Visual Intelligence

REVIEW Open Access

Large graph layout optimization based on vision and computational efficiency: a survey

Shuhang Zhang[1], Ruihong Xu[1] and Yining Quan[1*]

Abstract

Graph layout can help users explore graph data intuitively. However, when handling large graph data volumes, the high time complexity of the layout algorithm and the overlap of visual elements usually lead to a significant decrease in analysis efficiency and user experience. Increasing computing speed and improving visual quality of large graph layouts are two key approaches to solving these problems. Previous surveys are mainly conducted from the aspects of specific graph type, layout techniques and layout evaluation, while seldom concentrating on layout optimization. The paper reviews the recent works on the optimization of the visual and computational efficiency of graphs, and establishes a taxonomy according to the stage when these methods are implemented: pre-layout, in-layout and post-layout. The pre-layout methods focus on graph data compression techniques, which involve graph filtering and graph aggregation. The in-layout approaches optimize the layout process from computing architecture and algorithms, where deep learning techniques are also included. Visual mapping and interactive layout adjustment are post-layout optimization techniques. Our survey reviews the current research on large graph layout optimization techniques in different stages of the layout design process, and presents possible research challenges and opportunities in the future.

Keywords: Large graph layout, Node-link diagram, Optimization, Visualization

1 Introduction

Large graph visual analysis has played a prominent role in various research fields. However, the generation of large graph layouts often faces the problems of high time complexity, stacking of visual elements, and cluttered visual effects. Although there is no clear definition of large graph data, with the development of big data, the scale of graph data shows an explosive growth, leading to traditional layout methods having difficulty generating an expected layout within a limited amount of time. Therefore, improving computing efficiency and optimizing visual effects are the most urgent needs in the current field of large graph visualization. Previous surveys about large graph visualization mainly focus on the layout algorithms and the usages of different kinds of optimization techniques, but in which stage these methods should be implemented is not mentioned, which is thought to be more beneficial for other relevant research and system design of large graph visual analysis.

Taxonomy. In this paper, the large graph layout process is divided into three stages: pre-layout, in-layout, and post-layout (see Table 1). According to the tasks of each stage, the corresponding optimization techniques are summarized from the perspective of computational efficiency and visual quality in each implementation stage, which is more instructive and practical for further research.

Computing efficiency. The methods of computing efficiency optimization are summarized from the aspects of graph data compression, computing architecture and algorithm, which are almost absent in previous studies. These techniques are mainly carried out in pre-layout and in-layout stage. The graph layout methods based on the deep learning model are included in Sect. 4.3. This field is still in its infancy and needs further in-depth research.

*Correspondence: ynquan@xidian.edu.cn
[1]School of Computer Science and Technology, Xidian University, Xi'an, China

Table 1 Categories of techniques and relative works for large graph visualization

Stage	Techniques		Articles
pre-layout	data compression	graph sampling	[1–10]
		graph aggregation	[11–16]
in-layout	architecture based		[17–23]
	algorithm based		[24–29]
	deep learning based		[30–34]
post-layout	visual mapping	edge bundling	[35–38]
		visual encoding	[16, 39, 40]
	interaction		[41–46]

Visual Quality. Visual mapping is an automatic method for optimizing visual quality, which mainly includes deformation and abstraction. However, manual interaction is necessary in some cases, which provides exploration and batch manipulation functions. These methods are usually applied in the post-layout stage, and some are combined with the techniques from the pre-layout and in-layout stages.

Although there are different methods for graph layout, our survey will mainly focus on the general node-link models, which are used in large graph visualization more often, while other layout models will also be mentioned. Only the optimization of static graph data is discussed, while dynamic graph data are not considered.

2 Relative works

2.1 Large graph visualization

A survey on large graph visual analysis technology conducted by Von et al. [47] summarized previous work from four perspectives: algorithm analysis, visual representation, user interaction and visual analysis. Although their paper designed evaluation indicators for graph layout quality including general criteria, dynamic criteria and aesthetic scalability, it is far from now and does not cover the latest related works. Hu et al. [48] reviewed layout algorithms and visualization techniques for large graphs. They provided a detailed summary and classification of previous graph layout algorithms and introduced various approaches to optimizing the visual effect, which involves graph abstraction, interactive exploration and visual mapping techniques. However, the optimization techniques in different fields have made much progress, and new challenges are raised by current studies.

2.2 Graph visualization

Beck et al. [49] summarized the visualization of dynamic graphs from the perspectives of animation, time and rendering methods, and designed evaluation indicators for the visualization of dynamic graphs. Vehlow et al. [50] studied the visualization of group structure in graphs, focusing on the study of the display coding of elements in the real graph group structure. Didimo et al. [51] studied graph layout techniques in non-planar spaces, discussed the constraints and limitations of graph visualization in non-planar spaces, and summarized the differences in the layout effects of different non-planar spaces. Schöttler et al. [52] studied the visualization- and interaction-related work of geospatial networks and discussed the trade-off between geographic map information and visualization readability from four dimensions. Burch et al. [53] put forward that the graph visualization was divided into graph interpretation, graph memory and graph creation, and new aesthetic indicators were established.

3 Pre-layout: graph data compression

The excessive size of data is the main cause for exceeding the time limit, poor visual effect, and low interaction quality. Reducing the size of the graph data is one of the most effective optimization approaches and is often used in research fields such as community detection and recommendation systems. This approach can reduce both the time cost and the number of elements in the view. Nevertheless, how to preserve the structural features of the graph is a new problem to be solved during the preprocessing of graph data. This process is called graph abstraction, where graph sampling and graph aggregation are the two main processing strategies. Taking the operations on nodes for examples, Fig. 1 describes the differences between the sampling and aggregation strategies.

3.1 Graph sampling

The graph sampling method filters the nodes or edges in the graph according to a certain pattern and extracts a subgraph as a representative sample of the original graph. The sampling pattern can be random, deterministic or partially random. A good sampling strategy can preserve the topological structure while reducing the size of the original graph. The current mainstream graph sampling techniques can be divided into three categories: vertex-based sampling, edge-based sampling, and travel-based sampling. Wu et al. [1] analyzed the preserving degree of visual features of node-link graphs by different sampling strategies from the perspective of visualization and provided a supplement for evaluation metrics.

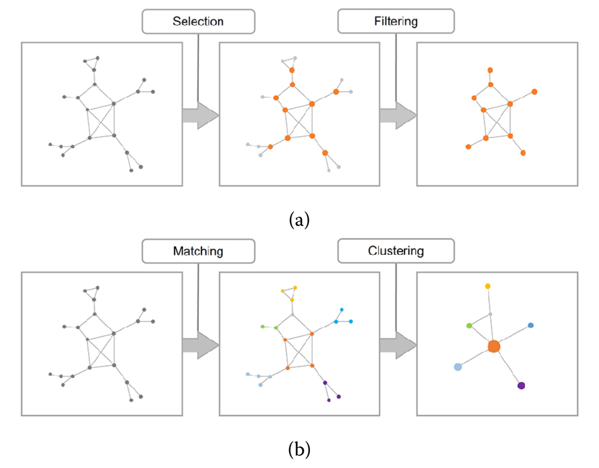


Figure 1 The graph sampling and graph aggregation processes, which take the operations on nodes for examples. (a) Graph sampling selects nodes from original graph and filters the unselected part. (b) Graph aggregation matches nodes and clusters each group of nodes into a weighted super-node

3.1.1 Vertex-based graph sampling

Selecting a set of nodes from a graph and preserving the connections between the selected nodes is called vertex-based graph sampling such as random node sampling (RN) [54]. This is a method of sampling among the nodes randomly, while other methods such as random PageRank node sampling (RPN) and random degree node sampling (RDN) [54] optimize the process by weighted random. Although the mechanism of these methods is simple, the result of sampling may vary in connectivity and density distribution of the graph due to randomness.

