PPRviz: Effective and Efficient Graph Visualization based on Personalized PageRank

[Technical Report]

Paper ID: 405

ABSTRACT

Graph visualization is an important problem that finds applications in various domains, *e.g.*, social network analysis, traffic planning, and bioinformatics. Existing solutions for graph visualization, however, fail to scale to large graphs with millions of nodes, as they either provide inferior visualization results or incur significant computational cost. To address the deficiencies of prior works, we propose PPRviz, a multi-level visualization method for large graphs. Lying in the core of PPRviz is a new measure of graph node distance, PDist, that is specifically designed for visualization. In particular, PDist is formulated based on personalized PageRank, and it provides non-trivial theoretical guarantees for two well-adopted aesthetic measures. We present efficient algorithms for estimating PDist with provable accuracy and time complexity, while incurring small preprocessing costs. Extensive experiments show that PPRviz significantly outperforms 13 state-of-the-art competitors on 12 datasets in terms of both effectiveness and efficiency, and that PPRviz can provide interactive visualizations within one second on billion-edge graphs.

ACM Reference Format:

Paper ID: 405. 2021. PPRviz: Effective and Efficient Graph Visualization based on Personalized PageRank: [Technical Report]. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '22), June 12-17, 2022.* ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Graph visualization is an effective approach to provide users with intuitive understandings of the structures of graphs. It finds important applications in practice, such as biological structures exploration [2], social recommendation [39], and road network design [37]. This motivates a plethora of graph visualization solutions [1, 5–7, 18, 23, 24, 27, 30–32, 42, 45, 49, 50, 59, 60, 64, 71, 76, 80] and software [8, 10, 19, 25].

Given a graph $G = (V, E)$ with n nodes, the primary objective of existing solutions is to find two-dimensional coordinates for a set of nodes, such that the coordinates on the screen can reflect the related topological information of G. However, they are designed for small or medium-size graphs, and are unable to scale to massive graphs. In particular, most of the existing methods [18, 23, 24, 27, 30–32, 45, 49, 64, 76, 80] adopt a *single-level* approach that simultaneously visualizes all nodes on the screen. When n is large, this single-level approach results in unreadable visualizations due to the cluttering of nodes, and it incurs significant overheads as it requires computing the coordinates of a huge number of nodes. *Multi-level* methods [1, 6, 7, 11, 29, 42, 59, 60, 71, 74, 84] mitigate the above issue by recursively partitioning the input graph G into a hierarchy of supergraphs, where each supernode in the level-ℓ supergraph corresponds to a set of supernodes (or nodes) in the lower level. The user is allowed to navigate through the hierarchy and request a selected set of supernodes (or nodes) to be visualized. In other words, the multi-level method groups nodes into supernodes and provides a *partial* view of G, which avoids the cluttering in visualization and the significant cost of visualizing all nodes simultaneously.

However, existing multi-level methods suffer from two deficiencies. First, they often produce low-quality visualizations that either fail to illustrate node clusters clearly, or put nodes in overlapping positions that make them difficult to read. To explain, we note that existing methods typically require (i) computing a node distance matrix $\mathbf{\Delta} \in \mathbb{R}^{k \times k}$ that quantifies the distances among the k nodes to be visualized, and (ii) embedding each node into a two-dimensional Euclidean space, such that the node distances in $\mathbf{\Delta}$ are preserved as much as possible. Therefore, the quality of the visualization highly depends on how $\mathbf{\Delta}$ quantifies the distance between nodes. Unfortunately, the node distance measures adopted by existing methods are inferior in the sense that they fail to accurately capture the topology of the input graph. For example, the approaches adopted in [1, 6, 7, 29, 59, 84] derive node distances based only on direct connections among nodes, ignoring all indirect connections via multiple hops. Furthermore, the methods in [60, 74] quantify the distance of two nodes v_i and v_j based on the length of the shortest path between them, but disregard whether v_i and v_j are well-connected to each other (*e.g.*, whether there exists only one path connecting v_i to v_j, or a large number of short paths between them). As a consequence, the existing methods are unable to accurately represent topological information.

Second, existing multi-level methods still entail considerable computation overheads. In particular, to determine the distance between two supernodes $\mathcal{V}_i$ and $\mathcal{V}_j$, they require deriving either the distance between each pair of leaf nodes under $\mathcal{V}_i$ and $\mathcal{V}_j$ [60], or the coordinate of each such leaf node [59]. As a consequence, they are unable to generate visualizations in a reasonable time for interactive graph explorations.

To mitigate the deficiencies of existing solutions, this paper proposes PPRviz, an efficient algorithm for multi-level graph visualization. Lying in the core of PPRviz is a new node distance measure, referred to as PDist, that is specifically designed for improved visualization quality. PDist is formulated based on *personalized PageRank*

SIGMOD '22, June 12-17, 2022, Philadelphia, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

(PPR) [66], a classic metric that measures the proximity of graph nodes based on how well they are connected to each other via random walks. Compared with PPR, the formulation of PDist takes into account several issues unique in graph visualization, such as the symmetry of node distances, the effect of node degrees, and the discrepancy of edge lengths. We prove that PDist provides non-trivial worst-case guarantees for two widely-used aesthetic criteria, and we also extend PDist for multi-level visualization in such a way that it measures the distance between two supernodes based on the connectivity among the underlying nodes.

Computing PDist for supernodes in multi-level visualization, however, poses a significantly challenge in efficiency, due to the large number of underlying nodes that we need to examine. To address this challenge, we devise Tau-Push, an efficient algorithm for estimating PDist with provable worst-case accuracy and efficiency. Tau-Push borrows ideas from existing methods for PPR estimation, but differs from the latter in that it exploits the characteristics of PDist for significant reduction of computation costs.

We experimentally evaluate PPRviz against 13 state-of-the-art methods using 12 real-world graphs. Our results demonstrate that PPRviz provides better visualization readability, as it avoids negative artifacts such as overlapping nodes and edge distortions, and it significantly outperforms competitors in terms of two aesthetic metrics. In addition, the visualization latency of PPRviz is orders of magnitude smaller than that of competing methods, and it incurs only a small pre-computation overhead. In particular, PPRviz requires only 1 second to generate a supernode visualization from a graph with 41 million nodes and 3 billion edges, whereas none of the competing methods is able to return a visualization within 100 seconds.

To summarize, we make the following contributions:

- We propose PDist, a new node distance measure that not only captures topological information but also optimizes aesthetic metrics.
- We devise Tau-Push, an algorithm for efficient PDist computation that enables PPRviz to achieve sub-second latency even when visualizing large graphs.
- We conduct extensive experiments to demonstrate the superiority of PPRviz over the state-of-the-art methods in both effectiveness and efficiency.

2 PRELIMINARIES

In this section, we introduce the graph visualization problem and highlight key technical challenges for solving it in terms of both quality and efficiency.

Problem formulation. Let $G = (V, E)$ be a graph, where V is a set of n nodes and E is a set of m edges. Without loss of generality, we assume that G is a directed graph. Given a graph G, the graph visualization problem asks for an intelligible graph layout and a suitable interaction mechanism to visualize G. Existing solutions typically draw the input graph on a two-dimensional Euclidean space, representing nodes with dots and edges with lines [18, 24, 27, 31, 32, 45, 49, 59, 76]. Accordingly, the layout of a graph G is defined by a position matrix $\mathbf{X} \in \mathbb{R}^{n\times 2}$, where $\mathbf{X}[i] \in \mathbb{R}^2$ (*i.e.*, the i-th row of $\mathbf{X}$) records the coordinates of node $v_i \in V$ in the two-dimensional space, and $\|\mathbf{X}[i] - \mathbf{X}[j]\|$ is the distance between v_i and v_j in the layout.

Visualization quality. A high-quality graph layout should not only capture the topological information in the input graph but also provide good *readability* [12, 33, 81]. In particular, the position matrix $\mathbf{X}$ in the Euclidean space should accurately reflect the structure of G, in the sense that well-connected nodes should have a small distance to each other, while poorly-connected nodes should have large distances. For good readability, the layout should avoid negative visual artifacts such as nodes overlapping with each other or edges with drastically different lengths. In relation to this, there exist a number of aesthetic criteria in the literature that quantifies the readability of graph layouts. In this paper, we adopt two most commonly-used aesthetic metrics [14, 51, 65, 68, 69, 78], *i.e.*, *node distribution (ND)* and *uniform length coefficient variance (ULCV)*, as defined in the following.

Definition 2.1 (ND). For a position matrix $\mathbf{X}$, the node distribution is $\mathrm{ND}(\mathbf{X}) = \sum_{i<j} \frac{1}{||\mathbf{X}[i]-\mathbf{X}[j]||^2}$.

