A Layout-Based Classification Method for Visualizing Time-varying Graphs

YUNZHE WANG, School of Electronic and Information Engineering, Suzhou University of Science and Technology and Department of Computing, The Hong Kong Polytechnic University
GEORGE BACIU, Department of Computing, The Hong Kong Polytechnic University
CHENHUI LI, School of Computer Science and Technology, East China Normal University

Connectivity analysis between the components of large evolving systems can reveal significant patterns of interaction. The systems can be simulated by topological graph structures. However, such analysis becomes challenging on large and complex graphs. Tasks such as comparing, searching, and summarizing structures, are difficult due to the enormous number of calculations required. For time-varying graphs, the temporal dimension even intensifies the difficulty. In this paper we propose to reduce the complexity of analysis by focusing on subgraphs that are induced by closely related entities. To summarize the diverse structures of subgraphs, we build a supervised layout-based classification model. The main premise is that the graph structures can induce a unique appearance of the layout. In contrast to traditional graph theory-based and contemporary neural network-based methods of graph classification, our approach generates low costs and there is no need to learn informative graph representations. Combined with temporally stable visualizations, we can also facilitate the understanding of sub-structures and the tracking of graph evolution. The method is evaluated on two real-world datasets. The results show that our system is highly effective in carrying out visual-based analytics of large graphs.

CCS Concepts: • **Computing methodologies → Machine learning**; • **Human-centered computing → Visualization design and evaluation methods**.

Additional Key Words and Phrases: time-varying graph, structural classification, simplified visualization

1 INTRODUCTION

Topological graphs are now at the base of many applications, such as bioinformatics, chemistry, social science, design, etc. ***Graph*** is a type of data structure that describes the relationship between a set of objects. In the graph context, we often refer to the objects and their mutual relationship with two pairs of terms: (*node, link*) and (*vertex, edge*), interchangeably. For example, we can abstract social network users as graph nodes and the interactions between them as graph links. When analysing the graph model, we may discover insightful patterns and answer higher level questions such as: what is the degree distribution of entities? which group of entities are more densely connected than others? is there any entity that dominate the entire graph?

To assist in the analysis, visualization is an effective tool because images make certain patterns obvious to observe [31]. However, with the explosion of data volume, we are faced with various problems, not only from the aspect of graph analysis, but from visual representations.

Authors' addresses: Yunzhe Wang, yunzhew1991@gmail.com, School of Electronic and Information Engineering, Suzhou University of Science and Technology, Department of Computing, The Hong Kong Polytechnic University; George Baciu, csgeorge@comp.polyu.edu.hk, Department of Computing, The Hong Kong Polytechnic University; Chenhui Li, chli@cs.ecnu.edu.cn, School of Computer Science and Technology, East China Normal University.

Manuscript submitted to ACM

Clustering is one of the techniques that are used to reduce the complexity of graph analysis. We can partition a large graph into multiple components that have smaller sizes. Besides, they may expose specific patterns based on the clustering criteria [67]. In this work, we refer to the components as communities. A ***community*** is a natural aggregation of graph entities. Entities within the same community are more densely connected than those in different communities. Extensive researches have been conducted on static graphs [8, 25]. However, in time-varying graphs, the temporal dimension brings new chanllenges in tracking the evolution of communities [3].

Scalablity is an outstanding issue in representing graphs. In the spatial domain, placing a large number of nodes and edges on a limited display causes heavy visual clutters. Along the temporal axis, we adopt the snapshot model to describe time-varying graphs [50]. Consequently, when the time steps accumulate, it is difficult to show and interact with the entire historical information.

We draw snapshots as node-link diagrams for the benefit of intuitive perception of topology. Given a large graph, our goal is to eliminate visual clutter without losing important information. Hence, we let each node in the diagram denote a community, but the original entity members of a community are collapsed and their connectivity status is no longer accessible from observation. We further make a compensation by summarizing the diverse community structures into a few topological patterns, and map them to meaningful visual encodings.

A ***topological pattern*** reflects the regularity of connections between nodes with no respect to the number of them. We incorporate four types of topological patterns and classify the subgraph induced by each community to one of them. Li et. al provided several patterns according to the common topology in graph theory [38]. Similar to their work, our patterns are called: *chain*, *loop*, *clique*, and *egocentric*.

In this paper, we propose a framework that facilitates time-varying graph analysis based on classification and visual simplification. The focus of analysis is at the community structure, e.g., whether a community has changing structures temporally. Different from previous work on graph classification [42, 43, 54], we take layout images rather than graph representations as input. Essentially, our solution is a transformation procedure followed by image classification. Our motivation is twofold: 1) it is difficult to learn proper representations that fully capture the characteristics of the entire graph [26]; 2) graph theory-based classification methods often cause high computational costs, and they require expertise to be understood [60, 64].

We firstly transform the data type from graph to image. The rationale is that a determined layout algorithm generates similar drawings for graphs of similar structures, as shown in Figure 1. From (a) to (e), all the graphs consist of 30 vertices. The layouts of (c), (d) and (e) look more similar because the graph data of (d) and (e) are generated by randomly removing a small fraction of edges from (c). ***Layout images*** are depictions of nodes and links on the screen at specific coordinates computed by layout algorithms. By learning visual characteristics, machines can distinguish different types of layout images, and thus separate graphs of different structures.

There are mainly two streams of approaches for representing the temporal changes of graphs: timelines and animations [5]. Compared to the former, animations are better for displaying a long sequence of snapshots, but are worse at conducting comparison tasks. Therefore, we provide smooth animations by mapping identical communities and fixing their positions on the screen. Animations are deemed smooth when the visual differences between consecutive frames are minimized. The mapping operations rely on a cross-time method. It extracts communities at each time step by applying the Louvain method [8] only once at the global level where all snapshots are aggregated into a Super Graph [16].

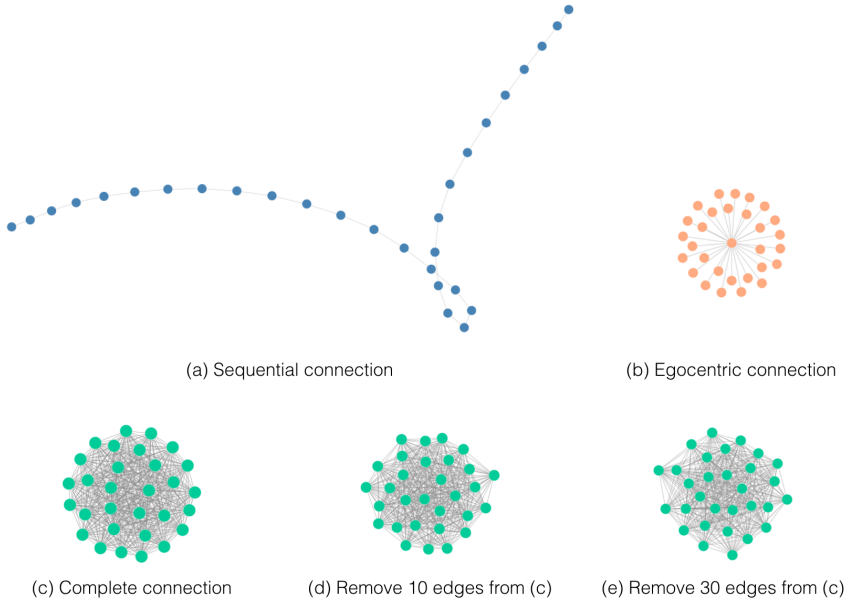


Fig. 1. When applying the Yifan Hu algorithm with the same configuration, similar graph structures result in similar layouts.

We improve the visual design by juxtaposing the pattern glyphs of two snapshots in one animation frame. The pentagonal shapes of glyphs also contribute to the smooth transition of frames. Consequently, it becomes convenient to understand the local connectivity, search for specific topology, and capture the global and local structural differences.

Our contributions are as follows:

- we simplify time-varying graph analysis on structures by visualization techniques. We not only summarize the structures of communities, but present them by animated node-link diagrams with well-design visual encodings, which are more intuitive and interpretable than pure graph representations. Supported tasks include searching for communities of similar structures and capturing temporal changes;
- we present a new way of thinking for graph classification. We try to analogise this difficult problem to a much simpler image-classification task. As far as we know, few work has been done to classify graphs according to the visual differences of their layout projections;
- we define meaningful topological patterns as class labels. The goal is to efficiently navigate users through diverse graph structures. In this work, we give four patterns for example. However, new patterns can be easily integrated into our system through a re-training procedure;
- we implement a low-cost dynamic community detection method. The membership of communities and their positions on screen are calculated at a global level. Meanwhile, we can preserve the visual consistency of node-link diagrams between consecutive snapshots.

2 RELATED WORK

2.1 Graph Classification

Classification of graph structures depends on the similarity measurement of connectivity, wherein embedding techniques are widely used. A graph embedding is a vector that encodes the characteristics of the original graph. Netsimile [7] obtains a signature vector of graph by inspecting and aggregating a set of topological features of individual nodes. Cai and Wang [13] put forward a similar approach also based on the local statistics of nodes, and the SVM(Support Vector Machine) method is used for classification. The two methods have the same complexity of feature extraction which is $O(|E|)$ where $|E|$ denotes the number of graph edges, and they mainly differ at the selection of topological features. However, the quality of the features is hard to evaluate.

Another stream of methods aim to search for specific subgraphs and use their frequency to learn representations for graphs. Nguyen et al. [45] adopted an analogy to neural document embedding. A graph is regarded as a document and subgraphs are treated as words. The algorithm simultaneously takes a set of graphs and returns their distributed representations. Graphlets are small connected and non-isomorphic induced subgraphs. The number of unique features grows exponentially with the increase of the number of nodes that a graphlet contains. A kernel that measures the similarity between graphs can be defined as the product of vectors that consist of the frequency of graphlets [62]. Subgraph-based methods are not suitable for large graphs because of the exhuastive enumeration of subgraphs. Therefore, Shervashidze et al. [55] proposed two speedup strategies: one is based on sampling and the other is designed for graphs with bounded degree.