In recent years, an increasing number of studies have begun to pay attention to the association between sampling nodes and the topological structural features and semantic space in neighbors. In terms of structural features, Zhao et al. [2] noticed that several nodes accounting for a relatively small part of a graph play a crucial role in the connectivity and community structure characteristics of the graph. They classified these nodes into four types and defined the series of nodes as the minority structure of the graph. They also designed a detection algorithm for each type of minority structure.

In terms of semantic space, Cai et al. [3] proposed the adaptive UNI sampling (adpUNI) model on the basis of the original uniform sampling (UNI) [55]. The model considered the distribution and association of node attributes, divided nodes into multiple intervals, and adaptively adjusted the sampling rate to solve the problem of changes of graph connectivity caused by the uniform sampling probability of nodes in the UNI model. To improve the representativeness and connectivity of sampling, they proposed the adpUNI+N model to optimize the adpUNI model by adding the neighbor nodes to the sample set. Zhou et al. [4] developed a context-aware sampling method through graph representation learning. Dimensionality reduction was first performed to extract the two-dimensional representation of nodes from the Hilbert space. Next, an adaptive blue noise sampling strategy was introduced, which selects a subset of nodes from the vector space, and filters nodes within a certain range of selected nodes. Then, in the farther range, the nodes with better connectivity were selected as the target sampling nodes for the next round. This process would loop until all the nodes were filtered or selected.

3.1.2 Edge-based graph sampling

Similar to the vertex sampling-based method, the main idea of edge-based sampling is to select a subset from the edge set with nodes on its ends. The simplest and most classic method is also to sample in a random way as random edge sampling (RE) [54], although this method faces the same problem of graph connectivity and density distribution change as random node sampling. Based on the classical algorithm, total inductive edge sampling (TIES) [57] adopts a graph induction method to supplement the edges between the sampling nodes to restore the topology and connectivity of the graph.

Recent research on edge-based sampling has started to focus on the impact of edge metrics on graph sampling. Batjargal et al. [5] proposed edge metricity-based faster graph sparsification (EM-FGS) based on edge metrics. They defined a semimetric edge: with the given edge weight, if there are other alternative paths between the end nodes of an edge, the edge is called a semimetric edge. By continuously removing semimetric edges, the graph structure can be effectively simplified. The distributed graph sampling (DGS) model developed by Jaouadi et al. [6] selects the minimum centrality value of the nodes at both ends of the edge as the edge metric, which is used as the basis for edge removal. Based on the MapReduce framework, a distributed computing method is used to improve the computing speed. According to the experimental verification, the time complexity and space complexity of the DGS model are both $O(|E|)$, where $|E|$ is the scale of the edge set.

3.1.3 Traversal-based graph sampling

There are a variety of travel-based methods. The common sampling strategy of these methods is to start from a set of initial points or edges and continuously widen the sample size according to the current sampling information. The advantage of this strategy is that the connectivity of the graph is better preserved. Depth-first (DF) and breadth-first (BF) sampling are two basic traversal-based sampling methods [7]. The snowball algorithm (SB) [58] adopts a strategy similar to breadth-first, which selects a fixed proportion of neighbor nodes and visits the nodes in each iteration. The forest fire algorithm (FF) [56] selectively burns the connection according to the probability of forward burning, and there is a certain probability that the node at the other end will burn its own connection.

Random walk (RW) [8] is another representative idea of travel-based sampling. Although this method is relatively simple to implement, the major drawback is that it is easy to get stuck in local areas and keep looping. For this problem, many methods have been proposed for improvement. DRaWS [9] is the latest sampling method based on random walk, which combines clique and single node relationships. In this work, two random walk strategies are combined: one is based on the many-to-one structure to obtain the shortest sampling path, thereby reducing the computational cost; the other is based on the one-to-many structure so as to shorten the sampling path. This method can significantly reduce the computational cost while maintaining the original graph structure. The MH random walk algorithm (MHRW) [10] combines the principles of random walk and Metropolis–Hastings (MH) algorithm. The algorithm randomly selects a node u and any other node v in its neighbors, and selects a number p randomly between $(0, 1)$. If the value of p is lower than the degree ratio of node u to node v, then the algorithm will walk from node u to node v; otherwise it stays at node u. This method is suitable for generating small samples, and is more applicable in well-connected graphs.

3.2 Graph aggregation

Graph aggregation is also known as graph coarsening. Different from graph sampling, the idea of graph aggregation is to highlight the skeleton features of the graph by merging and folding vertices and edges. Graph aggregation is usually a multi-level strategy since the folding operation can be repeated several times. It can also achieve multi-level exploration by combining it with the visual interaction technology. Currently, most of the graph aggregation methods are implemented based on weight calculation and community detection algorithms. In the result, a weighted "super-node" is usually used to represent a single community. The series of results can be metaphorically encoded by means of visual coding.

Matching is the earliest idea of graph aggregation. Early research mainly includes random matching [59] and edge matching [60]. The random matching algorithm continuously selects two adjacent nodes randomly in the graph to fold, but this method may cause damage to the partitioning result. In contrast, the edge matching method performs better in the preservation of the community structure. One of the popular algorithms for edge matching is heavy edge matching [60], which randomly selects a node to make it aggregate with nodes that are not contracted with the largest edge weight in neighbors. This method can obtain a high-quality partition and preserve the topology of the graph. A more recent model, multi-level coarsening compact areas (MCCA) [11] developed by Rhouma et al. uses a greedy strategy. During each iteration, well-connected nodes are first selected for merging, and then the weights of nodes and edges are updated to minimize the graph shrinkage rate. When the shrinkage rate reaches a threshold, the algorithm will stop. Glantz et al. [12] introduced the concept of edge rating, which improves the selection strategy of shrinking edges. This method uses the conductivity of the edge (the number of shortest paths traversed by the edge) as the weight of the edge, indicating the importance of the edge to the connectivity. Then, the minimum spanning tree of the graph is calculated, the edges are classified according to the rating, and the conductivities of all the basic cuts of the spanning tree is calculated within the linear time complexity, which is used as the evaluation index of node aggregation.