ND is the summation of the reciprocals of the squared distance between node pairs in the graph layout. A large ND score indicates the existence of visual clutter in the layout. In particular, overlapping nodes lead to an infinite ND score.

Definition 2.2 (ULCV). For a position matrix $\mathbf{X}$, let l_σ (resp. l_μ) be the standard deviation (resp. mean) of the edge lengths. The uniform length coefficient variance is $\mathrm{ULCV}(\mathbf{X}) = l_\sigma / l_\mu$.

ULCV measures the skewness in edge lengths. A large ULCV indicates that some edges are significantly longer or shorter than the others, in which case the layout tends to look distorted.

Multi-level visualization. Directly visualizing all nodes in a large graph usually results in a layout that resembles a giant hairball with almost zero readability, because of the sheer numbers of nodes and edges in the layout [33]. For this reason, existing work has proposed to interactively visualize large graphs in a *multi-level* manner to reduce the number of nodes in each visualization [1, 6, 7, 11, 71, 74]. In particular, the multi-level method consists of two phases, *i.e.*, (i) *preprocessing* and (ii) *interactive visualization*. In the preprocessing phase, a *supergraph* hierarchy is constructed to organize nodes of the graph G into a tree T, where (i) each leaf is a node in G, and (ii) each non-leaf node, referred to as a *supernode*, represents the union of its child nodes. For convenience, we say that each leaf node is at level-0, and that each non-leaf node is at level-$(\ell+1)$ if its children are at level-ℓ ($\ell \geq 0$). During the interactive visualization phase, the user can select any supernode $\mathcal{S}$ at level-$(\ell+1)$, and then ask for a visualization of the children of $\mathcal{S}$ (*i.e.*, each child node of $\mathcal{S}$ is visualized as a dot).

For such visualizations, the key issue is that we need to carefully decide the graph layout matrix for the child nodes of $\mathcal{S}$, especially when each child node is the supernode. In relation to this, a canonical approach is to measure the distance between two supernodes $\mathcal{V}_i$ and $\mathcal{V}_j$ based on the average distance between every node pair v_i and v_j, such that v_i (resp. v_j) is a leaf node under $\mathcal{V}_i$ (resp. $\mathcal{V}_j$). This approach, however, presents a significant challenge in terms of computation costs, since there may exist a large number of leaf

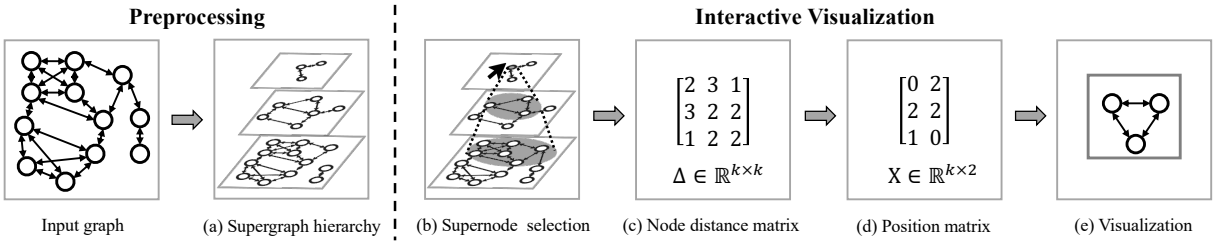


Figure 1: Procedure of PPRviz.

nodes under $\mathcal{V}_i$ and $\mathcal{V}_j$. As we show in Section 7.3, existing multi-level methods are unable to address this challenge, due to which they incur tremendous overheads when visualizing larges graphs with millions of nodes.

3 SOLUTION OVERVIEW

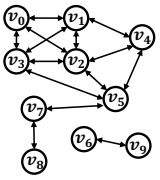
This section presents an overview of PPRviz, our solution for multi-level visualization. Figure 1 illustrates the workflow of PPRviz. In a nutshell, PPRviz processes the input graph G in three steps as follows.

- **Supergraph hierarchy construction**: Following prior works [1, 6, 7, 70, 74], PPRviz first constructs a supergraph hierarchy of the input graph, as shown in Figure 1(a). For this, we adopt a standard clustering algorithm, Louvain [16], to iteratively cluster nodes or supernodes at level-ℓ of the hierarchy into supernodes at level-$(\ell+1)$, under the constraint that each supernode $\mathcal{S}$ should have at most k children. This constraint ensures that the visualization of $\mathcal{S}$'s children does not contain an excessive number of nodes, which helps to avoid visual clutter. In our implementation of PPRviz, we set $k = 25$, as suggested in [44].
- **Node distance matrix computation**: During interactive visualization, a user selects a level-$(\ell+1)$ supernode $\mathcal{S}$ (marked by an arrow in Figure 1(b)) to explore. In response, PPRviz calculates a node distance matrix $\mathbf{\Delta} \in \mathbb{R}^{k\times k}$ for the k children of S (see Figure 1(c)). In the degenerated case when the supergraph hierarchy has only two levels (*i.e.*, when the input graph is small), PPRviz would compute a matrix $\mathbf{\Delta} \in \mathbb{R}^{n\times n}$ for visualizing the entire graph.
- **Position matrix embedding**: Given the node distance matrix $\mathbf{\Delta} \in \mathbb{R}^{k\times k}$, PPRviz converts it into a position matrix $\mathbf{X} \in \mathbb{R}^{k\times 2}$ (see Figure 1(d)), by solving the following optimization problem [32]:

$$\arg\min_{\mathbf{X}} L(\mathbf{X}|\mathbf{\Delta}) = \sum_{i<j} \left(1 - \frac{||\mathbf{X}[i]-\mathbf{X}[j]||}{\mathbf{\Delta}[i,j]}\right)^2. \quad (1)$$

  That is, it aims to ensure that the Euclidean distance $||\mathbf{X}[i] - \mathbf{X}[j]||$ derived from the position matrix $\mathbf{X}$ is close to the node distance $\mathbf{\Delta}[i,j]$. Towards this end, PPRviz employs the standard method for solving Eq. (1), namely, the *stress majorization* technique [32]. The time complexity of this method is $O(k^3)$ [32], which is insignificant when k is a small constant.

In what follows, we elaborate the node distance measure adopted in PPRviz and the algorithms used by PPRviz to compute the node distance matrix. Interested readers are referred to Appendix A for the details of PPRviz's adoption of Louvain [16] and stress majorization [32].



	PPR	PDist
(v_0, v_8)	0.01	3.73
(v_2, v_0)	0.11	0.92
(v_6, v_9)	0.44	1.12

Figure 2: PPR and PDist in a toy graph.

4 PPR-BASED NODE DISTANCE

This section presents PDist, our node distance measure for graph visualization. We first provide the formal definition and theoretical analysis of PDist in Section 4.1, and then presents a case study to demonstrate the visualization quality of PPRviz using PDist against existing methods.

4.1 PDist Definition

PDist is formulated based on *personalized PageRank (PPR)*, which is a measure for node proximity defined as follows. Given a directed graph $G = (V, E)$, two nodes $v_i, v_j \in V$, a *restart probability* α, the PPR $\pi(v_i, v_j)$ from v_i to v_j is defined as the probability that a *random walk with restart* (RWR) [79] originating from v_i would end at v_j. Specifically, an RWR starts from v_i, and at each step, it either (i) terminates at the current node with probability α, or (ii) with the remaining $1-\alpha$ probability, navigates to a random out-neighbor of the current node. Intuitively, if $\pi(v_i, v_j)$ is large, then there exists a relatively large number of paths from v_i to v_j, *i.e.*, v_i is well connected to v_j.

Based on PPR, we define our notion of PDist as follows.

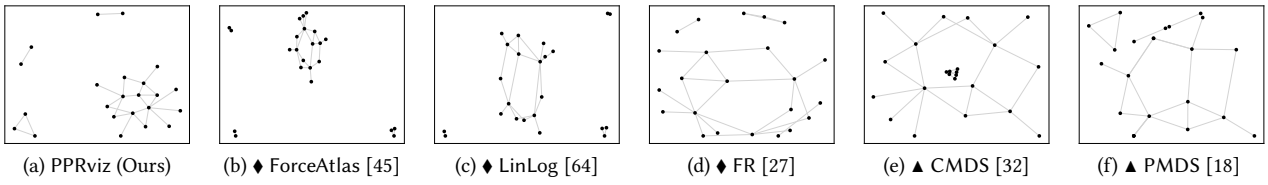
*Definition 4.1 (*PDist*).* Let $\mathbf{\Delta} \in \mathbb{R}^{n\times n}$ be the PDist matrix for all node pairs in a graph G and $\mathbf{\Delta}[i,j]$ be the PDist between nodes v_i and v_j. We define $\mathbf{\Delta}[i,j]$ as

$$\mathbf{\Delta}[i,j] = 1 - \log\big(\pi_d(v_i, v_j) + \pi_d(v_j, v_i)\big), \quad (2)$$

where $\pi_d(v_i, v_j) = \pi(v_i, v_j) \cdot d(v_i)$ is referred to as the *degree-normalized PPR* (DPPR) from v_i to v_j, and $d(v_i)$ is the out-degree of v_i.