Deep learning-based approaches have been extensively adopted to incorporate extra mechanisms into classification, such as the attention on noisy graphs where significant patterns only exist at certain regions [35]. Hence, only the attributes of informative nodes are needed for label prediction. Zhang and Chen [66] achieved graph-level embeddings from different aspects by packing node-level features into capsules and routing them to high-level layers in GNN (Graph Neural Networks).

Our method is different from all the aforementioned approaches in that we do not operate on the graph structure itself directly, but take it as the essential factor that decides the layout appearance. We can then classify graphs by discriminating the visual differences of their layouts.

2.2 Graph Partitioning

Partitioning tasks aim at dividing graph nodes and their incident links into disjoint groups with specific properties. Extensive research work have been done in this field. Commonly-used methods can be categorized into geometric-based [36, 41], spectral-based [6, 59] and multilevel-based [14, 15, 28].

The first type of methods depend on geometric information such as the coordinates of nodes, but ignore the contribution of links. Hence, their applicability is limited and they can hardly extend to scenarios of weighted links. Typical geometry graphs include grids and meshes. In 2-way partitioning, the basic idea is to find a hyper-plane or axis to separate nodes so that the derived two groups have approximately equal sizes.

The key to spectral-based algorithms is the Laplacian matrix which implies the structural properties of the original graph. The matrix is defined as $L = D - A$, where D is the degree matrix and A is the adjacency matrix. The set of eigenvalues of L is called the spectrum of graph. If the number of groups k is known, we can apply K-means or other clustering methods on eigenvectors of L and derive the corresponding node groups [44]. Otherwise, without knowing k in advance, techniques of maximizing the modularity can be used [1, 8, 11].

Ensemble clustering approaches take advantage of multiple clustering models and aim to produce results of higher qualities compared to those generated by individual models. By estimating a sparse adjacency matrix, the consensus method [40] largely reduces the complexity of spectral-based clustering. To manage the disagreement between different clusterings, strategies for aggregating models are proposed to improve the robustness of the final result [24, 56].

Multilevel algorithms consist of multiple processing stages. They recursively collapse nodes and links and then partition the smaller graph. One popular tool is METIS [30]. It is efficient and takes only a few seconds when managing graphs with millions of nodes. The objective is to minimize the number of cut edges and return divisions of balanced sizes. The drawback is that users have to specify the number of divisions in advance. This can affect the final results.

We partition graph snapshots at all time steps according to the community detection results at a global level. This method trades off the natural dense aggregation of nodes for achieving high consistency of temporal partitions.

2.3 Dynamic Community Detection

According to Rossetti et al. [50], existing dynamic community detection techniques can be categorised into three types: (1) Instant Optimal, (2) Temporal Trade-off and (3) Cross Time. For type (1), communities are detected independently for each snapshot and the overlap between communities at consecutive time steps is calculated. Methods of type (2) discover communities at the next time step based on communities at the last timestamp. The last type of methods consider graphs at all time steps as a whole and compute communities at a global level. From (1) to (3), the quality of community structures decreases while the temporal consistency increases.

Shang et al. [53] put forward a real-time algorithm aiming at tracking communities on a fine-grained level. The algorithm takes the result of the Louvain method at the first time stamp as an initial input, and updates it at following time stamps. Increased edges are classified into four types, each corresponding to different operations on previous communities, including keeping unchanged, combining and so on. However, the algorithm only deals with frequent and incremental changes and lacks the processing of the decrease of edges. By optimizing multiple objectives simultaneously with genetic algorithms, Folino et al. [21] get communities that trade off between the clustering accuracy and the deviation between time steps. In terms of streaming data, strategies of locally updating communities that are affected by adding or removing nodes or links are adopted [51]. The objective is to smooth the evolution of communities while avoiding external matching. Accordingly, a label propagation procedure broadcasts the changes to the neighbors of the node and adjust the local community memberships. This method largely decreases the running time of dynamic community detection. Zakrzewska and Bader [65] dynamically expand the seed set when graph changes and the community that each seed belongs to is updated. The algorithm also supports parallel processing. Gauvin et al. [23] factorize tensors that take time as one dimension. Graph structures are represented by adjacency matrices. The temporal activity patterns of communities can be extracted after factorization.

We propose a cross-time method out of the consideration to implement the smooth transition of animations. To detect communities, we adopt the Louvain algorithm [8] which is based on modularity maximization. Modularity is a scalar value which measures the density of links inside communities as compared to links between communities. It is one of the fastest implementations for large graphs with sizes of 100 million nodes and billions of links. Other highly efficient approaches for static graphs can be the InfoMap method [52] and the label propagation algorithm [49]. The complexity of the three methods are $O(|V| \log |V|)$, $O(|E|)$ and $O(|E|)$ where $|V|$ and $|E|$ denote the number of nodes and edges, respectively.

2.4 Visualization of Complex Time-Varying Graphs

When displaying time-varying graphs, we aim to promote the visual consistency, i.e., minimize the layout differences between snapshots. The reason is that abrupt changes may cause the discontinuity of users' mental map [48]. Incremental approaches are usually used to relieve the problem. Frishman et al. [22] obtained an initial layout by the force-directed algorithm. When nodes are added or removed, the layout is adjusted accordingly.

To represent the temporal axis, timeline-based designs facilitate users to compare snapshots at different time steps, but are criticized for the bad scalability [12]. Conversely, animations map the snapshot time to visual time and can scale as much as possible if the machine capability allows. Bach et al. [2] built a system called GraphDiaries for identifying changes in animated node-link diagrams. They also claimed that if the nodes pertinent to the visual tasks are highlighted through the entire time period, then a more stable drawing provides little benefit. Feng et al. [20] achieved smooth animations by firstly generating the initial graph layout of each snapshot from a super-graph [16]. Then they optimized the layout by fulfilling a few constraints. Compared to timelines, animations convey more information by embodying an unlimited number of time steps. Besides, in animations, individual snapshots make full use of the display, while in timelines, multiple snapshots share one display.

Large complex graphs can be visualized by aggregating nodes or links into groups to obtain high-level insights. Objects within the same group share similarities regarding attributes or topology [58]. Vehlow et al. [57] visualized the evolution of communities by presenting the time-varying graph and community structures in a single image. Consequently, users are supported to identify the relationship between the community evolution and the topology changes of the original graph. In their design, a graph is illustrated as a stack of ordered rectangles, each denoting a community. The topology of a community is drawn in the rectangle so that users can flexibly track the connection of a specific node. However, due to the limit of the display space, only a few time steps can be presented. Dunne and Shneiderman [17] pointed out that fan, connector and clique are three types of prominent motifs in social networks. The authors put forward custom algorithms of searching for the motifs. To simplify node-link diagrams, the discovered motifs are replaced by well-designed visual glyphs.

Our work is similar to [17], but mainly differs at two aspects. First, we do not exhaustively search for patterns of interest over the entire graph. Instead, we aggregate nodes into communities, and determine the topological pattern of the underlying graph of each community. Second, we target dynamic graphs and provide animated representations that support graph comparison at consecutive time steps.

3 SYSTEM OVERVIEW

We establish a visual system that provides simplified representations of large and complex time-varying graphs. This system aims to assist users in tracking the evolution of graph structures and searching for the topology of interest. Visual simplification is basically achieved by regarding the communities as the unit of visualization. Figure 2 shows the main stages of our work.

In the data processing procedure, we apply temporal aggregation to obtain the desired time granularity (e.g., a year). Larger granularity may result in less snapshots to process, but more graph dynamics would be concealed.

Communities of each snapshot are extracted in the graph partitioning step. But first we need to construct a Super Graph (SG) by combining all the snapshots. The Super Community (SC) is a set of communities detected from the SG. Taking the communities in SC as nodes, the corresponding layout determined by a force-directed algorithm [18] is

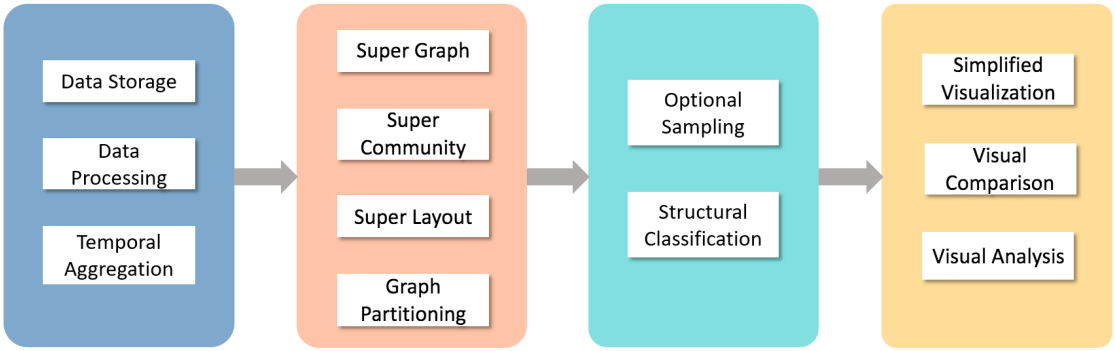


Fig. 2. The pipeline of building the visual system. It consists of four stages: data processing, graph partitioning, structural classification and conducting visual tasks.

called the Super Layout (SL). SC and SL are the references for computing communities and their layout positions at individual timestamps. The details of this process are explained in Section 5.1.

It is worth noting that calculating SG, SC and SL is the key to smooth animations. Besides, our method is time-efficient. We do not apply community detection and force-directed layout algorithms repeatedly to each snapshot. Instead, they are executed only once at the global level.

The classification of graph structures depends on the understanding of layout images. However, the graph drawings of large communities are likely to suffer from visual clutters, and machines may fail to distinguish them. We solve this problem by an optional sampling operation for communities whose sizes exceed the limit.

Since the visual simplification causes the loss of local information, we provide four pattern glyphs in Section 6 to imply community structures. We also explain how the glyph designs help to maintain the visual stability and how the integration of glyphs and graph diagrams benefit the comparison tasks in animations.

4 STRUCTURE CLASSIFICATION

4.1 Pattern Description

A topological pattern is a template graph from which we can easily perceive the connectivity characteristics. Suppose there are n vertices in the graph, and we define four types of patterns:

- ***chain***: n vertices are linearly connected by $(n-1)$ edges;
- ***loop***: a path of n vertices with the two end-vertices connected;
- ***clique***: n vertices are fully connected;
- ***egocentric***: a vertex is located in the center and all the remaining vertices connect to it.

