Graphics Forum*, volume 31, pages 875–884. Wiley Online Library, 2012.

[12] N. Elmqvist, T. nghi Do, H. Goodell, N. Henry, and J. daniel Fekete. ZAME: Interactive large-scale graph visualization. In *IEEE Pacific Visualization Symposium*, pages 215–222, 2008.

[13] G. W. Flake, S. Lawrence, and C. L. Giles. Efficient identification of web communities. In *Knowledge Discovery and Data Mining*, pages 150–160, 2000.

[14] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

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

[16] E. R. Gansner, Y. Hu, and S. G. Kobourov. Gmap: Drawing graphs as maps. In *Graph Drawing*, pages 405–407. Springer, 2010.

[17] E. R. Gansner, Y. Hu, and S. G. Kobourov. Viewing abstract data as maps. In *Handbook of Human Centric Visualization*, pages 63–89. Springer, 2014.

[18] M. Ghoniem, J.-D. Fekete, and P. Castagliola. On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis. *Information Visualization*, 4(2):114–135, 2005.

[19] N. Henry and J.-D. Fekete. Matrixexplorer: a dual-representation system to explore social networks. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):677–684, 2006.

[20] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[21] Y. Hu, S. G. Kobourov, and S. Veeramoni. Embedding, clustering and coloring for dynamic maps. In *IEEE Pacific Visualization Symposium*, pages 33–40. IEEE, 2012.

[22] R. Jianu, A. Rusu, Y. Hu, and D. Taggart. How to display group information on node-link diagrams: An evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 20(11):1530–1541, Nov 2014.

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

[24] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM (JACM)*, 51(3):497–515, 2004.

[25] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[26] J. Kunegis. KONECT – The Koblenz Network Collection. In *Proceding of International Web Observatory Workshop*, pages 1343–1350, 2013.

[27] J. Leskovec, K. J. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th international conference on World wide web*, pages 631–640. ACM, 2010.

[28] W. Meulemans, N. H. Riche, B. Speckmann, B. Alper, and T. Dwyer. Kelpfusion: a hybrid set visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 19(11):1846–1858, 2013.

[29] C. Muelder and K. liu Ma. A treemap based method for rapid layout of large graphs. In *IEEE Pacific Visualization Symposium*, pages 231–238, 2008.

[30] A. Nocaj and U. Brandes. Organizing search results with a reference map. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2546–2555, 2012.

[31] N. H. Riche and T. Dwyer. Untangling euler diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1090–1099, 2010.

[32] Z. Shen, K. liu Ma, and T. Eliassi-rad. Visual analysis of large heterogeneous social networks by semantic and structural abstraction. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1427–1439, 2006.

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

[34] C. Vehlow, T. Reinhardt, and D. Weiskopf. Visualizing fuzzy overlapping communities in networks. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2486–2495, 2013.

[35] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, MDS '12, pages 3:1–3:8, New York, NY, USA, 2012. ACM.

[36] J. Yang, Y. Liu, X. Zhang, X. Yuan, Y. Zhao, S. Barlowe, and S. Liu. Piwi: Visually exploring graphs based on their community structure. *IEEE Transactions on Visualization and Computer Graphics*, 19(6):1034–1047, 2013.