The intuition of PDist is that if two nodes v_i and v_j have large PPR values, then $\Delta[i,j]$ tends to be small, *i.e.*, PDist could help us put well-connected nodes at close positions in a visualization. However, notice that the formulation of PDist also takes into account the out-degrees of v_i and v_j, for the following reason.

Consider Figure 2, which shows a graph with nodes $v_0, v_1, \ldots, v_9$, as well as the PPR and PDist values for node pairs (v_0, v_8), (v_2, v_0), and (v_6, v_9). Observe that $\pi(v_2, v_0) = 0.11 < \pi(v_6, v_9) = 0.44$, even though node pairs (v_2, v_0) and (v_6, v_9) are both directly connected



(a) PPRviz (Ours) (b) ♦ ForceAtlas [45] (c) ♦ LinLog [64] (d) ♦ FR [27] (e) ▲ CMDS [32] (f) ▲ PMDS [18]

(g) ★ GFactor [3] (h) ★ SDNE [85] (i) ★ LapEig [13] (j) ★ LLE [72] (k) ★ Node2vec [35] (l) SimRank [46]

Figure 3: Visualization results for the *TwEgo* graph: force-directed methods are marked with ♦; stress methods are marked with ▲; graph embedding methods are marked with ★.

via an edge. This indicates that the PPR between adjacent nodes in a graph could vary considerably. The reason is that PPR is designed to rank nodes based on their relative importance from the perspective of a source node, but it is unsuitable for comparing the strength of connections between nodes when different source nodes are considered [91]. As a consequence, directly transforming PPR values into node distances would lead to a large variance in edge lengths in graph visualization, thus resulting in a large ULCV. To alleviate this issue, in our formulation of PDist, we normalize each $\pi(v_i, v_j)$ by multiplying it with the out-degree of the source node v_i, as it is shown in previous work [91] that such a normalization tends to result in a metric that more accurately quantifies the strength of connections between nodes. For example, in Figure 2, the PDist of (v_2, v_0) is relatively close to that of (v_6, v_9), which is more consistent with the fact that v_2 and v_6 are direct neighbors of v_0 and v_9, respectively. Meanwhile, the PDist of (v_0, v_8) is large, which reflects the fact that v_0 is far away from v_8 in the input graph.

In addition, we formulate PDist based on $\pi_d(v_i, v_j) + \pi_d(v_j, v_i)$ (see Eq. (2)) instead of $\pi_d(v_i, v_j)$ or $\pi_d(v_j, v_i)$ alone, since $\pi_d(v_i, v_j) \neq \pi_d(v_j, v_i)$ in general, whereas graph visualizations require that the distance from v_i to v_j should be the same as that from v_j to v_i. Furthermore, PDist takes the inverse of the logarithm of DPPR, so that it can produce a smaller distance for the node pair with better connectivity.

In our implementation of PPRviz, we truncate $\Delta[i, j]$ to the range of $[2, 2\log n]$, for better visualization quality. In particular, we set the lower bound to 2 to avoid node overlapping. Meanwhile, the upper bound is set to $2\log n$ to avoid leaving excessive blank space in the visualization.

Bounds for aesthetic criteria. The following theorems[1] establish the worst-case upper bounds in terms of both ND and ULCV (see Definitions 2.1 and 2.2) when adopting PDist in visualization.

Theorem 4.2. *Given the* PDist *matrix* Δ *of graph* G*, suppose that* $||\mathbf{X}[i] - \mathbf{X}[j]|| = \Delta[i, j]$ *for* $v_i, v_j \in V$*, then* $ND(\mathbf{X}) \leq \frac{n(n-1)}{8}$.

Theorem 4.3. *Given the* PDist *matrix* Δ *of graph* G*, suppose that* $||\mathbf{X}[i] - \mathbf{X}[j]|| = \Delta[i, j]$ *for* $v_i, v_j \in V$ *and the restart probability* $\alpha \leq \frac{1}{2} - \sqrt{\frac{1}{4} - \frac{1}{2e}}$*, then* $ULCV(\mathbf{X}) \leq \frac{\left(\log \frac{1}{2\alpha(1-\alpha)} - 1\right)}{4}$.

[1] All proofs appear in Appendix A.

Note that the upper bound of the restart probability α in Theorem 4.3 (which is approximately 0.243) is not restrictive, as α is usually set to 0.15 or 0.2 [9, 47, 57, 86, 87, 90].

4.2 Comparison with Existing Methods

To demonstrate the effectiveness of PDist, we qualitatively compare our PPRviz using PDist against 11 existing single-level methods. They include (i) 3 force-directed methods (*i.e.*, ForceAtlas [45], LinLog [64] and FR [27]); (ii) 2 stress methods (*i.e.*, CMDS [32] and PMDS [18]); (iii) 5 graph embedding methods (*i.e.*, GFactor [3], SDNE [85], LapEig [13], LLE [72] and Node2vec [35]); as well as (iv) a variant of PPRviz using SimRank [46] in PDist. The visualization results[2] of a real-world graph *TwEgo* are displayed in Figure 3. We can observe that PPRviz yields a high-quality visualization result, which clearly organizes the graph into a well-connected cluster and three cliques. In contrast, the competitors suffer from various issues such as node overlapping and edge distortion, *i.e.*, large ND and ULCV scores, as detailedly explained in the following.

Force-directed methods model a graph as a force system, where adjacent nodes attract each other and all nodes repulse each other [21, 24, 27, 45, 59, 64]. The position matrix is derived by minimizing the composite forces in the entire system. In other words, only the direct links in the graph topology are considered. Figures 3(b) to 3(d) show the visualizations of ForceAtlas, LinLog and FR respectively, where the main cluster can be observed. However, compared with the main cluster, nodes in small cliques are placed very close and the edge lengths are small, which lead to large ND and ULCV scores. The problems of force-directed methods can be explained as: (i) nodes in a small cluster should be close and create enough attractive force to balance the repulsive force from the largest connected cluster; (ii) the edge length between nodes in the main cluster tends to be large, since there are more neighbors who only need provide a small amount of attractive force.

Stress methods utilize the shortest distance as the node distance to guide the node placement [18, 30–32, 49, 50, 60, 76, 80]. In particular, they embed the pairwise shortest distance matrix into a two-dimensional coordinate matrix by optimization techniques, *e.g.*, gradient descent [31] and stress majorization [32], such that

[2] More results are in Appendix A.

the node distances are preserved as much as possible. Compared with our PDist, the shortest distance only considers the shortest path from the source node to the target node, and consequently ignores the topological information of other long and intricate paths. Furthermore, the shortest distance can be unrecognizable, *e.g.*, for the adjacent nodes, which may incur node overlapping and drag down the readability of results. Figures 3(e) and 3(f) illustrate the visualizations of CMDS and PMDS on the *TwEgo* graph, which suffer from severe node overlapping and hence lead to an infinite ND score. In addition, the node clusters are not as clear as PPRviz.

Graph embedding methods have been adapted for graph visualization in [34]. Here, we follow [34, 67, 77] and set the embedding dimension to 2 to obtain the position matrix $\mathbf{X}$. Figures 3(g) to 3(k) report the visualization results of 5 widely-adopted graph embedding methods [3, 13, 35, 72, 85] on the *TwEgo* graph. It can be observed that graph embedding methods suffer from severe node overlapping and edge distortion. In addition, it is very difficult to observe the cluster structures. This is because graph embedding methods are designed for node classification or link prediction tasks, and thus tend to gather similar nodes as close as possible. We also generate high dimensional node embeddings to calculate node distance and then use Eq. (1) to derive matrix $\mathbf{X}$, but the results are similar.

SimRank [46] is also used to evaluate the connectivity of nodes in a graph. Particularly, SimRank measures the similarity of two nodes according to the similarity of their incoming neighbors. We transform SimRank into a node distance using Eq. (2) and embed the SimRank distance matrix into the position matrix using Eq. (1). The visualization result in Figure 3(l) shows that the largest connected cluster is extremely compressed while the 2-cliques have very long edges. This is because the node pairs in the largest connected cluster have very large SimRank scores, whereas those in 2-cliques have a zero SimRank score.