In the *chain* pattern, vertices are connected in a sequence, and they potentially reflect the ordering information. The *loop* pattern contains a closed path, corresponding to a sparse connectivity. Conversely, vertices are densely connected in the *clique* pattern. The *egocentric* pattern has been extensively studied in the context of social networks [39, 61]. It leads us to find dominant entities.

According to Bollobás [9], there are various topological patterns that attract the attention of researchers. We select the above four patterns out of them because we concentrate on introducing a new idea of classifying graphs based on the visual differences of their layout images. Therefore, we adopt these four prevalent patterns for demonstration. More

importantly, they are interpretable and unique to make the corresponding layouts easy to distinguish by machines. The pattern drawings are displayed in the left column of Figure 3, using five or six nodes for simplicity.

Since we are unable to cover too many patterns for the sake of conciseness, we list a few alternative patterns in Figure 4 as supplements. It is convenient to integrate new patterns into our system, and the key is to generate a training set of new patterns and re-train the classification model.

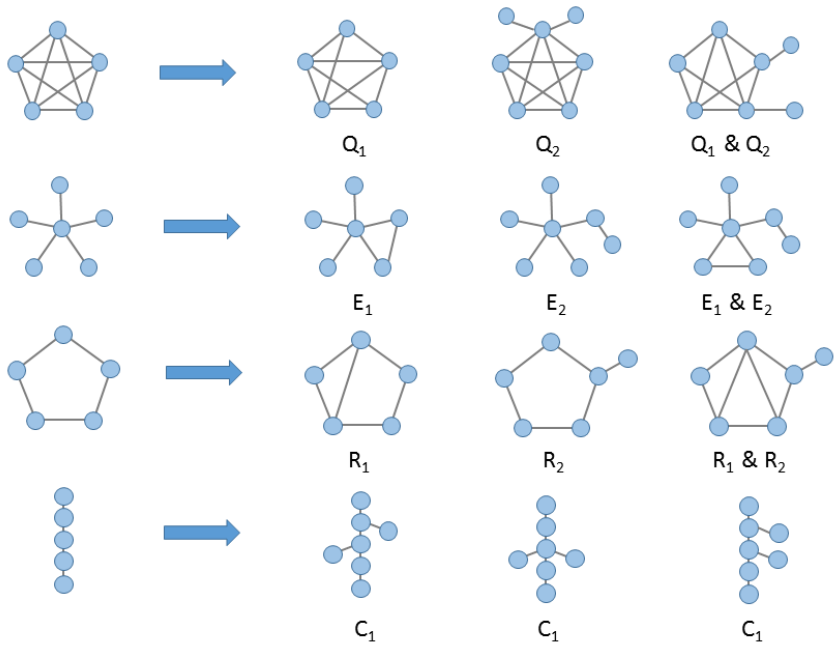


Fig. 3. Four types of topological patterns: *clique*, *egocentric*, *loop* and *chain* (left column) and their variants (right column).

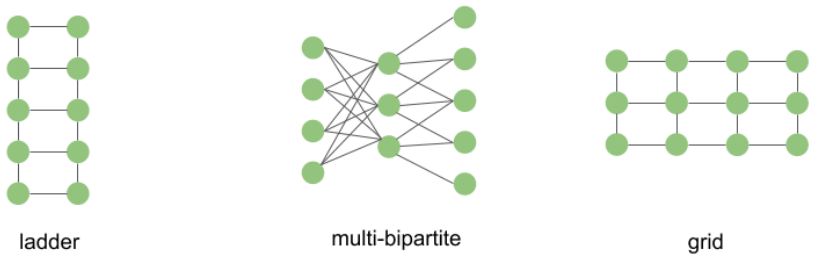


Fig. 4. Alternative topological patterns.

4.2 Sampling

We suggest an optional sampling operation on communities, for two reasons: first, large graphs cause visual clutters, and machines might fail to recognize the layout images of a certain class; second, users can hardly verify the classification result by observing the occluded graph drawings.

According to Nguyen et al. [46], we provide the following sampling strategies on vertices, edges, and explorations, respectively, and allow users to select any one of them by comparing the sampling results:

Random Vertex Sampling: randomly select a set of vertices from the graph and return the subgraph induced by these vertices;

Induced Random Edge Sampling: randomly select a set of edges from the graph and return the induced subgraph on vertices incident to at least one of the selected edges;

Random Walk Sampling: randomly select a starting vertex and then simulate a random walk. The sampling result includes all the visited vertices and edges.

We evaluate the sampling results by calculating their structural similarities with the original graphs. A higher similarity score suggests a better result. Therefore, we adopt the Netsimile [7] method which encodes a graph into a signature vector. Then, the similarity score can be obtained in the vector space.

The method works on constructing a feature matrix. Suppose a graph has n vertices, each having a k-dimensional feature. The matrix is defined as:

$$M = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1k} \\ f_{21} & f_{22} & \cdots & f_{2k} \\ \cdots & \cdots & \cdots & \cdots \\ f_{n1} & f_{n2} & \cdots & f_{nk} \end{bmatrix}, \tag{1}$$

where f_{ij} is the jth feature value of vertex i. A feature is a value representing the node degree, the clustering coefficient, and so on. A feature matrix is then converted to a signature vector,

$$V = [f(M_1), g(M_1), h(M_1), \ldots, f(M_k), g(M_k), h(M_k)], \tag{2}$$

where M_1 is the first column of M; f, g and h represent median, mean and standard deviation aggregators, respectively. The similarity score is equivalent to the reciprocal of the *Canberra distance* between two signature vectors V and V':

$$\text{distance}(V, V') = \sum_{i=1}^{3k} \frac{|V_i - V_i'|}{|V_i| + |V_i'|}, \tag{3}$$

where $3k$ denotes the number of components in vectors, and $|\cdot|$ returns the absolute value. The run time complexity of Netsimile is linear on the number of edges, and the score values lie in [0,1], with 0 and 1 imply the very different and the same structures, respectively.

Sampling results are then passed to the classification model to determine the type of topology. As the sampling method makes the original graph structure incomplete, the matched topological pattern is likely to be a happenstance. Therefore, we repetitively conduct the sampling-and-classification operation on the induced graph of any oversized community. The pattern that occurs the most frequently will be selected as the final result.

4.3 Classification Model

We classify the sub-graphs induced by all communities into four types of topology as defined in Section 4.1. Graphs that belong to the same class may have varying structures, but they share similar backbones that can be depicted by the corresponding pattern.

Traditional solutions of graph classification need to transform graphs to numeric representations that facilitate the execution of machine learning tasks [7, 34]. However, they involve complex calculations of many topological properties and require iterative graph searches. Hence, these methods are time-consuming and not applicable to large graphs. Kwon et al. [34] pointed out that, given a determinate layout method, topological similarities contribute to perceptual similarities. Hence, the classification of layout images should return consistent results with the classification of corresponding graph structures.

Our method predicts the class label of a graph by passing its layout image to the classification model. The image shows the node-link drawing of the graph. In our implementation, the visual positions of nodes and links are determined by a force-directed algorithm [18]. Users can add configurable constraints on node positions, such as a fixed orientation. The layout algorithm treats nodes and edges as charged particles and springs that attract or repel the incident particles, and it runs iteratively until the energy of the entire system E is minimized [32]:

$$E = \min \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{i,j} (\left| p_i - p_j \right| - l_{i,j})^2, \tag{4}$$

where n denotes the number of nodes; $k_{i,j} = K/d_{i,j}^2$, and K is a constant and $d_{i,j}$ is the shortest path distance between node i and j in the graph; p_i and p_j are the coordinates of nodes on the screen; $l_{i,j} = \frac{L_0}{\max_{i<j} d_{i,j}} d_{i,j}$, and L_0 is the length of a side of the screen. The energy function actually represents the minimization of the difference between the geometric and graph distances.

Notably, the configurations of generating graph layouts during the training and prediction stages must keep the same. Because graph drawings vary considerably when using different layout algorithms or parameters, which might reduce the classification accuracy.

Model: we adopt a pre-trained neural network, the ResNet-18 [27] for layout classification. It optimizes a residual mapping instead of an identity mapping. Apart from the basic building blocks, there are bottleneck modules, reducing the dimensions of deeper networks. ResNet-18 is a stack of basic building blocks. Cross-entropy is used to calculate the loss. The initial learning rate is 0.01 for the stochastic gradient descent optimizer. And it decays by 0.25 every 7 epochs. We add a softmax layer to normalize the output to a probability distribution over the four classes and choose the class with the highest probability to label the input sample.

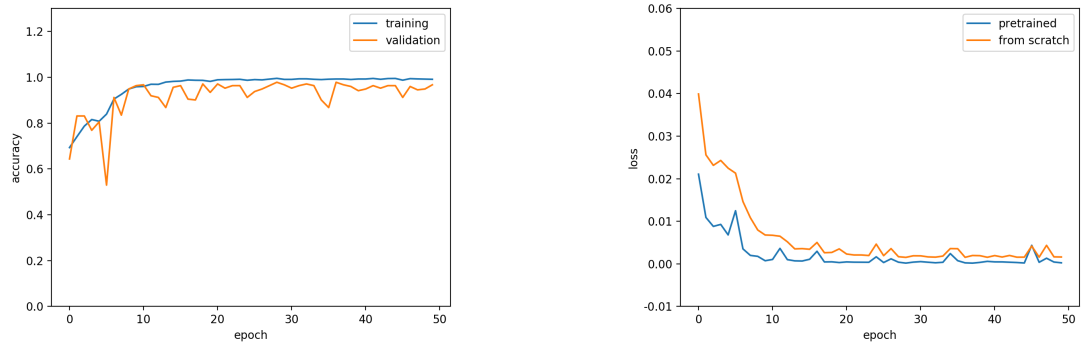
Training Set: Initially, we build a training set consisting of template graphs of varying sizes. The number of vertices ranges from 5 to 100. So, there are 384 (i.e., $(100 - 5 + 1) \times 4$) templates in total. To expand the training set, we use these templates as seeds. As shown in Figure 3, their variants can be created by running the following operations:

- *clique*: randomly remove existing edges (*Q1*) or add vertices on the periphery (*Q2*);
- *egocentric*: randomly connect existing alter vertices (*E1*) or add vertices on the periphery (*E2*);
- *loop*: randomly add edges between existing vertices (*R1*) or add vertices on the periphery (*R2*);
- *chain*: add vertices on the periphery and connect them to non-endpoint vertices (*C1*).