Paying more attention to the modularity and community structure of graphs is the trend in recent years for graph aggregation. By aggregating nodes within a community, a clearer community structure and topological relationship can be presented. Combined with the visual encoding techniques (see Sect. 5.1), the gathered nodes can be represented by different symbols such as Motif [16]. Purohit et al. [13] noticed that the spectrum of graphs (the first eigenvalue of the adjacency matrix) characterizes the scattering properties of the network. If the first eigenvalue of two networks is similar, then they have high structural similarity. The COARSENET method proposed in this work minimizes the eigenvalue change as constraints, where the aggregation degree of graph data can reach 90% within the complexity of linearity, and will not excessively affect the spread characteristics of the graph. Ohsaka et al. [14] introduced the MaxInf method based on the concept of maximum influence. After the graph is divided into multiple communities, each community is regarded as a single super-node with the weights of nodes and edges updated. This method can effectively preserve the structural features of the graph. Heuer et al. [15] designed a method that considers local and global structures to deal with very large graphs. They used the maximum modularity algorithm to complete the pre-processing of community detection and then proposed a coarsening algorithm that treats each group as a super-node at the next level. According to the density of the graph, the generated edges are weighted using three methods.

4 In-layout: computing performance

Time complexity is a major problem in large graph layout calculations. When handling large graph data volumes, the computational efficiency of algorithms with high time complexity will be very low, which is unacceptable for users and researchers. Traditional layout algorithms, for example, force-directed layout [61] and dimensionality reduction layout [62] often have high time complexity and are not suitable for calculating layouts directly. Therefore, traditional methods need to be optimized from the perspective of computing architecture and time complexity of the algorithm. In addition, deep learning based methods have been implemented as a new approch with great potential in terms of computational efficiency and layout quality, which will be introduced in Sect. 4.3.

4.1 Architecture based

Efficiency optimization based on computing architecture relies on high-performance computing (HPC) [17]. HPC is a system in which software and hardware work together, including computing nodes, storage and file systems, network switching, cluster management and resource scheduling. In fact, high-performance computing has been used in the calculation of the large-scale scientific problems such as atmosphere, ocean currents, and gene sequencing, as well as big data storage and data mining. The currently used high-performance computing architectures mainly include distributed computing and GPU computing.

Distributed computing. This architecture reduces the total running time required for computing by dividing computation tasks into multiple parts, assigning them to multiple computing devices for processing, and finally merging and summarizing the computing results. Distributed graph layout computing usually divides a graph into multiple subgraphs based on the topology and modularity of the graph. This segmentation method can effectively reduce the intersection problem between communities. Multi-GiLA [18] is the first multilevel force-directed graph layout algorithm based on a vertex-centric computation paradigm. The algorithm is divided into four stages: pruning, segmentation, layout, and restoration. All nodes with degree 1 are removed in the pruning stage to reduce the amount of computation and re-insertion during the recovery phase, which can also minimize the additionally introduced crossing edges. The method divides the vertices into several subsets, creates balanced partitions according to the topology, and assigns each subset to an independent computational unit. The layout process is divided into three stages. First, the distributed Solar Merger algorithm is used to reduce the size of the graph and smooth the subgraph. Then, inspired by the FM3 [19] algorithm, the distributed Solar Placer algorithm places the new subgraph according to the location information of the previous subgraph. Finally, the layout of each subgraph is calculated by the GiLA algorithm. MuGDAD [20] is another distributed layout method that segments a graph at multiple levels by computing maximum independent sets (MIS). This method clusters nodes according to the neighborhood of the selected node. Once the layout of a graph at the higher level is computed, a distributed join operation is utilized to propagate node positions to the lower level. MuGDAD uses a batch synchronization parallel model to find MIS, which defines a super-step. In these super-steps, each node in the graph will execute a series of tasks in parallel, including sending the message to be read in the next super-step to another node, reading the message received from the previous super-step, and changing the current node state.

GPU Computing. This architecture is a very commonly used hardware accelerator in high-performance computing and is widely used in PCs, workstations, and large device clusters. Compared with distributed computing, GPUs usually have a large number of computing units, but these computing units are only suitable for executing some simple tasks. For graph layout, GPU is usually used to complete the computation of force models or matrices. Godiyal et al. [21] used a variant of fast multipole method (FMM) to estimate repulsion at long distances in force-directed graphs, constructing and traversing kd-trees on the GPU to implement computations on multipoles. Compared to quadtrees [24], the kd-tree is a density decomposition tree rather than a spatial decomposition tree, and thus, its running time does not suffer from distribution dependence. Gajdoš et al. [22] proposed a variant of the Fruchterman-Reingold (F-R) force-directed layout algorithm to accelerate the computation of repulsion by GPU. First, the positions of vertices are copied to texture memory and the index of vertices is allocated according to the Hilbert Z-order curve. Then, each block loads the position from texture memory to sData shared memory according to the index. Finally, each thread computes the repulsion force in the context of shared memory and stores the result into sForces shared memory, and then saves the result to global memory. Zellmann et al. [23] used NVIDIA's RT kernel to implement hierarchical tree traversal in hardware, mapping the graph layout problem to a ray tracing problem, and solved the problem through dedicated ray tracing hardware.

4.2 Algorithm based

Reducing the algorithm complexity of time and space is the core goal of algorithm efficiency optimization research on graph layouts. Earlier optimization algorithms, such as the Barnes-Hut algorithm [24] were mostly used for collision detection between multiple objects. This algorithm reduces the original $O(n^2)$ time complexity to $O(n \log n)$ and is often used to optimize the repulsion calculation and collision detection.

Most of the recent studies have shown that the time complexity of the graph layout is reduced to linear or sublinear time. The previous graph layout algorithm usually takes the global influence into consideration. However, for each node, the influence from the nodes in its neighborhood should be much greater than those far from; therefore, recent studies have focused more on reducing the scale of computation by sparsing and sampling, so as to further optimize the execution efficiency of the algorithm. Gove [25] utilized a repulsion calculation method based on random vertex sampling, in which the vertex set used in its calculation consists of two parts: the randomly selected part and the fixed-size part. For the first part, using the improved Fisher-Yates shuffle algorithm, a random set of vertices R is selected to calculate its repulsion on another set of vertices U, where $|U| = |V|^{\phi}$, $|R| = |V|^{(1-\phi)}$ and the time complexity of this part is $O(|V|)$. For the second part, each vertex u corresponds to a fixed-sized node set C_u and only calculates the repulsion with C_u in each iteration. The vertexes far from the vertex u in C_u will be replaced by other nearer vertexes. The time complexity for this part is also $O(|V|)$. Meidiana et al. [26] established a new framework for force computation with sublinear time complexity. The computation of this framework is divided into three steps: analysis, initialization and force calculation. The analysis step generates a breadth-first search (BFS) tree with the center of the graph as the root and uses spectral sparseness to sample the edges. In the initialization process, the radial tree drawing algorithm is used to obtain the initial layout of the graph. The force calculation is based on the random vertex sampling algorithm. In each iteration, the number of nodes used to calculate the repulsion is $|V|^{0.5}$ and other $|V|^{0.2}$ nodes are used as the reference for repulsion calculation, so the time complexity of a single iteration is $O(|V|^{0.7})$, which is reduced to sublinear time. Zhu et al. [27] developed the DRGraph algorithm based on the tsNET [28] algorithm, and optimized the calculation of dimensionality reduction through three schemes. First, the graph distance is approximated by a sparse distance matrix, and the graph distance is calculated by taking the shortest path distance between a node and its neighbors into consideration only to simplify the node similarity evaluation. Second, the matrix gradient is calculated based on a subset of nodes using a negative sampling technique. On this basis, finally, a new graph aggregation technique is used to reduce the initial dataset size to form a rough graph layout faster. Then in the iterative process, the network is continuously refined until the graph structure is completely restored.