5 PPR-BASED SUPERNODE DISTANCE

During the interactive visualization, $O(k)$ level-ℓ supernodes in the user-specific level-$(\ell+1)$ supernode $\mathcal{S}$ are supposed to be placed on the screen. To measure supernode distances, we extend PDist to level-ℓ PDist in Section 5.1. Since the layout procedure only costs $O(k^3)$ (Section 3), the efficiency challenges mainly arise from the level-ℓ PDist computation, which is discussed in Section 5.2.

5.1 Level-ℓ PDist Definition

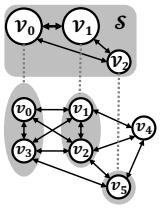
We first extend DPPR in Definition 4.1 to level-ℓ DPPR in Definition 5.1, and derive level-ℓ PDist by plugging level-ℓ DPPR into Eq. (2).

Definition 5.1 (Level-ℓ DPPR*).* For two level-ℓ supernodes $\mathcal{V}_i$ and $\mathcal{V}_j$, denote the set of leaf nodes in $\mathcal{V}_i$ as $F(\mathcal{V}_i)$, the level-ℓ DPPR $\pi_d(\mathcal{V}_i, \mathcal{V}_j)$ of $\mathcal{V}_j$ w.r.t. $\mathcal{V}_i$ is defined as

$$\pi_d(\mathcal{V}_i, \mathcal{V}_j) = \frac{\sum_{v_s \in F(\mathcal{V}_i), v_t \in F(\mathcal{V}_j)} \pi_d(v_s, v_t)}{|F(\mathcal{V}_i)| \cdot |F(\mathcal{V}_j)|}, \tag{3}$$

where $\pi_d(v_s, v_t)$ is DPPR of v_t w.r.t v_s.

Intuitively, the level-ℓ DPPR measures the connectivity from supernode $\mathcal{V}_i$ to $\mathcal{V}_j$ by taking the average of the DPPR values from the leaf nodes in $\mathcal{V}_i$ to those in $\mathcal{V}_j$. The idea that measure two supernodes by summarizing the structure of underlying leaf nodes is also adapted in [28, 60, 82].



	ℓ-DPPR	W-DPPR
$(\mathcal{V}_1, \mathcal{V}_0)$	0.56	0.74
$(\mathcal{V}_2, \mathcal{V}_0)$	0.39	0.69
$(\mathcal{V}_2, \mathcal{V}_1)$	0.50	0.69

Figure 4: A toy example of level-ℓ DPPR.

Compared with our level-ℓ DPPR, a simple and straightforward way is to take the level-ℓ children of supernode $\mathcal{S}$ as a weighted graph and compute the DPPR, called W-DPPR, on it. More precisely, the graph is constructed by treating each supernode $\mathcal{V}_i \in \mathcal{S}$ as a node and merging edges between supernodes $\mathcal{V}_i$ and $\mathcal{V}_j$ in $\mathcal{S}$ as a weighted edge, regardless of the micro-structure of each supernode as well as edges linking nodes outside $\mathcal{S}$. Although W-DPPR can be computed very efficiently as it requires only an $O(k^2)$ cost, it overlooks the partial structure of the graph, and thus, results in the sub-par quality of visualizations. To exemplify, we consider Figure 4, which shows a graph with nodes $v_0, v_1, \ldots, v_5$ and a level-2 supernode $\mathcal{S}$ with level-1 supernodes $\mathcal{V}_0, \mathcal{V}_1, \mathcal{V}_2$, as well as the level-$\ell$ DPPR (ℓ-DPPR in short) and W-DPPR values for supernode pairs $(\mathcal{V}_1, \mathcal{V}_0)$, $(\mathcal{V}_2, \mathcal{V}_0)$ and $(\mathcal{V}_2, \mathcal{V}_1)$. Intuitively, for the source supernode $\mathcal{V}_2$, $\mathcal{V}_1$ has better connectivity than $\mathcal{V}_0$, as $\mathcal{V}_2$ and $\mathcal{V}_1$ can reach each other via v_4 in the original graph. From the r.h.s table in Figure 4, we observe that the W-DPPR values of $(\mathcal{V}_2, \mathcal{V}_0)$ and $(\mathcal{V}_2, \mathcal{V}_1)$ are equal, which is counter-intuitive. In contrast, level-ℓ DPPR addresses this issue and accurately captures the structure of the original graph.

5.2 Efficiency Challenges

Approximate level-ℓ PDist. The exact computation of level-ℓ PDist is prohibitive as it requires computing exact PPR values, which incurs enormous costs for large graphs. Motivated by this, we define and calculate (θ, σ)-approximate level-ℓ PDist.

Definition 5.2 ((θ, σ)-approximate level-ℓ PDist*).* Let θ and σ be two constants, for any two supernodes $\mathcal{V}_i, \mathcal{V}_j \in \mathcal{S}$ and $\mathcal{V}_i \neq \mathcal{V}_j$, $\widehat{\Delta}[i, j]$ is a (θ, σ)-approximate level-ℓ PDist $\Delta[i, j]$ if it satisfies

- If $\Delta[i, j] < \sigma$, $|\Delta[i, j] - \widehat{\Delta}[i, j]| \leq \theta \cdot \Delta[i, j]$,
- If $\Delta[i, j] \geq \sigma$, $|\Delta[i, j] - \widehat{\Delta}[i, j]| \leq \theta \cdot \sigma$.

We show that the (θ, σ)-approximate level-ℓ PDist in Definition 5.2 can be obtained by computing the $(\epsilon, \delta)-$approximate level-ℓ DPPR, defined and proved as follows.

Definition 5.3 ($(\epsilon, \delta)-$approximate level-ℓ DPPR*).* Let ϵ and δ be two constants, for any two supernodes $\mathcal{V}_i, \mathcal{V}_j \in \mathcal{S}$ and $\mathcal{V}_i \neq \mathcal{V}_j$, $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ is an $(\epsilon, \delta)-$approximation of level-ℓ DPPR $\pi_d(\mathcal{V}_i, \mathcal{V}_j)$ if it satisfies the following conditions.

- If $\pi_d(\mathcal{V}_i, \mathcal{V}_j) < \delta$, $|\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) - \pi_d(\mathcal{V}_i, \mathcal{V}_j)| \leq \epsilon \cdot \delta$.
- If $\pi_d(\mathcal{V}_i, \mathcal{V}_j) \geq \delta$, $|\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) - \pi_d(\mathcal{V}_i, \mathcal{V}_j)| \leq \epsilon \cdot \pi_d(\mathcal{V}_i, \mathcal{V}_j)$.

Algorithm 1: Tau-Push

Input: Graph G, supernode $\mathcal{S}$, constants r_{max}, r'_{max}, τ.
Output: Estimated $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$, $\forall \mathcal{V}_i, \mathcal{V}_j \in \mathcal{S}$ and $\mathcal{V}_i \neq \mathcal{V}_j$.

1 **for** *each* $\mathcal{V}_i \in \mathcal{S}$ **do**
2 $\quad \forall \mathcal{V}_j \in \mathcal{S}, \mathcal{V}_j \neq \mathcal{V}_i$, $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) \leftarrow$FPSN $(\mathcal{V}_i, r_{max})$;
3 **for** $\mathcal{V}_j \in \mathcal{S}$ *where* $\tau_j > \tau$, $\tau_j = \frac{\sum_{v_t \in F(\mathcal{V}_j)} \pi_d(v_t)}{|F(\mathcal{V}_j)|}$ **do**
4 $\quad \forall \mathcal{V}_i \in \mathcal{S}, \mathcal{V}_i \neq \mathcal{V}_j$, $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) \leftarrow$BPSN $(\mathcal{V}_j, r'_{max})$;

Lemma 5.4. *With* $\epsilon = 1 - \left(\frac{\pi_d(\mathcal{V}_i,\mathcal{V}_j)+\pi_d(\mathcal{V}_j,\mathcal{V}_i)}{e}\right)^{\theta}$ *and* $\delta = \frac{e^{1-\sigma}}{2}$, (ϵ, δ)*−approximate level-*ℓ DPPR *ensures that level-*ℓ PDist *is* (θ, σ)*-approximate.*

Lemma 5.4 indicates that the error bound ϵ depends on DPPR itself, where more important node pairs (*i.e.*, with larger DPPR) require a tighter error bound. Further, the premier objective turns to approximate level-ℓ DPPR, which can be solved by extending the PPR approximation methods [4, 15, 26, 38, 41, 47, 48, 57, 58, 73, 75, 86–90, 92].