To avoid a switch of pattern, the number of nodes or edges to be modified by each operation is limited. To be specific, we set the number to be less than 10% than that of the corresponding template graph. These operations are executed

multiple times until we get 2000 samples of graph data for each type of pattern. The layout images are then generated by using the D3 library [10]. Table 1 contains a set of samples retrieved from our training set. All sample images are in *black-white* color scheme. To make a distinction between template and variant graphs, we use *black* and *steelblue* colors in the table, and each template is followed by a variant based on the same number of nodes.

We train the classification model on the Pytorch platform [47]. The reason that we prefer pre-trained models over models trained from scratch lies in that we have a limited number of training samples. In case users need to add new patterns, we do not want them to spend much time on generating massive layout images. Pre-trained models can well capture the general features of images, and they accept a smaller scale of training samples. Figure 5a shows accuracy of the model trained from scratch in 50 epochs. We found that this model also achieve good performance in our case. The highest validation accuracy is 97.79%. We compare the training loss of models trained in both ways in Figure 5b. The pre-trained model converges faster.



(a) model accuracy (b) model loss

Fig. 5. Training performance of classification models: (a) the training and validation accuracy of the model trained from scratch; (b) the training loss of the pre-trained model and the one trained from scratch.

5 SMOOTH ANIMATIONS

We implement the smooth animations by reducing the visual differences between every two consecutive frames, and the specific methods are: first, preserving the consistency of graph layouts; second, optimizing the visual design of pattern glyphs. Before explaining the details, we give the mathematical definitions of terminologies that will be used later as follows:

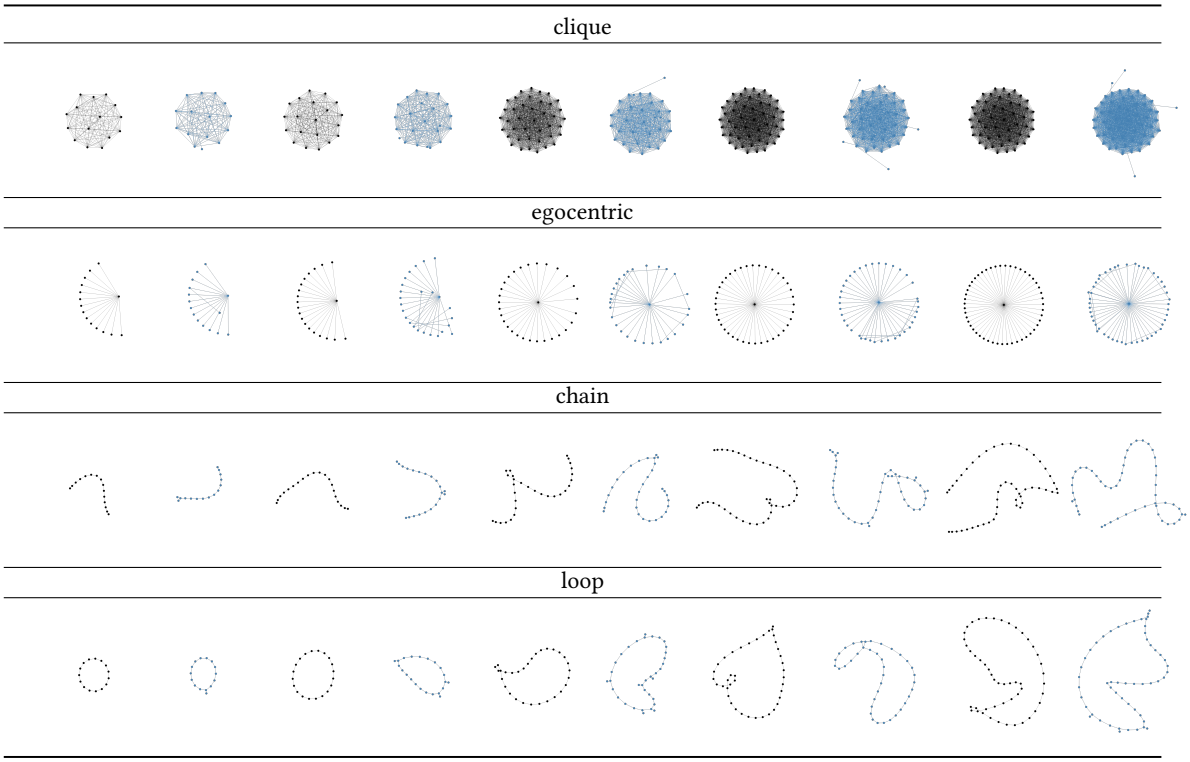
Definition 5.1. ***Time-Varying Graph:*** $G_T = \{G_1, G_2, \ldots, G_t\}$, where t is the number of time steps. For $1 \leq i \leq t$, $G_i = (V_i, E_i)$ represents the snapshot at time i. V_i and E_i are the set of vertices and edges, respectively.

Definition 5.2. ***Super Graph:*** $SG = (V_S, E_S)$, where $V_S = V_1 \cup V_2 \cup \cdots \cup V_t$ and $E_S = E_1 \cup E_2 \cup \cdots \cup E_t$.

Definition 5.3. ***Super Community:*** $SC = \{C_1, C_2, \ldots, C_k\}$ is the set of communities detected from SG, where k is the number of communities.

Definition 5.4. ***Super Layout:*** SL is the force-directed layout of the induced graph of $C_1, C_2, \ldots, C_k$, consisting of the screen positions of nodes that represent super communities.

Table 1. Training samples of the four topological patterns. Template and variant samples are colored black and steelblue, respectively. From left to right, the number of nodes are 15, 20, 30, 40 and 50.



5.1 Stable Layout

SC and SL are the references for calculating the communities and the layout of $G_1, G_2, \ldots, G_t$. Figure 6 shows an example of the calculation. We firstly compare the set of vertices in SG and G_i, and they are denoted by V_s and V_i respectively. For each vertex in V_i, we extract its associated community and the layout position from SC and SL. Hence, there are two communities in G_i: $C_1^{'}$ and $C_2^{'}$. Also, $|C_1^{'}| = |C_1|$, $|C_2^{'}| < |C_2|$ and $|\cdot|$ denotes the number of inner vertices. Despite of the size difference between $C_2^{'}$ and C_2, they remain at the same position on the display. Hence, our method can produce animations that seldom suffer from abrupt visual changes, except that adjacent snapshots have substantially different membership of communities.

The community detection and the layout algorithms are only applied to SG and SC. The complexity of them is $O(|V_S|\,log(|V_S|))$ and $O(|V_S|\,log(|V_S|) + |V_S| + |E_S|)$, respectively. Then for each snapshot, the time cost of extraction is linear to the number of vertices.

5.2 Pattern Transformation

At each time step, we use the classification model to determine the topological patterns of communities. By incorporating with the glyph-based visualizations, diverse community structures are summarized by a few patterns. Consequently, it

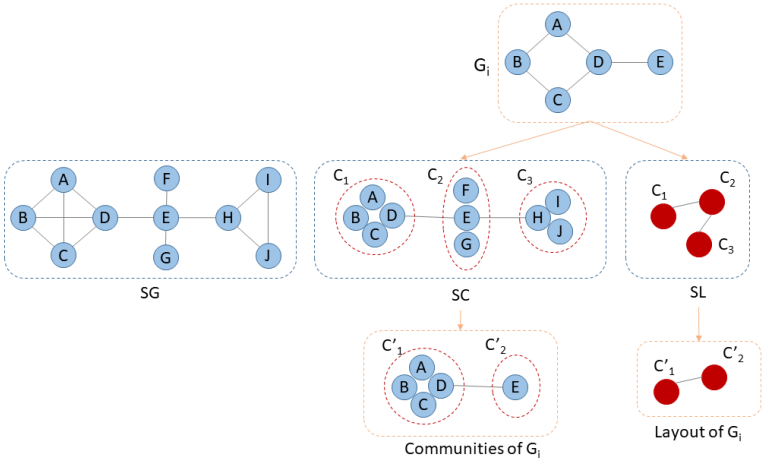


Fig. 6. For individual snapshots, we extract communities and their positions from SC and SL.

becomes easy to understand the overall distribution of the patterns. Besides, users can identify communities of similar structures and monitor the evolution of the whole graph.

We use five nodes as the base of a glyph drawing to achieve compactness and aesthetics. Provided that three nodes were used, it would be impossible to distinguish between *loop* and *clique*. Also, it is not impressive to demonstrate the dense connectivity of *clique* by four nodes. Using more than five nodes might add visual clutters and compromise the readability.

We improve the visual smoothness by controlling the transformation of glyphs. Glyphs are attached to the circles that denote communities. When the community pattern evolves, we do not want to get sharp visual changes. To achieve this goal, we bend the drawing of the *chain* pattern so as to make the four types of glyphs fit a pentagonal form. During the transformation, five peripheral nodes remain static and only edges appear or disappear.

As shown in Figure 7, if the *chain* pattern transforms to a *loop* pattern, only the bottom edge needs to be added. However, if the target pattern is *egocentric*, several edges need to be added and removed simultaneously, and the central node should be added. The transformations between other pairs of patterns are implemented similarly. The pentagonal appearance of pattern glyphs allows us to have a fluid visual perception when watching the animation.

6 VISUALIZATION

Our visual analytic system aims at assisting users to perform the following visual tasks that are related to exploring the structural changes of graphs:

- **Graph composition (T1)**: understand the graph structure at a high level. How many communities does the graph have at each time step? Are they densely connected or not?
- **Community importance (T2)**: measure the importance of a community by counting the number of vertices belonging to it and the total weight of its edges connected to other communities.