Force-directing and dimension reduction layout often involve an iterative process, and each iteration will make the graph layout adjusted with a certain step size. In previous research, the step size is usually specified manually, but there may be two problems with the directly specified step size: one is that the step size is too large, causing the calculation results to oscillate or diverge; the other is that the step size is too small, which causes the algorithm to converge too slowly. Egorov et al. [29] conducted in-depth research on this problem. They took the force-oriented model as the object to adaptively adjust the step size during the iterative process using the Wolfe condition [63] as a constraint for optimization. Their work proves that the upper bound on the value of the step size depends only on the characteristics of the graph and is maintained throughout the optimization process.

4.3 Deep learning based

The deep learning model is a kind of algorithm model with great potential that can make computers imitate the behavior of human vision, hearing and thinking, and solve many complex pattern recognition problems. Benefiting from the parallel computing technology, the deep learning models often have high computing efficiency. For graph research, the deep learning model is usually known as the graph neural network (GNN) [64]. However, when these methods are applied to a large graph, the size of the graph might vary from thousands to millions and make the input size of spectral-based GNN unchangeable, which means that there is an upper limit for the input size of the graph. Setting this limitation with a large value may require much more random access memory (RAM) since the spatial complexity of these models is usually $O(n^2)$ or $O(n^3)$. Since then, spatial-based GNNs have been more commonly used in large graph research.

In recent years, some scholars have tried to use the deep learning model to complete the graph layout calculation and have achieved certain results. The training of the deep learning model needs to formulate its learning task, that is, the loss function part. The loss calculation methods commonly used in map layout can be divided into two categories: one is to define the learning task as "imitation", compare the training results with the existing layout or layout algorithm generation results, and take the quantitative similarity evaluation index as the loss; the other uses aesthetics to measure the loss, including stress, node occlusion, community overlap and other common aesthetic evaluation indicators of graph layout.

The "imitation" learning method has the characteristics of fast learning speed and high efficiency, and the training data can be directly generated using the existing graph layout algorithm. Kwon et al. [30] established a model of the "encoding-decoding" structure based on the VAEs [65] method. The encoder generates representations in the latent space corresponding to the training data, while the decoder calculates the original layout of the generated graph according to the representation to learn the existing graph layout mode. In this model, the distance matrix of nodes is used as the input of the loss function, and the Gromov Wasserstein (GW) distance with permutation invariance is used to measure the similarity of equivalent nodes of the same group of structures, and the variational loss function defined in SWAE is used for training. DeepDrawing designed by Wang et al. [31] is improved based on the

BiLSTM model. According to the topological structure of the graph, several additional connections are added on the basis of the linear sequence, as revealed in Fig. 2. These connections combine the output of the previous adjacent nodes with the output of the previous round as the hidden layer information to retain the edge dependency of the nodes. The learning task of DeepDrawing is to imitate the layout of existing algorithms. The loss function uses the Procrustes statistical method to quantitatively evaluate the layout similarity of the graph by taking the coordinate set of the nodes in the graph as the input.

The effectiveness of the aesthetic metric loss has been verified in recent years, and the images generated by the model trained by this type of loss function are more suitable for qualitative and quantitative aesthetic evaluation.

Wang et al. [32] proposed a graph neural network based deep learning framework DeepGD, which can generate graphic layouts that meet multiple aesthetic indicators at the same time. The loss function of the model includes stress, t-SNE, node occlusion and other aesthetic indicators. During training, the model uses the weighted sum of loss components derived from the corresponding aesthetic indicators as the multi-objective loss function and uses adaptive weights and two SoftAdapt multi-objective training strategies based on importance to determine the weight coefficients of each aesthetic indicator. The experimental results show that DeepGD is superior to the latest graph layout algorithm in all five aesthetic indicators. Tiezzi et al. [33] introduced the concept of Neural Aesthete into their study: to take any two edges as input, and return the probability of intersection between the edges to calculate the gradient of the loss function. The graph neural drawer (GND) model introduced in this work formulates the problem as a node-centric regression task. Each vertex of the graph can infer its coordinates in the two-dimensional plane according to the graph topology, the target layout and the loss function. Input the position features defined by the Laplace eigenvector and embed the graph into Euclidean space through spectral technology; thus it has uniqueness and distance preservation. The $(DNN)^2$ proposed by Giovannangeli et al. [34] refers to the ResNet and uses the spectral graph convolution method to complete the feature extraction of the graph structure. In this model, stress and Kullback-Leibler (KL) divergence are selected as loss function, and layout quality is captured according to topology. Although the model is spectral-based, the loss function is still available for large graphs.

5 Post-layout: visual optimization

The post-layout optimization technology mainly optimizes the visual effect of the graph layout. Although the graph layout algorithm can generate a node distribution in line with expectations, it cannot effectively solve the visual confusion problem caused by excessive edge crossing,

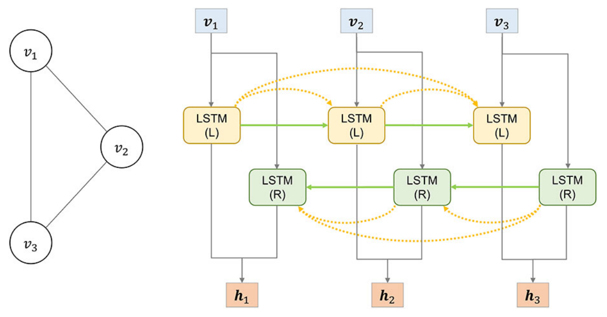


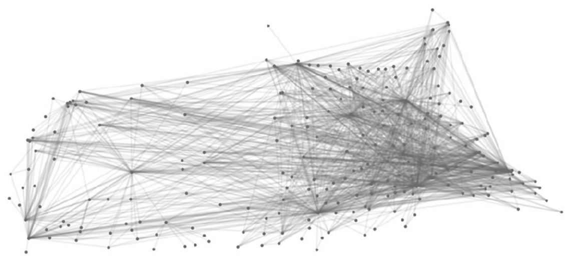
Figure 2 The message transmitting process of graph-LSTM. The nodes are ordered as (v1, v2, v3) and additional information transferring structures (the dot lines) are added between hidden layers according to the topology of the graph. [32]

and the semantic information related to nodes and edges cannot be represented. In addition, when the existing algorithms cannot stably output images that meet the expected visual effect, manual adjustment is needed. It is obviously unwise to adjust a large number of visual elements one by one, and it is more efficient to process data in batches through interaction.