Deficiencies of existing solutions. To estimate level-ℓ DPPR values w.r.t a source supernode $\mathcal{V}_i$, a straightforward solution is to perform the well-studied single source PPR (SSPPR) approximation methods [4, 15, 26, 38, 41, 73, 88, 90] from each single leaf node in $\mathcal{V}_i$ individually and combine their results in a degree-weighted fashion, which is rather inefficient. For instance, the level-ℓ supernode $\mathcal{V}_i$ needs an $O(k^\ell m)$ [38] cost to repeatedly invoke SSPPR queries from all $O(k^\ell)$ leaf nodes, and even an $O(mn)$ cost at the highest level. Moreover, most of the state-of-the-art methods [41, 73, 88, 90] are built on Forward-Push [4]. Given a source leaf node v_i, Forward-Push maintains two values for each node, *i.e.*, an estimated DPPR and a residue value, and then iteratively converts a certain portion of residues into estimated DPPR values and distributes the remaining residues to neighbors evenly before termination. In particular, it is easy to prove that the following invariant holds during the course of Forward-Push:

$$\pi_d(v_i, v_j) = \widehat{\pi}_d(v_i, v_j) + \sum_{v_k \in V} r(v_i, v_k) \cdot \pi(v_k, v_j), \tag{4}$$

where $\widehat{\pi}_d(v_i, v_j)$ is the estimated DPPR, $r(v_i, v_k)$ is a residue at node v_k and the initial residue for source node v_i is $d(v_i)$. The term $\sum_{v_k \in V} r(v_i, v_k) \cdot \pi(v_k, v_j)$ represents the approximation error. When the residue for each node is eventually depleted, we obtain an exact DPPR result. To ensure the (ϵ, δ)−approximation guarantee, existing methods either rely on numerous push operations to deplete residues or estimate the error term based on costly random walk indexing. However, most of these operations are unnecessary since a large fraction of nodes visited by them are not in the interested supernode $\mathcal{S}$. Particularly, when visualizing a $\mathcal{S}$ at level-1, only about $O(k/n)$ of the random walk samples are useful (*i.e.*, have their end nodes in $\mathcal{S}$).

6 Tau-Push ALGORITHM

Motivated by the inefficiency issues, in this section, we present Tau-Push, our solution for level-ℓ DPPR approximation. We introduce the main idea of Tau-Push in Section 6.1, followed by the detailed description in Section 6.2. We offer a rigorous theoretical analysis for Tau-Push in Section 6.3. Furthermore, we extend the state-of-the-art SSPPR solutions for level-ℓ DPPR computation, and compare them with Tau-Push in Section 6.4.

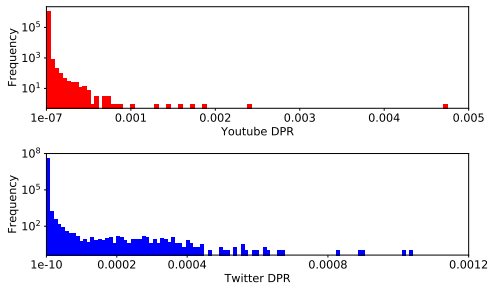


Figure 5: The distributions of DPR on *Youtube* and *Twitter*.

6.1 Overview

Our proposed Tau-Push aims to reduce $O(k^\ell)$ times of leaf-level function calls to one invocation from the level-ℓ supernode, and avoid the random walk samples without degrading the theoretical accuracy guarantees.

The basic idea is to perform *push* operations [4, 58] (*i.e.*, the deterministic version of RWR) from supernodes in a bidirectional manner. The sketch of Tau-Push is illustrated in Algorithm 1. Given a supernode $\mathcal{S}$, Tau-Push first invokes a core subroutine called Forward-Push from SuperNode (FPSN) to compute approximate level-ℓ DPPR values for each source supernode $\mathcal{V}_i \in \mathcal{S}$ (Lines 1-2). After that, Tau-Push proceeds to refine the approximate level-ℓ DPPR values of carefully selected target supernodes by Backward-Push from SuperNode (BPSN) (Lines 3-4). We then describe the main idea of FPSN and explain the rationale of BPSN later.

FPSN can be viewed as a generalized and optimized version of Forward-Push [4]. Recall from Section 5.2 that directly utilizing Forward-Push for level-ℓ DPPR approximation entails immensely expensive costs. Distinct from Forward-Push, FPSN starts push operations from all leaf nodes in the level-ℓ supernode simultaneously with the consideration of their degrees, such that $O(k^\ell)$ times of SSPPR queries are reduced. In addition, we propose to leverage degree-normalized PageRank (DPR) to guide the early termination of push operations in FPSN, thereby largely reducing the push operations and avoiding the random walks. The DPR of node v_j is defined as follows:

$$\rho_j = \sum_{v_k \in V} \frac{d(v_k)}{m} \cdot \pi(v_k, v_j), \tag{5}$$

which essentially is a weighted global PageRank [66]. The DPR values can be efficiently computed in a similar way to the global PageRank by letting the k-th entry in the initial global PageRank be $\frac{d(v_k)}{m}$. Note that if we perform push operations until every $r(v_i, v_k) \leq d(v_k) \cdot r_{max}$, then the error term in Eq. (4) is bounded by

$$\pi_d(v_i, v_j) - \widehat{\pi}_d(v_i, v_j) \leq m \cdot \rho_j \cdot r_{max}.$$

Therefore, (ϵ, δ)−approximate DPPR $\widehat{\pi}_d(v_i, v_j)$ can be satisfied for a given v_j by setting $r_{max} = \frac{\epsilon\delta}{m \cdot \rho_j}$. For the level-$\ell$ supernode, it can be achieved by setting the initial residue of each leaf node to a

Algorithm 2: FPSN

Input: Graph G, source $\mathcal{V}_i$, supernode $\mathcal{S}$, constant r_{max}, τ
Output: Estimated DPPR $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$, $\forall \mathcal{V}_j \in \mathcal{S}$.

1 $r_{init} \leftarrow \frac{d(\mathcal{V}_i)}{|F(\mathcal{V}_i)|}$ where $d(\mathcal{V}_i) = \sum_{v_s \in F(\mathcal{V}_i)} d(v_s)$;
2 $r(\mathcal{V}_i, v_j), \widehat{\pi}_d(\mathcal{V}_i, v_j) \leftarrow 0, \forall v_j \in V$;
3 $r(\mathcal{V}_i, v_j) \leftarrow \frac{d(v_j)}{d(\mathcal{V}_i)} \cdot r_{init}, \forall v_j \in F(\mathcal{V}_i)$;
4 **while** $\exists v_k \in V$ *such that* $r(\mathcal{V}_i, v_k) > d(v_k) \cdot r_{max}$ **do**
5 $\quad \widehat{\pi}_d(\mathcal{V}_i, v_k) \leftarrow \widehat{\pi}_d(\mathcal{V}_i, v_k) + \alpha \cdot r(\mathcal{V}_i, v_k)$;
6 $\quad$ **for** *each* $v_j \in N(v_k)$ **do**
7 $\quad\quad r(\mathcal{V}_i, v_j) \leftarrow r(\mathcal{V}_i, v_j) + (1-\alpha) \cdot \frac{r(\mathcal{V}_i, v_k)}{d(v_k)}$;
8 $\quad r(\mathcal{V}_i, v_k) \leftarrow 0$;
9 $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) \leftarrow \sum_{v_k \in F(\mathcal{V}_j)} \frac{\widehat{\pi}_d(\mathcal{V}_i, v_k)}{|F(\mathcal{V}_j)|}, \forall \mathcal{V}_j \in \mathcal{S}$;

well-designed value and extending DPR ρ_j in a multi-level manner (*i.e.*, τ_j in Algorithm 1), which are elaborated in Section 6.2

Based on the above analysis, FPSN can return $(\epsilon, \delta)-$approximate DPPR $\widehat{\pi}_d(v_i, v_j)$ for each target node $v_j \in \mathcal{S} \backslash v_i$ at level-0 by setting $r_{max} = \frac{\epsilon\delta}{m \cdot \rho_{max}}$ and $\rho_{max} = \max_{v_j \in \mathcal{S} \backslash v_i} \rho_j$. Therefore, the termination of FPSN can be facilitated by pre-computing DPR values, which only requires $O(m)$ running time [66] and $O(n)$ storage space. However, the cost of FPSN is dominated by the largest DPR value in $\mathcal{S}$, which can be extremely large as the DPR values of scale-free networks are highly skewed [56]. For instance, in Figure 5, the largest DPR value of *Youtube* is almost four orders of magnitude greater than the general cases. Therefore, for the supernode $\mathcal{S}$ containing the largest DPR of G, the node with a small DPR value will keep pushing even though its result quality has been satisfied, which makes the worst running time unaffordable.