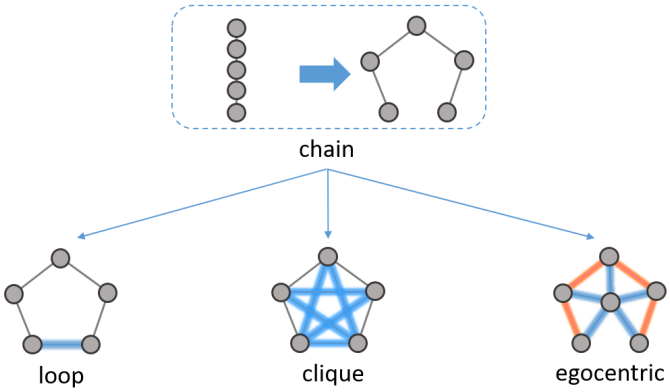


Fig. 7. An example of pattern transformation. The *chain* pattern smoothly transforms to other patterns. Emerging edges are marked by *blue* halos and *orange* halos represent disappearing edges.

- **Community duration (T3)**: at how many continuous time steps does a community appear? Identify stable communities that stay longer during the whole period.
- **Community evolution (T4)**: discover the graph evolution at a local level. How does the size and the structure of a community change temporally? What is the topological pattern that a community has most of the time? At what time points does a pattern transformation happen?
- **Topological distribution (T5)**: find communities with similar/different structures. What is the distribution of topological patterns in one snapshot? How does the distribution evolve temporally? Are the patterns evenly distributed, or some of them dominate?

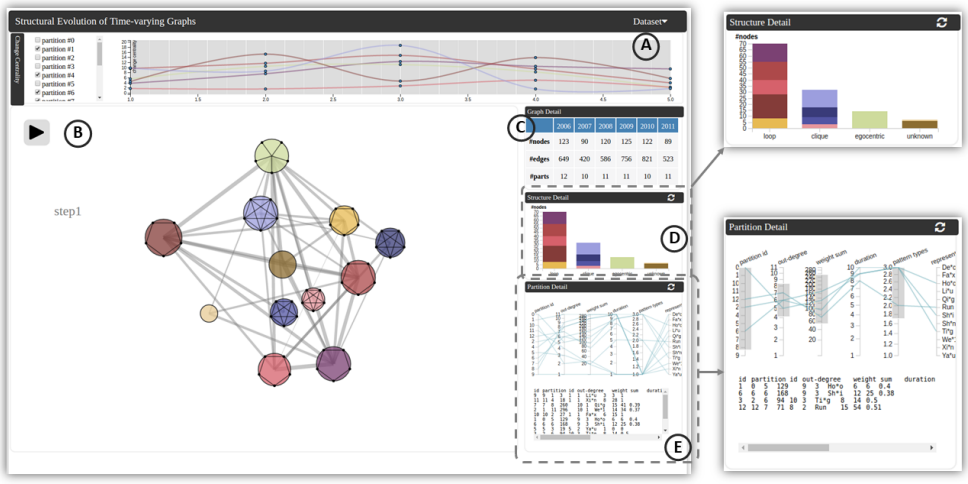


Fig. 8. The interface of the visual analytic system. The data in use is from the DBLP dataset. (A) represents the change centrality of communities, which quantitatively measures the structural changes between consecutive snapshots. (B) plays the smooth animation of time-varying graphs. (C) presents the temporal information of graph statistics. (D) shows the distribution of topological patterns at a specific time. (E) reveals the community details with multiple metrics.

Figure 8 shows the interface of our system that is implemented in JavaScript and D3. The animation is played in (B). Various statistic information of the graph and communities is provided to support visual analysis. Tables and charts in (D) and (E) can be updated when users select a certain time step. Specifically, users can access the distribution of the four topological patterns and the membership of communities in (D). In (E), multiple metrics are used to describe the characteristics of communities including the duration of communities' existence, the type of the topological patterns, the total weight of the inner connections, etc. As Parallel Coordinates (PC) are effective visualization tools for analyzing multivariate data, we use them to present these characteristics. PC consist of multiple parallel axes, each denoting a characteristic. A community is represented by a polygonal line that connects the vertices on axes. The vertices correspond to the characteristic values of communities. By inspecting individual axes, users can easily understand the value distribution of each characteristic. Besides, users can perform queries on different ranges of one or a combination of characteristics by brushing on the axes.

6.1 Variation Trend

We provide view (A) in Figure 8 to show the quantitative measurement of community variation. Despite that the fundamental changes of community structures can be captured by viewing the visual transformations of topological patterns, it is difficult to perceive the extent of variation. Besides, minor changes may not trigger the pattern transformation.

Therefore, we use a metric called *Change centrality* [19]. It measures the topological changes and the temporal changes of a vertex i simultaneously. From time t_1 to t_2, the value of the metric is calculated by:

$$r^n_{t_1,t_2}(i) = \frac{\left|N^n_{t_1}(i)\Delta N^n_{t_2}(i)\right|}{\left|N^n_{t_1}(i) \cup N^n_{t_2}(i)\right|}, \tag{5}$$

$$cc_{t_1,t_2}(i) = \frac{1}{2}\sum_{n=0}^{e_i}\frac{1}{2^{(n+1)}} r^n_{t_1,t_2}(i), \tag{6}$$

where e_i is the maximum graph distance from i to any other vertices; $r^n_{t_1,t_2}$ is the *change ratio* defined in Equation 5; $N^n_{t_1}(i)$ is the set of neighbours that are n steps away from i at time t_1; $|\cdot\Delta\cdot|$ calculates the number of nodes that are added to or removed from the n-step neighbors of vertex i; $|\cdot\cup\cdot|$ calculates the number of nodes in the union of the two sets. The change centrality $cc_{t_1,t_2}(i)$ of a vertex i is defined as a weighted summation of the changes of the adjacent neighbours, the adjacent neighbours of the latter and so on. The weight is defined as:

$$weight = \frac{1}{2^{(n+1)}}. \tag{7}$$

The weight decreases quickly with the increase of n, hence we make farther neighbours play a less important role in the metric by multiplying the weight. Besides, with the weight definition in Equation 8, the change centrality converges with its value in the range of $[0, 1]$. We then define the change centrality of a community as the summation of the changes of all its inner vertices:

$$cc_{t_1,t_2}(community) = \sum_{i=1}^{|C|} cc_{t_1,t_2}(i), \tag{8}$$

where $|C|$ is the number of vertices.

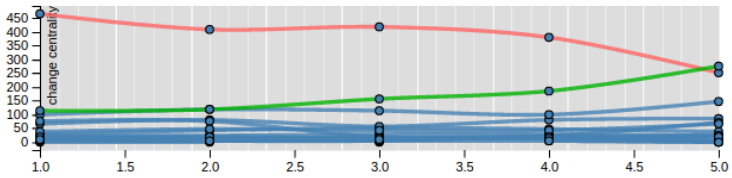


Fig. 9. Circles mark the change centrality values of communities. A curved line shows the changing trend of a community's change centrality. The data in use is from the Wikipedia dataset.

In Figure 9, the horizontal (x) and the vertical (y) axes denote *time* and *change centrality*, respectively. The centrality value at $x = t$ represents the extent of changes from t to $t + 1$. The larger the centrality value, the more changes occur to the entities of a community. We use curved lines to denote communities so that it would be convenient to observe and compare their changing trends. For example, the community highlighted by the red line in Figure 9 experiences much more changes than others. But, it tends to be stable as the centrality value is decreasing. The upgoing green line implies that the corresponding community is becoming more dynamic.

6.2 Community Evolution

It is difficult to compare the differences between frames in animations as they are displayed one by one on the screen. To alleviate this problem, we place two topological patterns of a community concentrically in a circle, one is for the previous time step, and the other is for the current step. As a result, the visualization remains compact, and users can directly perform visual comparisons. In order to enhance the attention retention, current patterns are placed in the outer circles and occupy larger space. The previous pattern is half the size of the current one. As shown in Figure 10, previous patterns locate in inner circles. It is worth noting that though *clique* and *egocentric* patterns are partially covered by the inner area, we can still recognize them.

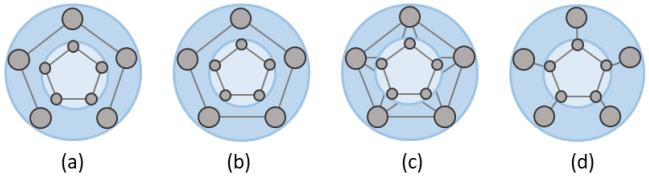


Fig. 10. Pattern comparisons of two consecutive time steps. Current patterns are placed in outer circles, and previous ones are in inner circles. In this example, current patterns are (a) *chain*, (b) *loop*, (c) *clique* and (d) *egocentric*, and all previous patterns are *loop*.

Changes might happen to multiple communities at the same time. For example, new communities appear, and old ones disappear. Accordingly, graph nodes that represent them need to be added or removed from the screen. For a stable community that exists at continuous time steps, we always make the corresponding node visible. However, we still need to present other types of dynamics including expansion, contraction, and pattern transformations. To discriminate these dynamics, we map them to the visual encodings listed in Table 2.

Generally, there are three categories of changes, the community existence, size, and structure. Different types of changes can happen to a community at the same time. If a community newly appears, it does not have any previous patterns. We regard this as an increase in size. Hence, we remove the inner circle of the glyph and adopt a thick solid

Table 2. Visual encodings of the community changes. Emerging communities have no previous patterns. We remove vanishing communities from the visualization. The size increase and decrease are illustrated by thick and thin borders of circles, respectively. The white filling of inner circles highlights the pattern transformation.

Existence	appear	
	disappear	remove from display
Size	increase	
	decrease	
Structure	unchanged	
	changed	

border. Conversely, if a community disappears, we delete it from the display. The size of a community equals to the number of the internal vertices. If the size is unchanged or increased, we use solid circle borders. Otherwise, we use thin borders instead. The *white* color is reserved for filling the inner circles of communities which undertake pattern transformations. If the topological pattern of a community is unchanged, then the outer and inner circles of the glyph are filled with the same color. In this work, we use different types of lines and colors to distinguish various changes. As the background color of circles might make the lines difficult to read, we can decrease the opacity of colors to increase the contrast. Though alternative shapes for nodes in node-link diagrams can be triangles, squares and so on, we choose circles so as to improve the aesthetics and compactness of visualizations by drawing them as circumcircles of pentagons.