5.1 Visual mapping

Visual mapping adjusts the style of the elements in the diagram according to the human visual principle to optimize the visual effect, making it easier for users to focus on the points of interest, and enriching the semantic meaning as shown in Fig. 3. For large graphs, the visual confusion caused by excessive edge crossing will bring great visual burden to users in the process of analysis and exploration. A common method to solve this problem is edge bundling technology. By bundling the edges in the graph, a clearer graph skeleton structure feature can be presented to users.

Traditional edge bundling algorithms are mostly implemented by layout algorithm evolution or based on edge features. Recently, some works have introduced relatively novel algorithms. Lhuillier et al. [35] developed an edge bundling algorithm FFTEB based on the fast Fourier transform. By transferring the bundling process from the space-time domain to the frequency domain, edge bundling can be implemented quickly on the GPU. Wu et al. [36] introduced the MLSEB algorithm based on the least squares approximation method, sampling the graph into point cloud data, and then using moving least squares to multiply projections to generate curvilinear bundlings. MLSEB reduces the distortion of local bundling by iteratively projecting each site onto its local regression curve, while generating bundling effects based on the convergence of its neighborhood density with other nearby sites. The edge-path bundling (EPB) algorithm proposed by Wallinger et al. [37] uses clustered edges of weighted paths as the bundling primitives to bundle each long edge in the graph to the


(a) Original layout

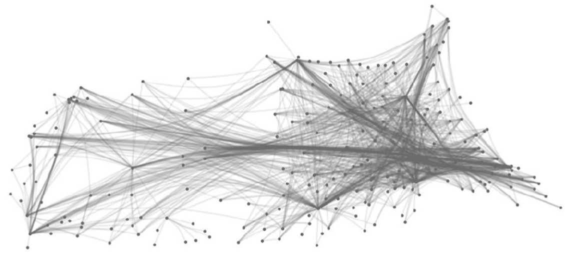
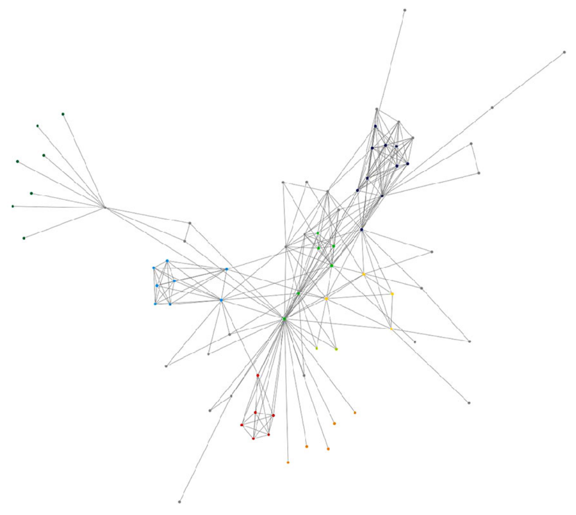

(b) Edge bundled layout

Figure 3 Graph layout with (a) and without (b) implementation of edge bundling techniques

shortest path existing between the endpoints of the edge. This method will not lead to the problem of independent edge ambiguity, because it adjusts the bundling level by shortest path distance, Euclidean distance, and a combination of the two. Sikansi et al. [38] designed a similarity-driven edge bundling algorithm SDEB, which is an improvement of HEB technology, creating a similarity hierarchy based on multi-level partitioning of data and grouping edges according to the similarity between nodes to guide bundling. The SDEB algorithm is performed in two steps. In the first step, a distance-preserving backbone structure is derived as a guide for bundling; in the second step, straight edges are bent toward this structure. A benefit of this backbone is the possibility of multi-scale visualization and exploration.

Another important role of visual channel mapping is visual metaphor, a process also known as visual encoding. Through visual coding, elements can be made to express richer semantic meaning. Visual coding needs to follow Gestalt theory to make the encoded image more consistent with people's general cognition. Van et al. [39] proposed a phrase network technology. By segmenting the text and extracting the text, the connection between the texts is constructed according to the syntax, and the text is directly used instead of the node. The thickness distribution of fonts and edges indicates the strength of the relationship between the frequency of text occurrence and the text. At the same time, the color depth is used as an additional code to indicate the ratio of node input and output. Dunne et al. [16] introduced the Motif simplification technology, as


(a) Original layout

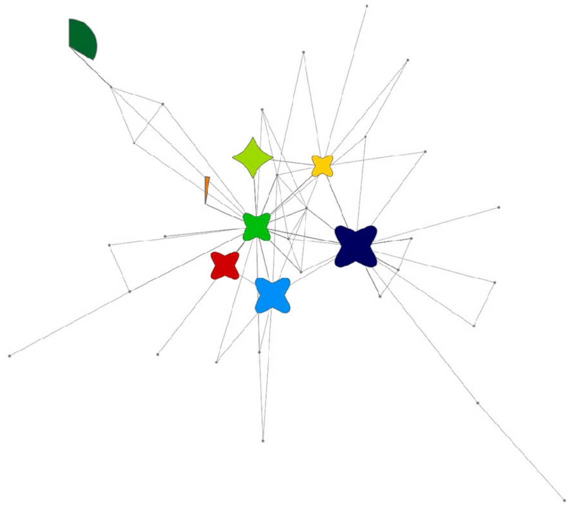

(b) Motif simplified layout

Figure 4 Graph layout before (a) and after (b) Motif simplification, which utilizes various glyphs to represent the common patterns of the structure [17]

shown in Fig. 4 which combines the graph aggregation algorithm to replace the common patterns of a large number of scattered nodes with compact and meaningful symbols. This technology greatly reduces the visual elements in the image and can present the main structural features of the graph more clearly, reducing the visual burden of users. At the same time, it is helpful for users to understand graph. Henry et al. [40] noticed that the node connections within the community in the graph structure were more compact, while the connections between the communities were sparser, and a hybrid tool NodeTrix was introduced. The tool employs an adjacency matrix to represent nodes in the same community and uses external edges to represent the connection between nodes in different communities, which can more clearly present the skeleton of the graph and improve the readability of the network.

5.2 Interactional visual optimization

A graph layout generated by graph layout algorithms cannot meet the expected aesthetic evaluation needs usually. A recent study conducted by Zhao et al. [66] shows that perceptions of a graph may vary from person to person owing to different levels of familiarity about the background story of the graph. Hence, interactional layout optimizing tools are urgently needed to explore and manipulate a graph layout in batches through visual interaction, and the layout of the elements in the graph can be quickly and efficiently adjusted to meet the preferences of users and assist users in completing the exploration of the graph.