To remedy this problem, in Lines 3-4, we devise an adaptation of Backward-Push [58] called BPSN, which follows the idea of FPSN but performs the push operation in the reverse direction from the target supernode with large DPR values. Similar to the aforementioned analysis for FPSN, given a target node $v_j \in \mathcal{S}$ at level-0, BPSN stops the reverse push when every $r(v_k, v_j) \leq r'_{max}$ and returns $(\epsilon, \delta)-$approximate DPPR $\widehat{\pi}_d(v_i, v_j)$ for each source leaf node $v_i \in \mathcal{S} \backslash v_j$, where $r'_{max} = \frac{\epsilon\delta}{d_{max}}$ and $d_{max} = \max_{v_i \in \mathcal{S} \backslash v_j} d(v_i)$. For the rationale, BPSN is only performed from a small number of target nodes with large DPR values (see Figure 5). Furthermore, the setting of r'_{max} ensures that the time complexity does not deteriorate under large DPR and out-degree values, since the starting target node with a large DPR value also tends to have a large degree [56], which is excluded from r'_{max}.

To summarize, Tau-Push consists of three key techniques: (i) *push from supernode*, by which FPSN and BPSN can mitigate numerous calls from leaf nodes; (ii) *DPR guided termination*, by which FPSN avoids the expensive random walk samplings; (iii) the *bidirectional push* that combines FPSN and BPSN, by which the worst time complexity of Tau-Push is improved.

6.2 Details

Push from supernode. The procedure of FPSN is shown in Algorithm 2. To compute $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$, FPSN maintains two variables for each node $v_k \in V$: (i) the reserve $\widehat{\pi}_d(\mathcal{V}_i, v_k)$, which is a lower bound of the partial level-ℓ DPPR

$$\pi_d(\mathcal{V}_i, v_k) = \frac{\sum_{v_s \in F(\mathcal{V}_i)} \pi_d(v_s, v_k)}{|F(\mathcal{V}_i)|};$$

(ii) the residue $r(\mathcal{V}_i, v_k)$, which is a by-product. For initialization, the residue of each node $v_j \in F(\mathcal{V}_i)$ is set to $d(v_j)/d(\mathcal{V}_i)$ fraction of r_{init}, where $r_{init} = d(\mathcal{V}_i)/|F(\mathcal{V}_i)|$ and $d(\mathcal{V}_i)$ is the summation of out-degrees of the leaf nodes in $\mathcal{V}_i$. The reserve and residue values of the leaf nodes outside $\mathcal{V}_i$ are all set to zero. During push operations, the residue and reserve values of the leaf nodes are updated according to Lines 5 - 8, in which $N(v_k)$ is the set of out-neighbors of node v_k. Push operations are conducted until $r(\mathcal{V}_i, v_k) \leq d(v_k) \cdot r_{max}$ for all $v_k \in V$, where r_{max} is set to $\frac{\epsilon\delta}{m\tau}$. Finally, the approximation of partial level-ℓ DPPR, *i.e.*, $\widehat{\pi}_d(\mathcal{V}_i, v_k)$, is aggregated into the approximation of level-ℓ DPPR $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ in Line 9.

DPR guided termination. Similar to Eq. (4), the following invariant shows the relation among level-ℓ DPPR, reserve, and residue values while pushing.

Lemma 6.1. FPSN *satisfies the following invariant:*

$$\pi_d(\mathcal{V}_i, \mathcal{V}_j) = \widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) + \sum_{v_t \in F(\mathcal{V}_j)} \sum_{v_k \in V} \frac{r(\mathcal{V}_i, v_k)}{|F(\mathcal{V}_j)|} \cdot \pi(v_k, v_t).$$

Let the average DPR of supernode $\mathcal{V}_j$ be $\tau_j = \sum_{v_s \in F(\mathcal{V}_j)} \rho_s / |F(\mathcal{V}_j)|$. Based on Lemma 6.1, we can derive the following lemma for the correctness of FPSN.

Lemma 6.2. *Given a source supernode $\mathcal{V}_i \in \mathcal{S}$ and a threshold τ, by setting $r_{max} = \frac{\epsilon\delta}{m\tau}$, Algorithm 1 returns $(\epsilon, \delta)-$approximate level-ℓ* DPPR $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ *for $\mathcal{V}_j \in \mathcal{S}$ with $\tau_j \leq \tau$.*

Bidirectional push. Algorithm 3 illustrates the pseudo-code of BPSN. which also exploits the idea of *push from supernode* in FPSN. More specifically, to compute $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ of a fixed target supernode $\mathcal{V}_j$ w.r.t every source supernode $\mathcal{V}_i$, BPSN maintains two values for each node $v_k \in V$: (i) reserve $\widehat{\pi}(v_k, \mathcal{V}_j)$, which is a lower bound of partial DPPR

$$\pi(v_k, \mathcal{V}_j) = \frac{\sum_{v_t \in F(\mathcal{V}_j)} \pi(v_k, v_t)}{|F(\mathcal{V}_j)|};$$

(ii) residue $r(v_k, \mathcal{V}_j)$, which is a by-product. The residue values of the nodes are initialized as Lines 1 - 2 in Algorithm 3, and backward push operation [58] is performed for a node v_k with $r(v_k, \mathcal{V}_j)$ following

$$\begin{cases} r(v_l, \mathcal{V}_j) = r(v_l, \mathcal{V}_j) + \frac{1-\alpha}{d(v_l)} \cdot r(v_k, \mathcal{V}_j), \forall v_l \in N_I(v_k) \\ \widehat{\pi}(v_k, \mathcal{V}_j) = \widehat{\pi}(v_k, \mathcal{V}_j) + \alpha \cdot r(v_k, \mathcal{V}_j); r(v_k, \mathcal{V}_j) = 0 \end{cases}, \quad (6)$$

where $N_I(v_k)$ is the set of in-neighbours of v_k. BPSN stops when every $r(v_k, \mathcal{V}_j) \leq r'_{max}$ and r'_{max} is set as $\frac{\epsilon\delta}{d_{max}}$, in which $d_{max} = \max_{\mathcal{V}_i \in \mathcal{S} \backslash \mathcal{V}_j} d_i$ and $d_i = \sum_{v_s \in F(\mathcal{V}_i)} d(v_s)/|F(\mathcal{V}_i)|$. Finally, the partial DPPR $\pi(v_k, \mathcal{V}_j)$ is assembled into the required approximate DPPR $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ in Line 5 of Algorithm 3.

Similar to the analysis of FPSN, the correctness of BPSN can be derived by employing the invariant in [58].

Lemma 6.3. *Given a target supernode $\mathcal{V}_j \in \mathcal{S}$, by setting $r'_{max} = \frac{\epsilon\delta}{d_{max}}$, Algorithm 3 returns $(\epsilon, \delta)-$approximate level-ℓ* DPPR $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ *for $\mathcal{V}_i \in \mathcal{S}$ and $\mathcal{V}_i \neq \mathcal{V}_j$.*

Algorithm 3: BPSN

Input: Graph G, target $\mathcal{V}_j$, supernode $\mathcal{S}$, constants r'_{max}, τ.
Output: Estimated DPPR $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$, $\forall \mathcal{V}_i \in \mathcal{S}$.

1 $r(v_i, \mathcal{V}_j), \widehat{\pi}(v_i, \mathcal{V}_j) \leftarrow 0, \forall v_i \in V$;
2 $r(v_i, \mathcal{V}_j) \leftarrow 1/F(\mathcal{V}_j), \forall v_i \in F(\mathcal{V}_j)$;
3 **while** $\exists v_k \in V$ *such that* $r(v_k, \mathcal{V}_j) > r'_{max}$ **do**
4 Perform push operation in Eq. (6) from v_k;
5 $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) \leftarrow \sum_{v_s \in F(\mathcal{V}_i)} \frac{d(v_s) \cdot \widehat{\pi}(v_s, \mathcal{V}_j)}{|F(\mathcal{V}_i)|}, \forall \mathcal{V}_i \in \mathcal{S}$;

6.3 Theoretic Analysis

Correctness. Note that Lemmas 6.2 and 6.3 ensure the correctness from any source $\mathcal{V}_i$ to target supernodes $\mathcal{V}_j$ with $\tau_j \leq \tau$ and $\tau_j > \tau$, repsectively. Hence, the following theorem shows the correctness of Tau-Push in Algorithm 1.