7 CASE STUDIES

To evaluate the effectiveness of our method, we conduct two case studies: for the Wikipedia Edit History dataset [33, 37], we focus on measuring the time cost of classifying community structures and quantifying the visual differences between animation frames; for the DBLP dataset [63], we conduct visual tasks and gain insights from the glyph-based visualizations. All experiments were conducted on a desktop machine with 2.20GHz, Intel Core i5 CPU, 16.0 GB RAM and GTX 1060 GPU.

7.1 Wikipedia Edit History Data

We retrieved the complete Wikipedia revision records from January to December, 2007 and aggregated the data by month. We were interested in studying the relationship between different editors, e.g., did some of them ever collaborate? Two editors were connected if they had ever worked on the same article. Specifically, for each snapshot, top 1600 editors were abstracted as vertices of the network. We ranked editors by the number of different revisions that they had made.

The first row of Figure 11 contains the node-link diagrams of the original networks. They were generated by applying the Yifan Hu's layout algorithm [29] in Gephi [4] with the same configuration. We can see that severe visual clutters prevent us from recognising network changes in the core. The third row of Figure 11 shows the result of our method. We simplify the complex drawings of original networks by clustering closely related vertices into communities and substituting them by a single node. Since the structural information is hidden from view, we attach pattern glyphs for compensation. Table 3 lists the computation time of community detection, layout and classification. The classification

Table 3. The time cost (in seconds) of community detection, graph drawing and classification.

	detection	***drawing***	***classification***
Jan.	0.12	0.004	1.23
Feb.	0.13	0.007	1.87
Mar.	0.18	0.009	1.38
Apr.	0.20	0.007	1.15

time is the summation of the time taken to classify all communities at each time step. According to Figure 11, there are about 17 communities in each snapshot. The time cost of training the classification model is about 44 minutes and 43 seconds.

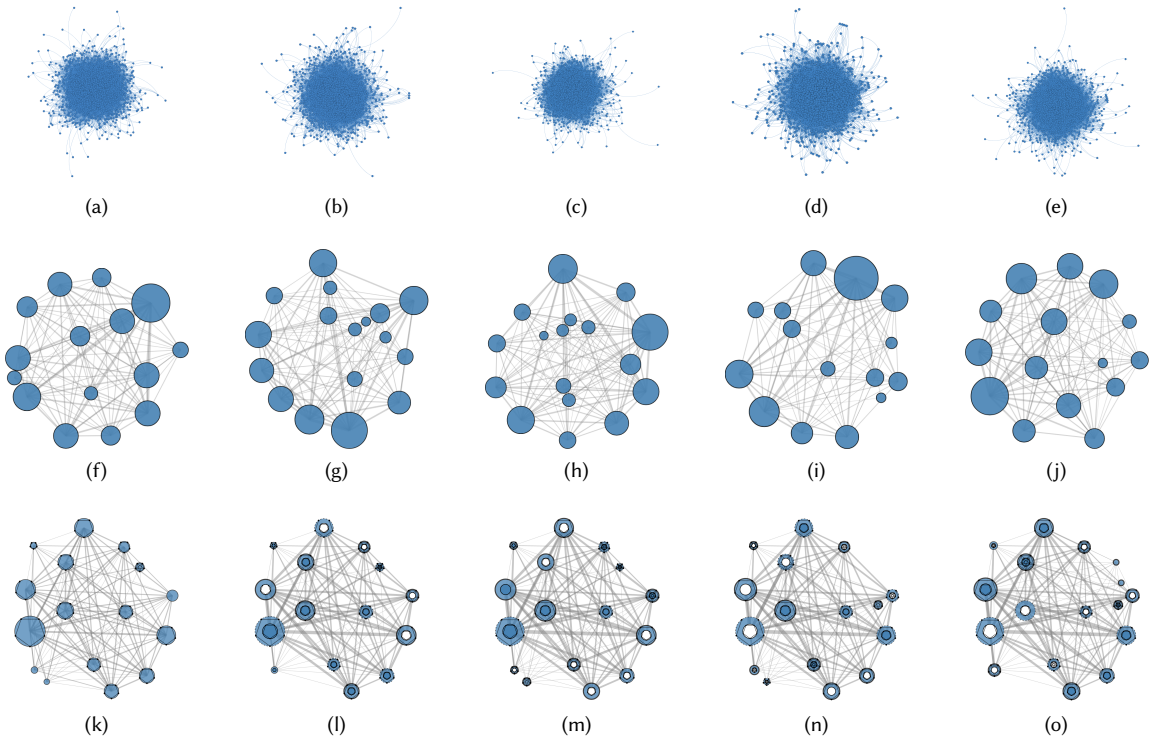


Fig. 11. Graph drawings of the Wikipedia data in five months, from January to May. The first row shows the original graphs. Nodes and links denote editors and their co-authorship, respectively. For the second and third row, nodes represent communities. Communities and their positions in the second row are computed independently at each time step. The third row presents the visualization results generated by our method.

We set the limit of the number of vertices in each community to 100. In case a community exceeds the limit, users are allowed to apply one of the sampling techniques: RV, RE and RW. The performance of these techniques varies on graphs of different structures. We suggest users to select the one that maintains the structural information to the most extent. Accordingly, we use Netsimile [7] to quantify the structural similarity between the sampling result and the

original graph. Figure 12 shows an example of a stable community. It contains about 150 vertices and firstly appears in February. From the line chart on the bottom, we can see that RV achieves the best result.

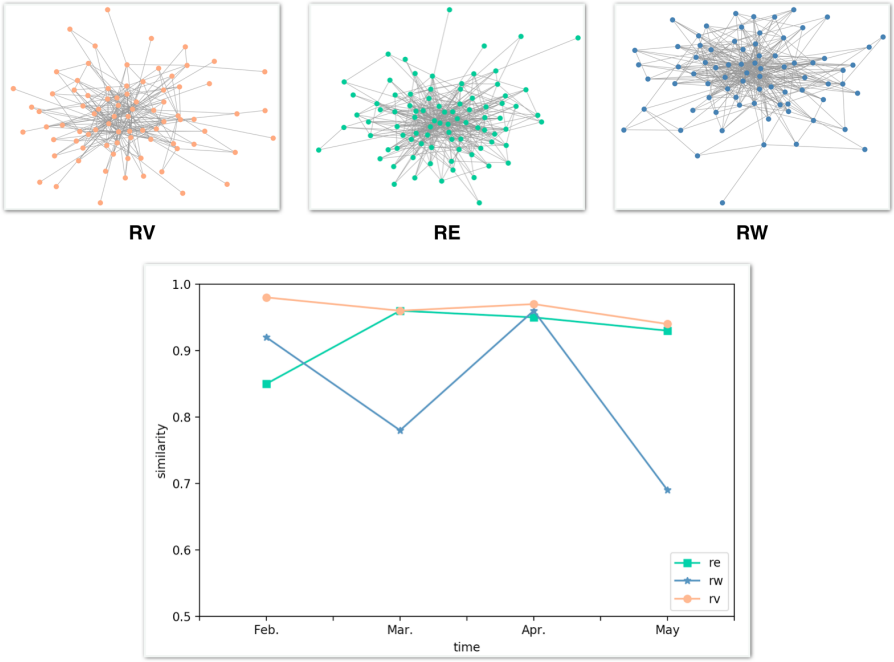


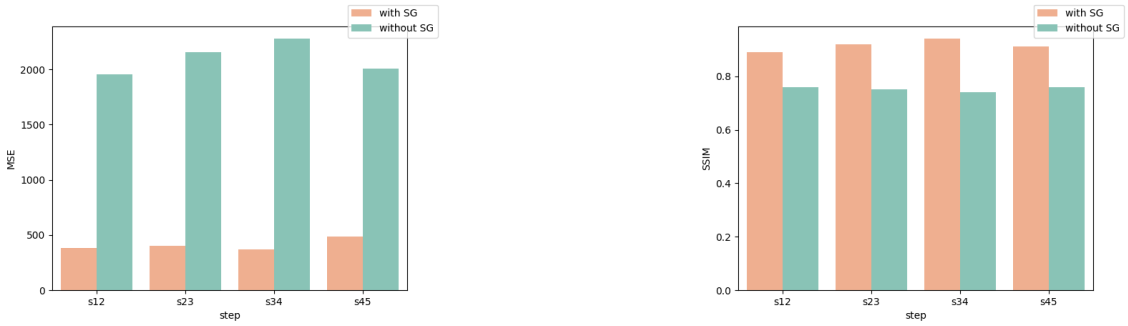
Fig. 12. The sampling results of a graph by RV, RE and RW. Their structural similarities to the original graph are measured by Netsimile [7]. A higher similarity score implies a better result.

To evaluate the visual smoothness of the animation, we compare every two consecutive frames by MSE (Mean Squared Error) and SSIM (Structural Similarity Index Metric). MSE compares two frames pixel by pixel. If it equals to 0, the two frames are the same. Higher MSE values indicate larger dissimilarities between frames. The SSIM value lies in $[-1, 1]$, and it reaches to 1 when two frames are the same. Figure 13 shows the metric values. $si(i + 1)$ indicates the comparison between frame i and frame $i + 1$. We compared our smooth animations (with SG) with the animation where community detection and graph layout were conducted separately at individual time steps (without SG). It is clear that our method produces less visual artifacts.

7.2 DBLP Data

The DBLP dataset contains the co-authorship history between researchers in computer science. We built an egocentric network by designating a highly active researcher as the *ego* and all other researchers who had ever cooperated with him as the *alters*. The relationship between alters themselves were also considered. The time span was from the year 2006 to 2015, and separate snapshots were created for each year. Totally, there were about 515 authors and 3268 co-authorship records involved in the 10 years. Our objective was to perform the visual tasks and explore significant patterns.

We merged snapshots of all time steps to build a *Super Graph*. Then the community detection and layout methods were conducted on the SG to obtain SC and SL. Figure 14 shows that there are 17 communities in SC, and they provide



(a) Mean Squared Error

(b) Structural Similarity Index Metric

Fig. 13. Measuring the visual differences between animation frames with and without calculating the SG, SC and SL.

references for partitioning individual snapshots. At each time step, an extracted community had an equal or smaller size than the corresponding community in SC. In Figure 14, we label identical communities with indices from 0 to 16. By comparing our SG-based layout and the layout obtained by independent calculations at each timestamp, we can see that, for the former, communities (e.g., *community 1*) stay at the same locations across the animation frames. However, the positions of an identical community change a lot for the latter. Therefore, our method is better at preserving the mental map.