Interactional exploration of graph element positions is a mainstream method. OnionGraph designed by Shi et al. [41] simplifies the graph structure by aggregating nodes through a hybrid method of node attributes and graph topology. Combined with the metaphor of "onion", the nodes containing node subsets are aggregated into multiple concentric circles to indicate how many abstract levels they contain and allow users to explore from top to bottom. Suh et al. [42] developed an interactive layout adjustment system with the guidance of the persistence homology (PH) [43]. (See Fig. 5.) The initial graph layout is constructed by the F-R force-directed algorithm, and each node is regarded as a separate component. The minimum spanning tree algorithm is used to find the edge set that connects all components with the least cost, and the reciprocal of the edge weight is used as the length of the barcode to indicate the PH value between components. Users can click or filter the persistent barcode according to the structural characteristics of the graph to control whether additional gravity or repulsion force is generated between the components to optimize the layout. Taurus introduced by Xue et al. [44] is a general framework which presents a unified force representation as quotient-based forces. This framework formulates most existing techniques as a combination of power functions of graph-theoretical and Euclidean distances, and hence the strengths and weaknesses of existing techniques can be compared and new methods can be explored. A universal augmented stochastic gradient descent is developed additionally for all layout techniques which can generate optimal graph layout results.

Altering the style of the visual elements interactively can also beautify the layout to meet the aesthetic needs of users. Wang et al. [45] proposed a user- and task-driven visual aggregation processing method based on the existing edge bundling algorithm, allowing users to explore different edge bundling technologies, and pointed out the areas where each edge-bundling technology provides desired results. The smoothness constraint defined by the Laplace operator is used to generate smooth and clear

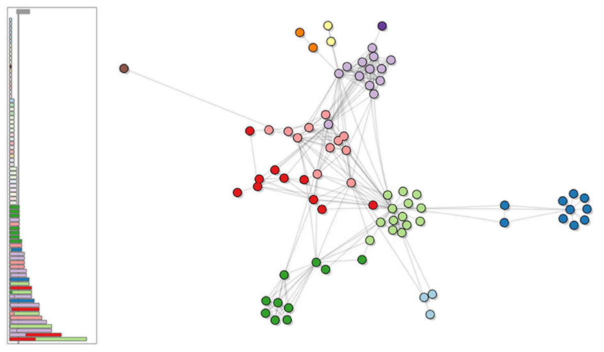


Figure 5 PH guided layout. Cluster nodes by manual adjustment of contractions and repulsions through interactions with PH bars [43].

transition results between specified regions and improve the visual quality of the graph skeleton structure. Finally, a post-processing rendering layer is provided to sort the edges and generate visual coding of width and color for the edges. Topology-aware space distortion (TASD) technique developed by Wang et al. [46] interactively distorts geometric space based on user attention and visual representation structure. TASD includes two mechanisms: recognition and distortion. The recognition mechanism is used to locate the region of interest in visualization, and the distortion mechanism allocates more screen space to the region of interest according to certain rules. This study implements a specific TASD tool, ZoomHalo, which allows users to interactively explore through mouse clicking or the gaze tracking technique.

6 Conclusion

This paper reviews the optimization techniques related to large graph layouts from the perspective of computational efficiency and vision. From the stages of pre-layout, in-layout, and post-layout, various optimization techniques and their application scenarios are discussed and the latest progress of related research fields in recent years is summarized. However, there is still room for further exploration on optimization for large graph visualization. Here, we provide the following summaries and suggest the following prospects for consideration in future research of large graph layout optimization.

Graph filter and graph aggregation. These techniques can effectively retain the graph structural features in compression. It is a trendency that an increasing number of correlational studies have started to pay attention to the structural topology and semantic representation of graphs, which can produce an abstract graph that is more representative. Although most of the current studies have formulated a series of coarsening rules, the randomness of the process will generate a different result each time through the same method, which may cause significant changes in the network topology and thus affect users' judgment. More deterministic compression algorithms for graph data are urgently needed.

Computational efficiency. Related studies have reduced the time complexity of graph layout algorithms to linear or even sublinear time in a single iteration and optimized the calculation process based on distributed, GPU and other high-performance computing architectures. However, insufficient attention has been given to the impact of iteration on graph layout calculation. Egorov et al. [29] only verified the constraints on the iteration step size proposed by them, but the optimization of the iteration process needs further investigation.

Deep learning graph drawing. Deep learning models have sufficient advantages on computational efficiency. Although some scholars have carried out research in this area, it has not been verified whether this kind of method is available for large graph data rendering, and its layout quality will be difficult to guarantee when users are exploiting unknown graph structures. Furthermore, it is challenging to demonstrate the interpretability of the deep learning models.

Visual mapping and interaction. Related research in this field is currently relatively mature. The layout enhanced by visual mapping can express the graph skeleton structure and graph semantic information more clearly and intuitively. Effective interaction design allows users to conduct an in-depth analysis and exploration of large graphs. In future research, we need to explore more visual mapping methods and design efficient interaction forms.

Acknowledgements
The authors thank Qi Liu for his material provision and helpful discussion on the paper.

Funding
This work was supported by National Natural Science Foundation of China (No. 61872432).

Abbreviations
adpUNI, adaptive UNI sampling; BF, breadth-first; BFS, breadth-first search; DF, depth-first; DGS, distributed graph sampling; EM-FGS, edge metricity-based faster graph sparsification; EPB, edge-path bundling; F-R, Fruchterman-Reingold; FF, forest fire algorithm; FMM, fast multipole method; GND, graph neural drawer; GNN, graph neural network; GW, Gromov Wasserstein; HPC, high-performance computing; KL, Kullback-Leibler; MCCA, Multi-level Coarsening Compact Areas; MH, Metropolis–Hastings; MHRW, MH random walk; MIS, maximum independent sets; PH, persistence homology; RE, random edge sampling; RN, random node sampling; RDN, random degree node sampling; RPN, random PageRank node sampling; RW, random walk; SB, snowball algorithm; TASD, topology-aware space distortion; TIES, total inductive edge sampling; UNI, uniform sampling.

Availability of data and materials
Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

Declarations

Competing interests
The authors declare no competing interests.

Author contributions
All authors contributed to the study conception and design. The idea for the article was designed by YQ and SZ. Material preparation, literature search and analysis were performed by SZ and RX. The first draft of the manuscript was written by SZ and all authors commented on previous versions of the manuscript. YQ revised the manuscript. All authors read and approved the final manuscript.

Received: 28 November 2022 Revised: 19 January 2023
Accepted: 23 March 2023 Published online: 17 July 2023