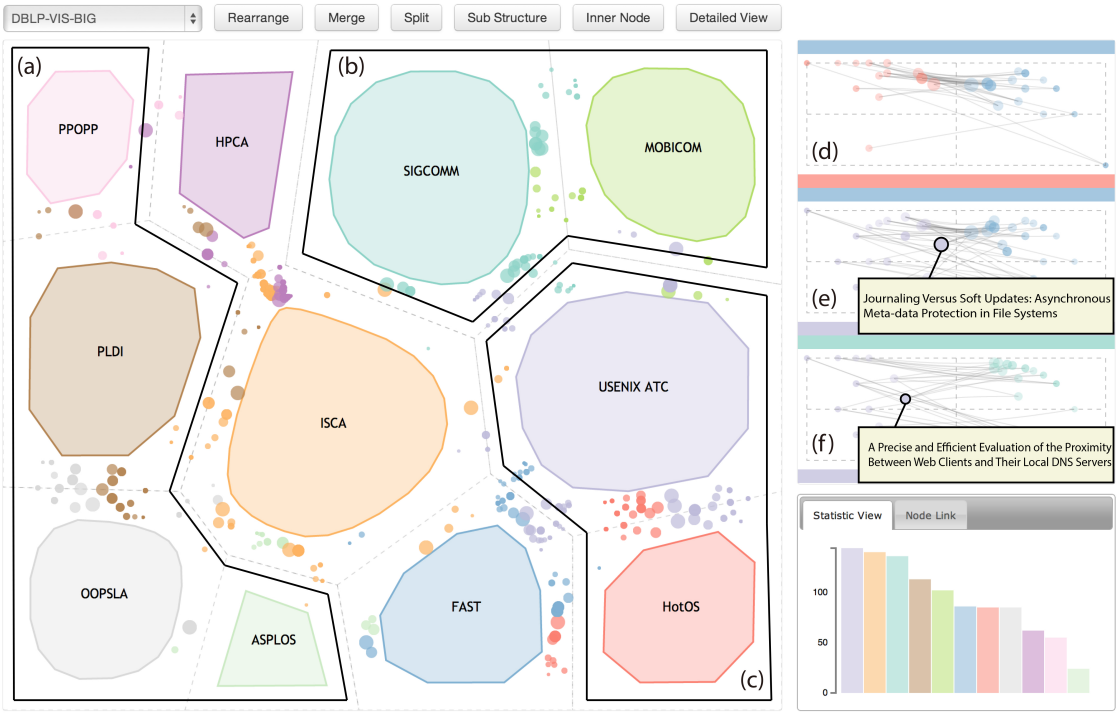


Figure 5: This figure shows the DBLP dataset consisting of eleven conferences in the fields of Programming Language, Computer Networks, Operating System and Computer Architecture, etc. Nodes represent papers and edges represent two papers having at least one common author.

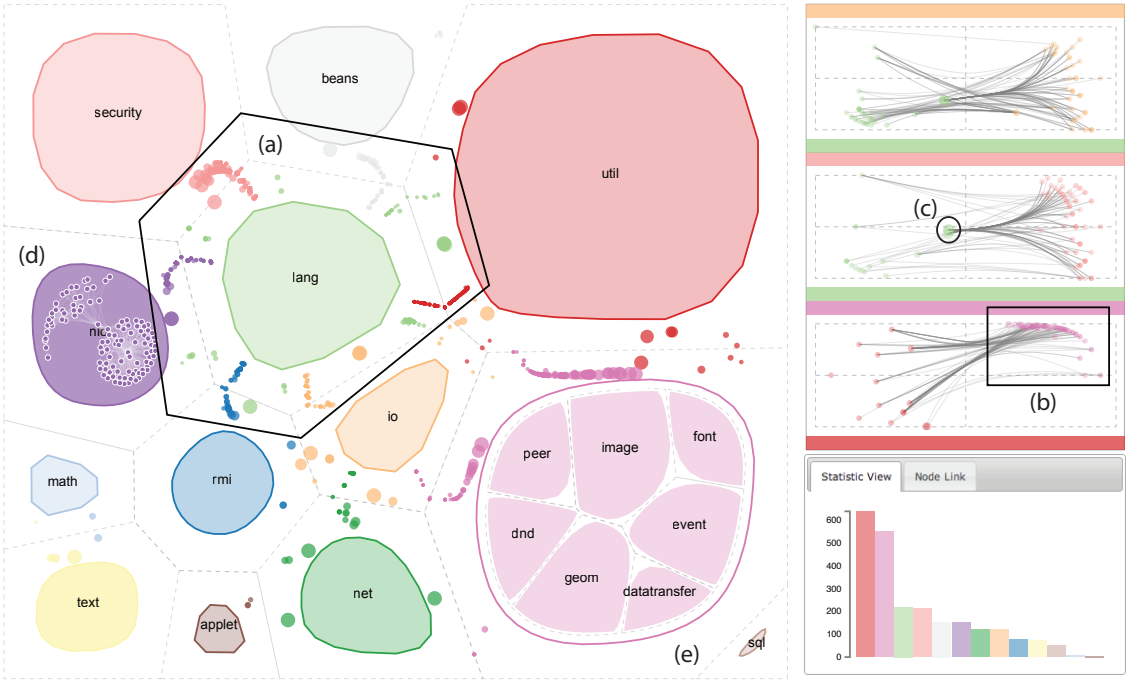


Figure 6: JDK java package dependency network comprising 2379 classes (vertices). (a) Package java.lang are imported by other classes by default. (b) Classes in package java.awt sometimes import classes from java.util. In the mean time, they depend on classes in the same package to a great extent. (c) Several individual classes from java.lang and java.util with high degrees. (d) The filtered topology structure for sub-package java.nio. (e) The second level structures of sub-package java.awt.