Figure 15 shows the snapshots of four years. The first row consists of the force-directed drawings of the original graphs. The second row lists the animation frames generated by our method. Due to the stable membership and the screen positions of communities, the animation is quite smooth. Visual differences between frames are caused by the variation of the following properties of communities: the connectivity, the size and the transformation of topological patterns.

We evaluated the visualizations by performing the tasks proposed in Section 6. Based on the original graphs, we found that there was the least number of authors involved in the year 2007. Correspondingly, fewer communities and sparser connections are displayed in the second animation frame of Figure 15 (**T1**). We also noticed that the connectivity of communities was more complicated and denser in 2006. From 2006 to 2007, *community 9, 10, 11* disappeared, and a new *community 16* appeared. Most of the communities (i.e., *1, 2, 4, 6, 8*) experienced the size contraction (**T4**).

By comparing the node size, the number of outgoing connections and the width of links, *community 1, 4, 7, 8* were believed to be more important (**T2**). Besides, we found these communities were stable as their topological patterns were unchanged during the existence (**T3, T4**). In 2006, the inner circles of communities were empty because this year was assumed to be the beginning of time. Since *community 5* had a relatively small size, it was unpractical to classify the structure. We treated it as an anomaly. After looking into the statistics, we discovered that only one author belonged to this community. We deduced that this author had no close cooperation with others.

By inspecting the evolution of community structures, we found that *community 2, 6* were rather dynamic. Their patterns changed at every step. Besides, the patterns mainly switched between *chain* and *clique*, implying that the collaboration status between authors changed dramatically from sparse to dense. In fact, *chain* and *clique* were the two most frequently appeared patterns in this case. It is worth noting that the enduring *community 1* was always assigned the *egocentric* pattern (**T3, T4, T5**). So we looked into the inner authors and found that the *ego* node kept residing

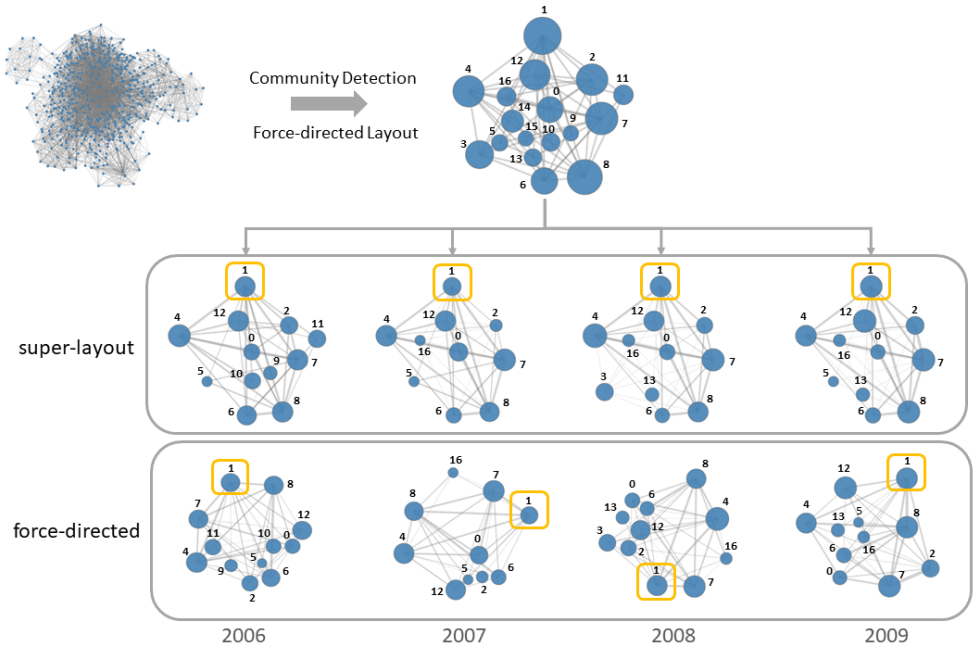


Fig. 14. Applying our method to the DBLP dataset. The layout of snapshots shown in the *super-layout* row is extracted from the SC and SL. Compared to the layout in the *force-directed* row, our method causes less movement of communities.

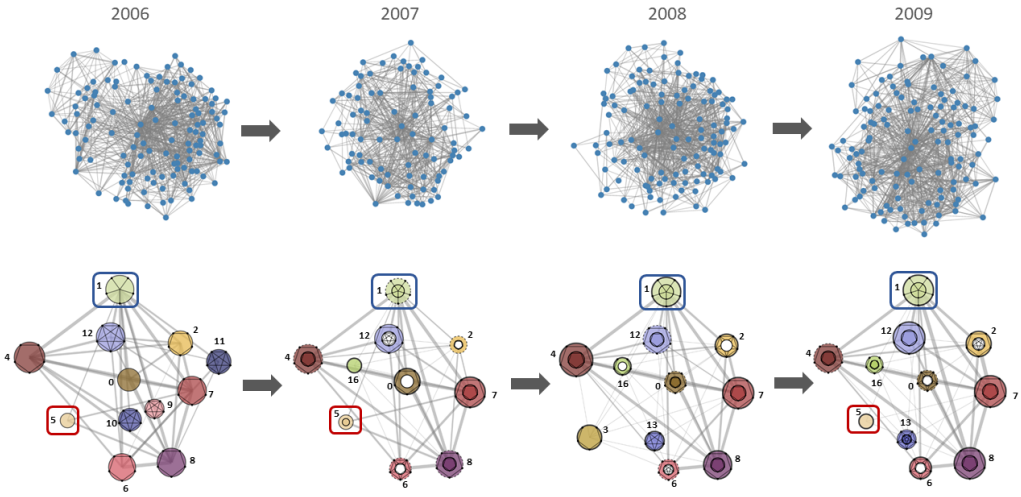


Fig. 15. Comparison of the original graph drawings (first row) and the simplified drawings with topological pattern glyphs (second row).

in this community. According to the characteristic of egocentric networks, it is not difficult to explain the pattern assignment of *community 1*.

We can further verify the aforementioned observations by investigating the statistics of communities. Basically, there were 10 to 12 communities in each year (**T1**). Figure 8 (D) shows that 5 communities had the *loop* pattern and 4 communities had the *clique* pattern in 2006. Only one community had the *egocentric* pattern (**T5**). By comparing Figure 8 (A) and Figure 9, we discovered that most of the communities in the DBLP dataset were more dynamic than those in the Wikipedia dataset, because the change centrality curves of the former are more fluctuant than those of the latter.

8 DISCUSSION

Although our method is effective at visualizing large time-varying graphs and displaying their structural evolution, there is still room for improvement. We provide multiple visual encodings to indicate the size variations and the structural transformations. In this way, users can conveniently identify the communities that undergo changes. Encodings can be combined to show the concurrency of different types of changes. However, when a large number of communities simultaneously appear on the screen, users will be overwhelmed in discriminating and interpreting all the changes. One potential solution is to control the number of visible communities, which can be achieved by the hierarchical clustering methods. Alternatively, we can use other graph partitioning methods such as METIS [30] which divides a graph into a designated number of components while minimizing the edge cuts.

The foundation of our method is constructing a Super Graph. We then implement smooth animations and reduce the time cost by conducting calculations at the global level. However, benefits are achieved at the cost of lossing the optimal community structures. We compute communities by extracting subsets from SC rather than by considering the unique connectivity at individual time steps. To obtain better community structures, we will consider applying the *Temporal Trade-off* methods and even the *Instant Optimal* methods in the future. But these methods produce a dynamic membership of communities, and the smoothness of visualizations would be compromised.

Our classification and glyph-based visualizations make complex structures comparable at a coarse level. Users can quickly locate the communities that match the pattern of a specific topology. During the classification procedure, communities are forced to be assigned one of the topological patterns pre-defined by users. However, a community probably matches a certain pattern with a low likelihood, because the structure cannot be actually described by any of the patterns. Therefore, we allow users to change or add patterns into the system in case the overall matching rate is low, and it only requires a re-training process of the classification model.

Currently, our method does not support users to visually explore the differences between communities that possess the same pattern. Therefore, we integrate the change centrality and the NetSimile method to quantify the differences between graph structures. The statistics are displayed along with the simplified visualizations. Thus, users can conduct both visual explorations and numeric investigations.

9 CONCLUSION

We build a visual system for presenting the structural evolution of large time-varying graphs. We achieve visual simplification by taking a single node of the node-link diagram to represent a community instead of an original vertex. Meanwhile, the visual scalability is extended with collapsed graph drawings. We also improve the consistency of the layout of snapshots by conducting fundamental calculations on the Super Graph. Our method makes the understanding of complex graph structures easier by classifying them into predefined topological patterns and illustrates the patterns by the transformable glyphs. In addition, the visual designs of the glyphs contribute to achieving smooth animations. For the future work, explorations into further graph classes can be performed by analyzing and increasing the feature

attributes of vertices. To that end, visual encodings will need to be optimized in order to deal with more complicated categories of community changes.

10 ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China under Grants (No. 61802128).

REFERENCES

[1] Gaurav Agarwal and David Kempe. 2008. Modularity-maximizing graph communities via mathematical programming. *The European Physical Journal B* 66, 3 (2008), 409–418.

[2] Benjamin Bach, Emmanuel Pietriga, and Jean-Daniel Fekete. 2014. GraphDiaries: Animated transitions and temporal navigation for dynamic networks. *IEEE Transactions on Visualization and Computer Graphics* 20, 5 (2014), 740–754.

[3] Shweta Bansal, Sanjukta Bhowmick, and Prashant Paymal. 2011. Fast community detection for dynamic complex networks. *Communications in Computer and Information Science* 116 (2011), 196–207.

[4] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. 2009. Gephi: An open source software for exploring and manipulating networks. In *Third International AAAI Conference on Weblogs and Social Media*.

[5] Fabian Beck, Michael Burch, Stephan Diehl, and Daniel Weiskopf. 2017. A taxonomy and survey of dynamic graph visualization. *Computer Graphics Forum* 36 (2017), 133–159.

[6] Mikhail Belkin and Partha Niyogi. 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems*. 585–591.