References

1. Wu, Y., Cao, N., Archambault, D., Shen, Q., Qu, H., & Cui, W. (2016). Evaluation of graph sampling: a visualization perspective. *IEEE Transactions on Visualization and Computer Graphics*, *23*(1), 401–410.
2. Zhao, Y., Jiang, H., Qin, Y., Xie, H., Wu, Y., Liu, S., et al. (2020). Preserving minority structures in graph sampling. *IEEE Transactions on Visualization and Computer Graphics*, *27*(2), 1698–1708.
3. Cai, G., Lu, G., Guo, J., Ling, C., & Li, R. (2020). Fast representative sampling in large-scale online social networks. *IEEE Access*, *8*, 77106–77119.
4. Zhou, Z., Shi, C., Shen, X., Cai, L., Wang, H., Liu, Y., et al. (2020). Context-aware sampling of large networks via graph representation learning. *IEEE Transactions on Visualization and Computer Graphics*, *27*(2), 1709–1719.
5. Batjargal, D., Khan, K. U., & Lee, Y.-K. (2019). EM-FGS: graph sparsification via faster semi-metric edges pruning. *Applied Intelligence*, *49*(10), 3731–3748.
6. Jaouadi, M., & Romdhane, L. B. (2021). A distributed model for sampling large scale social networks. *Expert Systems with Applications*, *186*, 115773.
7. Doerr, C., & Blenn, N. (2013). Metric convergence in social network sampling. In P. Hui, E. Miluzzo, & H. Haddadi (Eds.), *Proceedings of the 5th ACM workshop on hot topics in planet-scale measurement* (pp. 45–50). New York: ACM.
8. Li, R.-H., Yu, J. X., Qin, L., Mao, R., & Jin, T. (2015). On random walk based graph sampling. In J. Gehrke, W. Lehner, K. Shim, et al. (Eds.), *2015 IEEE 31st international conference on data engineering* (pp. 927–938). Los Alamitos: IEEE.
9. Zhang, L., Jiang, H., Wang, F., & Feng, D. (2020). DRaWS: a dual random-walk based sampling method to efficiently estimate distributions of degree and clique size over social networks. *Knowledge-Based Systems*, *198*, 105891.
10. Salamanos, N., Voudigari, E., & Yannakoudakis, E. J. (2017). Deterministic graph exploration for efficient graph sampling. *Social Network Analysis and Mining*, *7*, Article No. 24.
11. Rhouma, D., & Ben Romdhane, L. (2018). An efficient multilevel scheme for coarsening large scale social networks. *Applied Intelligence*, *48*(10), 3557–3576.
12. Glantz, R., Meyerhenke, H., & Schulz, C. (2014). Tree-based coarsening and partitioning of complex networks. In J. Gudmundsson & J. Katajainen (Eds.), *13th International symposium on experimental algorithms* (pp. 364–375). Berlin: Springer.
13. Purohit, M., Prakash, B. A., Kang, C., Zhang, Y., & Subrahmanian, V. S. (2014). Fast influence-based coarsening for large networks. In S. A. Macskassy, C. Perlich, J. Leskove, et al. (Eds.), *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 1296–1305). New York: ACM.
14. Ohsaka, N., Sonobe, T., Fujita, S., & Kawarabayashi, K. (2017). Coarsening massive influence networks for scalable diffusion analysis. In S. Salihoglu, W. Zhou, R. Chirkova, et al. (Eds.), *Proceedings of the 2017 ACM international conference on management of data* (pp. 635–650). New York: ACM.
15. Heuer, T., & Schlag, S. (2017). Improving coarsening schemes for hypergraph partitioning by exploiting community structure. In C. S. Iliopoulos, S. P. Pissis, S. J. Puglisi, & R. Raman (Eds.), *16th international symposium on experimental algorithms (SEA 2017)*. Schloss Dagstuhl: Schloss Dagstuhl - Leibniz Center for Informatics.
16. Dunne, C., & Shneiderman, B. (2013). Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In W. E. Mackay, S. A. Brewster, & S. Bødker (Eds.), *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 3247–3256). New York: ACM.
17. Qian, X. (2021). Graph processing and machine learning architectures with emerging memory technologies: a survey. *Science China-Information Sciences*, *64*, 160401.
18. Arleo, A., Didimo, W., Liotta, G., & Montecchiani, F. (2018). A distributed multilevel force-directed algorithm. *IEEE Transactions on Parallel and Distributed Systems*, *30*(4), 754–765.