Theorem 6.4. *For any user-selected supernode $\mathcal{S}$ and threshold τ, by setting r_{max} and r'_{max} as Lemmas 6.2 and 6.3 respectively, Algorithm 1 returns $(\epsilon, \delta)-$approximate level-ℓ* DPPR $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ *for $\mathcal{V}_i, \mathcal{V}_j \in \mathcal{S}$ and $\mathcal{V}_i \neq \mathcal{V}_j$.*

For the following analysis about τ setting and time complexity, we focus on the level-0 children, *i.e.*, v_i of a level-1 supernode $\mathcal{S}$, where the worst complexities of FPSN and BPSN are achieved. Take FPSN as an example. As the level increases, each supernode $\mathcal{V}_i$ contains more leaf nodes and τ_i tends to $1/n$ (*i.e.*, the average DPR of the entire graph). Thus, the largest τ_i and worst time complexity occur in the level-0.

Setting τ. The threshold τ determines how quickly FPSN can be terminated and how many BPSN are performed. A small τ engenders a low cost for FPSN and a high cost for BPSN, and vice versa. Therefore, the appropriate setting of τ should balance the workloads of FPSN and BPSN in Tau-Push. First, note that the compleixty of FPSN for a given leaf source $v_i \in \mathcal{S}$ is $O\left(d(v_i) \cdot \frac{m\tau}{\epsilon\delta}\right)$ [4], by which the worst time complexity of FPSN occurs when the source out-degree $d(v_i)$ is the largest. Assume that there is only one largest out-degree in the scale-free network. Following the proof by [22], the largest out-degree is $O\left(n^{\frac{1}{b-1}}\right)$, where $b \in [2, 3]$ is the exponent of degree distribution and $b = 2$ on the *Twitter* graph. Hence, the worst time complexity of Lines 1 - 2 of Algorithm 1 is $O\left(\frac{knm\tau}{\epsilon\delta}\right)$. Next, for a leaf target v_j, the worst time complexity of BPSN is $O\left(\frac{\sum_{v_i \in V} d(v_i) \cdot \pi(v_i, v_j)}{r'_{max}}\right)$ [86] and can be simplified as $O\left(\frac{m \cdot \rho_j}{r'_{max}}\right)$ using Eq. (5). As $\rho_j = O(1)$ and $r'_{max} = \frac{\epsilon\delta}{d_{max}}$, the time complexity of BPSN can be re-written as $O\left(\frac{d_{max} \cdot m}{\epsilon\delta}\right)$. Since there are at most $O(1/\tau)$ target nodes with average DPR larger than τ, the time complexity of Lines 3 - 4 of Algorithm 1 is $O\left(\frac{1}{\tau} \cdot \frac{d_{max} \cdot m}{\epsilon\delta}\right)$.

Based on the aforementioned analysis, the worst-case time complexity of Tau-Push is $O\left(\frac{knm\tau}{\epsilon\delta} + \frac{d_{max} \cdot m}{\tau\epsilon\delta}\right)$. Since the degree and DPR follows the same power law [56], the probability that a random-select node has out-degree larger than d_{max} is $\frac{1/\tau}{n}$. Hence, following the proof by [22], $d_{max} = O(\tau n)$ for the scale-free networks. Therefore, the time complexity turns into $O\left(\frac{knm\tau}{\epsilon\delta} + \frac{nm}{\epsilon\delta}\right)$. For the single-source approximation in FPSN, we set δ to $\frac{1}{10 \cdot k} = O(\frac{1}{k})$ [73] as nodes in the same supernode have good connectivity (*i.e.*, large DPPR value) and we focus on the top-$O(k)$ DPPR values for a source. However, the above setting is not suitable for single-target approximation in BPSN because the source nodes and their degrees are unknown with a given target node. Hence, we set $\delta = O(n\tau/k)$ for BPSN as empirically $\sum_{v_i \in F(\mathcal{S})} \pi_d(v_i, v_j)/F(\mathcal{S}) = n \cdot \rho_j/k$ for a random level-1 supernode $\mathcal{S}$ and ρ_j should be comparable to τ. With these configurations, by setting $\tau = 1/\sqrt{kn}$, the worst-case time complexity of Tau-Push in Algorithm 1 is minimized to $O\left(\frac{km}{\epsilon} \cdot \sqrt{kn}\right)$. Note that, by employing FPSN only, the worst-case complexity is $O\left(\frac{k^2nm}{\epsilon}\right)$, which is $\sqrt{kn}$ times slower than Tau-Push.

Indexing scheme. BPSN is only conducted from target $\mathcal{V}_j$ with $\tau_j > \tau$ and is independent of the query supernode $\mathcal{S}$. Hence, we store the leaf-level results of BPSN with $\tau = 1/\sqrt{kn}$, where the index space is $O\left(k \cdot \sqrt{kn}\right)$ since BPSN estimates DPPR of $O\left(\sqrt{kn}\right)$ target nodes w.r.t. $O(k)$ source supernodes in $\mathcal{S}$. Besides, $O(n)$ DPR values are pre-computed, as mentioned in Section 6.1. Overall, the index space of Tau-Push is $O\left(n + k \cdot \sqrt{kn}\right)$.

Time for a random supernode $\mathcal{S}$. In scale-free networks, both DPR and DPPR values follow the power law [9, 56, 57, 73, 89], hence, the i-th largest DPR value is $\frac{1}{i \cdot \log n}$ [73]. As there are n/k supernodes at level-1 and $\mathbb{E}[\tau]$ is upper-bounded by the average of the (n/k) largest DPR values, we have

$$\mathbb{E}[\tau] = \frac{1}{n/k} \cdot \sum_{i=1}^{n/k} \frac{1}{i \cdot \log n} = \frac{k \cdot \log(n/k)}{n \cdot \log n} \leq \frac{k}{n}.$$

Recall that τ is set to $1/\sqrt{kn}$. Hence, given a random query supernode $\mathcal{S}$, Tau-Push will only conduct FPSN if $\mathbb{E}[\tau] \leq 1/\sqrt{kn}$, *i.e.*, $n \geq k^3$. Note that FPSN costs $O\left(\frac{d(\mathcal{V}_i)}{|F(\mathcal{V}_i)|} \cdot \frac{m\tau}{\epsilon\delta}\right)$ [4] from a supernode $\mathcal{V}_i \in \mathcal{S}$, by setting r_{init} as Line 1 in Algorithm 2 and r_{max} as Lemma 6.2. Therefore, by plugging $\mathbb{E}[\tau]$ into the above complexity, we can derive that the complexity of Tau-Push for a random selected supernode $\mathcal{S}$ is $O\left(\sum_{\mathcal{V}_i \in \mathcal{S}} \frac{d(\mathcal{V}_i)}{|F(\mathcal{V}_i)|} \cdot \frac{km}{\epsilon\delta n}\right)$. For easy presentation, we simplify the above complexity by setting $m/n = O(\log n)$ and $\sum_{\mathcal{V}_i \in \mathcal{S}} \frac{d(\mathcal{V}_i)}{|F(\mathcal{V}_i)|} = O(k \cdot \log n)$, where k is the number of supernodes in $\mathcal{S}$ and $O(\log n)$ is the average node degree of scale-free networks. Recall that $\delta = O(1/k)$ [73]. Hence, the complexity is massaged into $O\left(\frac{k^3 \cdot (\log n)^2}{\epsilon}\right)$.

6.4 Comparing Tau-Push with Alternatives

FORA. To estimate level-ℓ DPPR for a source supernode $\mathcal{V}_i \in \mathcal{S}$, we follows the vanilla solution as elaborated in Section 5.2, where FORA [88] is invoked from each leaf node $v_s \in \mathcal{V}_i$. Especially, FORA first utilizes Forward-Push [4] to derive rough approximations of the DPPR values, and then estimates the error term in Eq. (4) by exploiting random walk samplings [26]. Adapting the conclusions in [88], we can show that FORA yields approximate DPPR for each $v_s \in \mathcal{V}_i$.

Table 1: Index space and time complexity of methods for approximate level-ℓ DPPR computation (best in bold).