[7] Michele Berlingerio, Danai Koutra, Tina Eliassirad, and Christos Faloutsos. 2012. NetSimile: A scalable approach to size-independent network similarity. *Computer Science* 12, 1 (2012), 28(1–28).

[8] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008, 10 (2008), P10008.

[9] Béla Bollobás. 2013. *Modern Graph Theory*. Vol. 184. Springer Science & Business Media.

[10] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (2011), 2301–2309.

[11] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Gorke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. 2007. On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering* 20, 2 (2007), 172–188.

[12] Matthew Brehmer, Bongshin Lee, Benjamin Bach, Nathalie Henry Riche, and Tamara Munzner. 2016. Timelines revisited: A design space and considerations for expressive storytelling. *IEEE Transactions on Visualization and Computer Graphics (TVCG, Proceedings of InfoVis 2015)* 22, 1 (2016), 449–458.

[13] Chen Cai and Yusu Wang. 2018. A simple yet effective baseline for non-attributed graph classification. *arXiv preprint arXiv: 1811.03508* (2018).

[14] Inderjit Dhillon, Yuqiang Guan, and Brian Kulis. 2005. A fast kernel-based multilevel algorithm for graph clustering. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. 629–634.

[15] Inderjit Dhillon, Yuqiang Guan, and Brian Kulis. 2007. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29, 11 (2007), 1944–1957.

[16] Stephan Diehl, Carsten Görg, and Andreas Kerren. 2001. Preserving the mental map using foresighted layout. In *Data Visualization 2001*. Springer, 175–184.

[17] Cody Dunne and Ben Shneiderman. 2013. Motif simplification: Improving network visualization readability with fan, connector, and clique glyphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 3247–3256.

[18] Tim Dwyer. 2009. Scalable, versatile and simple constrained graph layout. *Computer Graphics Forum* 28, 3 (2009), 991–998.

[19] Paolo Federico, Jurgen Pfeffer, Wolfgang Aigner, Silvia Miksch, and Lukas Zenk. 2012. Visual analysis of dynamic networks using change centrality. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*. IEEE Computer Society, 179–183.

[20] Kun-Chuan Feng, Chaoli Wang, Han-Wei Shen, and Tong-Yee Lee. 2012. Coherent time-varying graph drawing with multifocus+ context interaction. *IEEE Transactions on Visualization and Computer Graphics* 18, 8 (2012), 1330–1342.

[21] Francesco Folino and Clara Pizzuti. 2014. An evolutionary multiobjective approach for community discovery in dynamic networks. *IEEE Transactions on Knowledge and Data Engineering* 26 (2014), 1838–1852.

[22] Yaniv Frishman and Ayellet Tal. 2004. Dynamic drawing of clustered graphs. In *IEEE Symposium on Information Visualization*. IEEE, 191–198.

[23] Laetitia Gauvin, André Panisson, and Ciro Cattuto. 2014. Detecting the community structure and activity patterns of temporal networks: a non-negative tensor factorization approach. *PloS one* 9, 1 (2014), e86028.

[24] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. 2007. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data* 1, 1 (2007), 4.

[25] Michelle Girvan and Mark EJ Newman. 2002. Community structure in social and biological networks. In *Proceedings of the National Academy of Sciences*, Vol. 99. National Acad Sciences, 7821–7826.

[26] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *ArXiv preprint* abs/1709.05584 (2017).

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[28] Bruce Hendrickson and Robert Leland. 1995. A multi-Level algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, Vol. 95. 1–14.

[29] Yifan Hu. 2005. Efficient, high-quality force-directed graph drawing. *Mathematica Journal* 10, 1 (2005), 37–71.

[30] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing* 20, 1 (1998), 359–392.

[31] Daniel A Keim. 2002. Information visualization and visual data mining. *IEEE transactions on Visualization and Computer Graphics* 8, 1 (2002), 1–8.

[32] Stephen G Kobourov. 2012. Spring embedders and force directed graph drawing algorithms. *arXiv preprint arXiv:1201.3011* (2012).

[33] Gueorgi Kossinets. 2012. Processed wikipedia edit history. http://snap.stanford.edu/data/bigdata/wikipedia08/enwiki-20080103

[34] Oh-Hyun Kwon, Tarik Crnovrsanin, and Kwan-Liu Ma. 2018. What would a graph look like in this layout? A machine learning approach to large graph visualization. *IEEE Transactions on Visualization and Computer Graphics* 24 (2018), 478–488.

[35] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. 2018. Graph classification using structural attention. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1666–1674.

[36] Robert Leland and Bruce Hendrickson. 1994. An empirical study of static load balancing algorithms. In *Proceedings of IEEE Scalable High Performance Computing Conference*. 682–685.

[37] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. 2010. Governance in social media: A case study of the wikipedia promotion process. In *Fourth International AAAI Conference on Weblogs and Social Media*.

[38] Chenhui Li, George Baciu, and Yunzhe Wang. 2015. ModulGraph: Modularity-based visualization of massive graphs. In *SIGGRAPH Asia 2015 Visualization in High Performance Computing*. ACM, 11.

[39] Qingsong Liu, Yifan Hu, Lei Shi, Xinzhu Mu, Yutao Zhang, and Jie Tang. 2015. EgoNetCloud: Event-based egocentric dynamic network visualization. In *Visual Analytics Science and Technology (VAST), 2015 IEEE Conference on*. IEEE, 65–72.

[40] Dijun Luo, Chris Ding, Heng Huang, and Feiping Nie. 2011. Consensus spectral clustering in near-linear time. In *2011 IEEE 27th International Conference on Data Engineering*. 1079–1090.

[41] Gary L. Miller, Shang Hua Teng, William Thurston, and Stephen A. Vavasis. 1998. Geometric separators for finite-element meshes. *SIAM Journal on Scientific Computing* 19 (1998), 364–386.

[42] Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak A. Rao, and Bruno Ribeiro. 2019. Relational pooling for graph representations. *ArXiv* abs/1903.02541 (2019).

[43] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang P. Liu, and Shantanu Jaiswal. 2017. Graph2Vec: Learning Distributed Representations of Graphs. *ArXiv* abs/1707.05005 (2017).

[44] Andrew Ng, Michael I. Jordan, and Yair Weiss. 2001. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*. 849–856.

[45] Dang Khoa Nguyen, Wei Luo, Tu Dinh Nguyen, Svetha Venkatesh, and Dinh Phung. 2018. Learning graph representation via frequent subgraphs. In *Proceedings of the 2018 SIAM International Conference on Data Mining*. 306–314.

[46] Quan Hoang Nguyen, Seok-Hee Hong, Peter Eades, and Amyra Meidiana. 2017. Proxy graph: Visual quality metrics of big graph sampling. *IEEE Transactions on Visualization and Computer Graphics* 23, 6 (2017), 1600–1611.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*. 8026–8037.

[48] Helen C. Purchase, Eve E. Hoggan, and Carsten Görg. 2006. How important is the "mental map"? - An empirical investigation of a dynamic graph layout algorithm. In *International Symposium on Graph Drawing*.

[49] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review* 76 3 Pt 2 (2007), 036106.

[50] Giulio Rossetti and Rémy Cazabet. 2018. Community discovery in dynamic networks: A survey. *Computing Surveys* 51, 2 (2018), 1–37.

[51] Giulio Rossetti, Luca Pappalardo, Dino Pedreschi, and Fosca Giannotti. 2016. Tiles: An online algorithm for community discovery in dynamic social networks. *Machine Learning* 106 (2016), 1213–1241.

[52] Martin Rosvall and Carl T. Bergstrom. 2008. Maps of random walks on complex networks reveal community structure. In *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 105. 1118–1123.

[53] Jiaxing Shang, Lianchen Liu, Feng Xie, Zhen Chen, Jiajia Miao, Xuelin Fang, and Cheng Wu. 2014. A real-Time detecting algorithm for tracking community structure of dynamic networks. *ArXiv preprint* abs/1407.2683 (2014).

[54] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. 2011. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research* 12, 9 (2011), 2539–2561.

[55] Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*. 488–495.

[56] Zhiqiang Tao, Hongfu Liu, Sheng Li, Zhengming Ding, and Yun Fu. 2019. Robust spectral ensemble clustering via rank minimization. *ACM Transactions on Knowledge Discovery from Data* 13, 1 (2019), 4.1–4.25.

[57] Corinna Vehlow, Fabian Beck, Patrick Auwärter, and Daniel Weiskopf. 2015. Visualizing the evolution of communities in dynamic graphs. *Computer Graphics Forum* 34, 1 (2015), 277–288.

[58] Corinna Vehlow, Fabian Beck, and Daniel Weiskopf. 2017. Visualizing group structures in graphs: A survey. *Computer Graphics Forum* 36, 6 (2017), 201–225.

[59] Scott White and Padhraic Smyth. 2005. A spectral clustering approach to finding communities in graph. In *Proceedings of the 2005 SIAM International Conference on Data Mining*. *SIAM*. 274–285.

[60] Jia Wu, Shirui Pan, Xingquan Zhu, Chengqi Zhang, and Philip S. Yu. 2018. Multiple structure-view learning for graph classification. *IEEE Transactions on Neural Networks and Learning Systems* 29, 7 (2018), 3236–3251.

[61] Yanhong Wu, Naveen Pitipornvivat, Jian Zhao, Sixiao Yang, Guowei Huang, and Huamin Qu. 2016. EgoSlider: Visual analysis of egocentric network evolution. *IEEE transactions on visualization and computer graphics* 22, 1 (2016), 260–269.

[62] Pinar Yanardag and S. V. N Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1365–1374.

[63] Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems* 42, 1 (2015), 181–213.

[64] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. GNNExplainer: Generating explanations for graph neural networks. In *Advances in Neural Information Processing Systems*, Vol. 32. 9244–9255.

[65] Anita Zakrzewska and David A. Bader. 2015. A dynamic algorithm for local community detection in graphs. In *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 559–564.

[66] Xinyi Zhang and Lihui Chen. 2019. Capsule graph neural network. In *International Conference on Learning Representations*.

[67] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2009. Graph clustering based on structural/attribute similarities. In *Proceedings of the VLDB Endowment*, Vol. 2. 718–729.