19. Hachul, S., & Jünger, M. (2004). Drawing large graphs with a potential-field-based multilevel algorithm. In J. Pach (Ed.), *12th International symposium on graph drawing* (pp. 285–295). Berlin: Springer.
20. Hinge, A., Richer, G., & Auber, D. (2017). MuGDAD: multilevel graph drawing algorithm in a distributed architecture. In *Conference on computer graphics, visualization and computer vision* (pp. 1–8). IADIS.
21. Godiyal, A., Hoberock, J., Garland, M., & Hart, J. C. (2008). Rapid multipole graph drawing on the GPU. In I. G. Tollis & M. Patrignani (Eds.), *16th International symposium on graph drawing* (pp. 90–101). Berlin: Springer.
22. Gajdoš, P., Ježowicz, T., Uher, V., & Dohnálek, P. (2016). A parallel Fruchterman–Reingold algorithm optimized for fast visualization of large graphs and swarms of data. *Swarm and Evolutionary Computation*, *26*, 56–63.
23. Zellmann, S., Weier, M., & Wald, I. (2020). Accelerating force-directed graph drawing with RT cores. In *2020 IEEE visualization conference (VIS)* (pp. 96–100). Los Alamitos: IEEE.
24. Barnes, J., & Hut, P. (1986). A hierarchical O (N log N) force-calculation algorithm. *Nature*, *324*(6096), 446–449.
25. Gove, R. (2019). A random sampling O (n) force-calculation algorithm for graph layouts. *Computer Graphics Forum*, *38*(3), 739–751.
26. Meidiana, A., Hong, S.-H., Torkel, M., Cai, S., & Eades, P. (2020). Sublinear time force computation for big complex network visualization. *Computer Graphics Forum*, *39*(3), 579–591.
27. Zhu, M., Chen, W., Hu, Y., Hou, Y., Liu, L., & Zhang, K. (2020). DRgraph: an efficient graph layout algorithm for large-scale graphs by dimensionality reduction. *IEEE Transactions on Visualization and Computer Graphics*, *27*(2), 1666–1676.
28. Kruiger, J. F., Rauber, P. E., Martins, R. M., Kerren, A., Kobourov, S., & Telea, A. C. (2017). Graph layouts by t-SNE. *Computer Graphics Forum*, *36*(3), 283–294.
29. Egorov, D., & Bezgodov, A. (2015). Improved force-directed method of graph layout generation with adaptive step length. *Procedia Computer Science*, *66*, 689–696.
30. Kwon, O.-H., & Ma, K.-L. (2019). A deep generative model for graph layout. *IEEE Transactions on Visualization and Computer Graphics*, *26*(1), 665–675.
31. Wang, Y., Jin, Z., Wang, Q., Cui, W., Ma, T., & Qu, H. (2019). Deepdrawing: a deep learning approach to graph drawing. *IEEE Transactions on Visualization and Computer Graphics*, *26*(1), 676–686.
32. Wang, X., Yen, K., Hu, Y., & Shen, H.-W. (2021). DeepGD: a deep learning framework for graph drawing using GNN. *IEEE Computer Graphics and Applications*, *41*(5), 32–44.
33. Tiezzi, M., Ciravegna, G., & Gori, M. (2022). Graph neural networks for graph drawing. *IEEE Transactions on Neural Networks and Learning Systems*. Retrieved January 18, 2023, from https://ieeexplore.ieee.org/abstract/document/9810169.
34. Giovannangeli, L., Lalanne, F., Auber, D., Giot, R., & Bourqui, R. (2021). Deep neural network for DrawiNg networks, *(DNN)*^{2}. In H. C. Purchase & I. Rutter (Eds.), *International symposium on graph drawing and network visualization* (pp. 375–390). Berlin: Springer.
35. Lhuillier, A., Hurter, C., & Telea, A. (2017). FFTEB: edge bundling of huge graphs by the fast Fourier transform. In D. Weiskopf, Y. Wu, & T. Dwyer (Eds.), *2017 IEEE Pacific visualization symposium (PacificVis)* (pp. 190–199). Los Alamitos: IEEE.
36. Wu, J., Zeng, J., Zhu, F., & Yu, H. (2017). MLSEB: edge bundling using moving least squares approximation. In F. Frati & K.-L. Ma (Eds.), *25th International symposium on graph drawing and network visualization* (pp. 379–393). Berlin: Springer.
37. Wallinger, M., Archambault, D., Auber, D., Nöllenburg, M., & Peltonen, J. (2021). Edge-path bundling: a less ambiguous edge bundling approach. *IEEE Transactions on Visualization and Computer Graphics*, *28*(1), 313–323.
38. Sikansi, F., da Silva, R. R., Cantareira, G. D., Etemad, E., & Paulovich, F. V. (2020). Similarity-driven edge bundling: data-oriented clutter reduction in graphs layouts. *Algorithms*, *13*(11), 290.
39. Van Ham, F., Wattenberg, M., & Viégas, F. B. (2009). Mapping text with phrase nets. *IEEE Transactions on Visualization and Computer Graphics*, *15*(6), 1169–1176.
40. Henry, N., Fekete, J.-D., & McGuffin, M. J. (2007). NodeTrix: a hybrid visualization of social networks. *IEEE Transactions on Visualization and Computer Graphics*, *13*(6), 1302–1309.
41. Shi, L., Liao, Q., Tong, H., Hu, Y., Zhao, Y., & Lin, C. (2014). Hierarchical focus+ context heterogeneous network visualization. In I. Fujishiro, U. Brandes, H. Hagen, & S. Takahashi (Eds.), *2014 IEEE Pacific visualization symposium* (pp. 89–96). Los Alamitos: IEEE.
42. Suh, A., Hajij, M., Wang, B., Scheidegger, C., & Rosen, P. (2019). Persistent homology guided force-directed graph layouts. *IEEE Transactions on Visualization and Computer Graphics*, *26*(1), 697–707.
43. Petri, G., Scolamiero, M., Donato, I., & Vaccarino, F. (2013). Topological strata of weighted complex networks. *PLoS ONE*, *8*(6), e66506.
44. Xue, M., Wang, Z., Zhong, F., Wang, Y., Xu, M., Deussen, O., & Wang, Y. (2023). Taurus: towards a unified force representation and universal solver for graph layout. *IEEE Transactions on Visualization and Computer Graphics*, *29*(1), 886–895. https://doi.org/10.1109/TVCG.2022.3209371.
45. Wang, Y., Xue, M., Wang, Y., Yan, X., Chen, B., Fu, C.-W., & Hurter, C. (2019). Interactive structure-aware blending of diverse edge bundling visualizations. *IEEE Transactions on Visualization and Computer Graphics*, *26*(1), 687–696.
46. Wang, W., Badam, S. K., & Elmqvist, N. (2022). Topology-aware space distortion for structured visualization spaces. *Information Visualization*, *21*(2), 166–181.
47. Von Landesberger, T., Kuijper, A., Schreck, T., Kohlhammer, J., van Wijk, J. J., Fekete, J-D., & Fellner, D. W. (2011). Visual analysis of large graphs: state-of-the-art and future research challenges. *Computer Graphics Forum*, *30*(6), 1719–1749.
48. Hu, Y., & Shi, L. (2015). Visualizing large graphs. *Wiley Interdisciplinary Reviews: Computational Statistics*, *7*(2), 115–136.
49. Beck, F., Burch, M., Diehl, S., & Weiskopf, D. (2017). A taxonomy and survey of dynamic graph visualization. *Computer Graphics Forum*, *36*(1), 133–159.
50. Vehlow, C., Beck, F., & Weiskopf, D. (2017). Visualizing group structures in graphs: a survey. *Computer Graphics Forum*, *36*(6), 201–225.
51. Didimo, W., Liotta, G., & Montecchiani, F. (2019). A survey on graph drawing beyond planarity. *ACM Computing Surveys*, *52*(1), Article No. 4.
52. Schöttler, S., Yang, Y., Pfister, H., & Bach, B. (2021). Visualizing and interacting with geospatial networks: a survey and design space. *Computer Graphics Forum*, *40*(6), 5–33.
53. Burch, M., Huang, W., Wakefield, M., Purchase, H. C., Weiskopf, D., & Hua, J. (2020). The state of the art in empirical user evaluation of graph visualizations. *IEEE Access*, *9*, 4173–4198.
54. Leskovec, J., & Faloutsos, C. (2006). Sampling from large graphs. In T. Eliassi-Rad, L. H. Ungar, M. Craven, & D. Gunopulos (Eds.), *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 631–636). New York: ACM.
55. Gjoka, M., Kurant, M., Butts, C. T., & Markopoulou, A. (2010). Walking in Facebook: a case study of unbiased sampling of OSNs. In *The 29th IEEE International Conference on Computer Communication* (pp. 2498–2506). Los Alamitos: IEEE.
56. Lee, S. H., Kim, P.-J., & Jeong, H. (2006). Statistical properties of sampled networks. *Physical Review E*, *73*(1), 016102.
57. Ahmed, N., Neville, J., & Kompella, R. R. (2011). *Network sampling via edge-based node selection with graph induction*. Technical report, Purdue University.
58. Kurant, M., Markopoulou, A., & Thiran, P. (2011). Towards unbiased BFS sampling. *IEEE Journal on Selected Areas in Communications*, *29*(9), 1799–1809.
59. Hendrickson, B., Leland, R. W., et al. (1995). A multi-level algorithm for partitioning graphs. In S. Karin (Ed.), *Proceedings of supercomputing 1995* (pp. 28). New York: ACM.
60. Karypis, G., & Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, *20*(1), 359–392.
61. Fruchterman, T. M., & Reingold, E. M. (1991). Graph drawing by force-directed placement. *Software, Practice & Experience*, *21*(11), 1129–1164.
62. Harel, D., & Koren, Y. (2000). A fast multi-scale method for drawing large graphs. In J. Marks (Ed.), *International symposium on graph drawing* (pp. 183–196). Berlin: Springer.
63. Wolfe, P. (1969). Convergence conditions for ascent methods. *SIAM Review*, *11*(2), 226–235.
64. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(1), 4–24.
65. Kingma, D. P., & Welling, M. (2013). *Auto-encoding variational Bayes*. arXiv preprint arXiv:1312.6114.
66. Zhao, Y., Shi, J., Liu, J., Zhao, J., Zhou, F., Zhang, W., et al. (2022). Evaluating effects of background stories on graph perception. *IEEE Transactions on Visualization and Computer Graphics*, *28*(12), 4839–4854. https://doi.org/10.1109/TVCG.2021.3107297.