	FORA	FORA-TP	Tau-Push
Indexing space	$O\left(\frac{\log n \cdot \sqrt{km}}{\epsilon}\right)$	$O\left(\frac{k \cdot \log n \cdot \sqrt{m}}{\epsilon}\right)$	$O\left(n + k \cdot \sqrt{nk}\right)$
Time complexity	$O\left(\frac{m \cdot \sqrt{km}}{\epsilon}\right)$	$O\left(\frac{k \cdot \log n \cdot \sqrt{m}}{\epsilon}\right)$	$\boldsymbol{O\left(\frac{k^3 \cdot (\log n)^2}{\epsilon}\right)}$

Lemma 6.5. *For each source node $v_s \in V$, by setting the initial residue value as $r(v_s, v_s) = d(v_s)$ and performing $\omega = r_{sum} \cdot W$ random walks ,* FORA *returns $(\epsilon, \delta)-$approximate* DPPR $\widehat{\pi}_d(v_s, v_t)$ *with probability at least $1 - p_f$, where $r_{sum} = \sum_{v_j \in V} r(v_s, v_j)$, $W = \frac{(2+2\epsilon/3) \cdot \log(1/p_f)}{\epsilon^2 \delta}$ and p_f is the failure probability.*

Based on Lemma 6.5, plugging the approximate DPPR into Eq. (3) yields $(\epsilon, \delta)-$approximate level-ℓ DPPR. However, FORA has high computation complexity. Specifically, the Forward-Push phase costs $O(d(v_s)/r_{max})$ [4], and the random walk phase costs $O(\omega) = O(m \cdot r_{max} \cdot W)$ as $r_{sum} \leq m \cdot r_{max}$ when push phase finishes. Following [88], we set $r_{max} = \sqrt{d(v_s)/(m \cdot W)}$ to balance the time complexities of two phases. Thus, FORA costs $O\left(\sqrt{d(v_s) \cdot m \cdot W}\right)$ for a source leaf node $v_s \in \mathcal{V}_i$. By summarizing the time complexities for each $v_s \in \mathcal{V}_i$ and $\mathcal{V}_i \in \mathcal{S}$, the time complexity for computing all pairwise approximate level-ℓ DPPR in a supernode $\mathcal{S}$ is $O\left(\sum_{\mathcal{V}_i \in \mathcal{S}} \sum_{v_s \in F(\mathcal{V}_i)} \sqrt{d(v_s) \cdot m \cdot W}\right)$, which is prohibitively high, as a high-level supernode $\mathcal{V}_i$ can contain many leaf nodes.

FORA-TP. To improve the efficiency of the above FORA adaptation, we propose FORA-TP, which conducts push and sampling from the perspective of supernodes. In particular, FORA-TP first performs Tau-Push from a source supernode $\mathcal{V}_i$ to get lower bounds for $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$. Subsequently, the lower bounds are refined by random walk samples, which increases $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ by the fraction of the random walks stopping at $F(\mathcal{V}_j)$. For brevity, the procedure of FORA-TP is summarized in Appendix A. Given a single source supernode $\mathcal{V}_i$, FORA-TP reduces $O(F(\mathcal{V}_i))$ times of function calls of vanilla FORA to only once by utilizing Tau-Push. Based on Lemma 6.1, the following theorem establishes the correctness of FORA-TP.

Theorem 6.6. *For any supernode $\mathcal{V}_i \in \mathcal{S}$, by performing $\omega = \frac{r_{sum}}{\gamma} \cdot W$ random walks,* FORA-TP *returns $(\epsilon, \delta)-$approximate level-ℓ $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ with probability at least $1-p_f$, where $\gamma = \min_{\mathcal{V}_i \in \mathcal{S}} |F(\mathcal{V}_i)|$ and $W = \frac{(2+2\epsilon/3) \cdot \log(1/p_f)}{\epsilon^2 \delta}$.*

Following the time complexity analysis of vanilla FORA, we can show that FORA-TP costs $O\left(\sum_{\mathcal{V}_i \in \mathcal{S}} \frac{d(\mathcal{V}_i)}{|F(\mathcal{V}_i)| \cdot r_{max}} + \frac{m \cdot r_{max}}{\gamma} \cdot W\right)$ to approximate all-pair level-ℓ DPPR in $\mathcal{S}$. For the setting of r_{max}, we balance the complexity of Tau-Push from all source nodes $\mathcal{V}_i \in \mathcal{S}$ with that of the random walk phase. Thus, we have $r_{max} = \sqrt{\gamma \cdot \sum_{\mathcal{V}_i \in \mathcal{S}} d(\mathcal{V}_i)/|F(\mathcal{V}_i)|/(mW)}$, and the time complexity becomes $O\left(\sqrt{\sum_{\mathcal{V}_i \in \mathcal{S}} (d(\mathcal{V}_i)/|F(\mathcal{V}_i)|) \cdot (mW/\gamma)}\right)$.

In Table 1, we compare Tau-Push with FORA and FORA-TP in terms of index space, and time complexity for a random selected supernode $\mathcal{S}$. The results are obtained by setting $p_f = 1/n$ [73, 86–88] and other parameters as explained in Section 6.3, *i.e.*, $\delta = O(1/k)$ and $\frac{d(\mathcal{V}_i)}{|F(\mathcal{V}_i)|} = O(\log n)$.

Table 2: Dataset statistics ($K = 10^3$, $M = 10^6$, $B = 10^9$)

Dataset	n	m	Description
TwEgo	23	52	Ego network[55]
FbEgo	52	146	Ego network[55]
Wiki-ii	186	632	Authorship network[53]
Physician	241	1.8K	Social network[53]
FilmTrust	874	2.6K	User trust network[53]
SciNet	1.5K	5.4K	Collaboration network[53]
Amazon	334.9K	1.9M	Product network [55]
Youtube	1.1M	6.0M	Social network [55]
Orkut	3.1M	234.4M	Social network [55]
DBLP	5.4M	17.2M	Collaboration network [53]
It-2004	41.3M	2.3B	Crawled network [17]
Twitter	41.7M	3.0B	Social network [54]

Index space. For FORA and FORA-TP, we follow [88] and store $r_{max} \cdot d(v_k) \cdot W$ random walks starting from each node v_k, and thus the overall index size is $r_{max} \cdot m \cdot W$. As illustrated in Table 1, due to the difference in r_{max} settings, the index space of FORA-TP is $O\left(\sqrt{k}\right)$ times of FORA, since FORA-TP stores more samples to balance the cost between pushing from k source nodes and the random walk samples. As explained in Section 6.3, Tau-Push needs $O\left(n + k \cdot \sqrt{nk}\right)$ index space. In the experiments, we observe that the index space of Tau-Push is usually one order of magnitude smaller than those of FORA and FORA-TP.

Time complexity. Benefit from the *push from supernode* technique in Tau-Push, FORA-TP improves the time complexity of FORA by $n/\sqrt{k}$. Tau-Push further improves FORA-TP by $\sqrt{m}/(k^2 \cdot \log n)$, which is significant for large graphs (*i.e.*, large m). For example, on the *Twitter* graph, Tau-Push is about $5\times$ faster than FORA-TP.

7 EXPERIMENT EVALUATION

In Section 7.1, we introduce the experiment settings, followed by the evaluation of visualization quality and efficiency in Sections 7.2 and 7.3 respectively. Further, we evaluate PPRviz and its variants in Section 7.4.

7.1 Experiment Settings

Competitors and parameter settings. We compare PPRviz with 13 competitors[3] from different categories. (i) 5 *single-level graph layout methods*: FR [27], LinLog [64], ForceAtlas [45], CMDS [32], PMDS [18]; (ii) 5 *single-level graph embedding methods*: GFactor [3], SDNE [85], LapEig [13], LLE [72], Node2vec [35]; (iii) *an adaptation of PPRviz with SimRank* [46] as introduced in Section 4.2; (iv) 2 *multi-level methods*: OpenOrd [59] and KDraw [60]. For the fair comparison, we follow [5–7] to modify OpenOrd and KDraw such that only the partial view of the cluster $\mathcal{S}$ in the zoom-in path is visualized. For PPRviz, OpenOrd, and KDraw, k (*i.e.*, the maximum number of supernodes in $\mathcal{S}$) is set to 25 as discussed in Section 3.

[3] All codes are open-sourced at https://cutt.ly/BWMmx7T.

Table 3: ND of PPRviz and the baselines, the best in bold and the second best in italic, ∞ indicates infinity.

	PPRviz	OpenOrd/FR	LinLog	ForceAtlas	CMDS	PMDS	GFactor	SDNE	LapEig	LLE	Node2vec	SimRank
TwEgo	*2.1E+02*	**1.2E+02**	1.1E+03	1.8E+03	1.2E+03	∞	3.1E+08	∞	∞	4.6E+02	1.1E+04	5.2E+02
FbEgo	*2.4E+03*	**1.1E+03**	9.5E+03	1.3E+04	2.0E+04	∞	3.6E+12	∞	∞	3.9E+07	1.2E+05	6.2E+03
Wiki-ii	**2.7E+04**	*2.7E+04*	1.4E+05	8.1E+04	4.9E+04	∞	9.2E+11	∞	∞	7.5E+29	2.5E+06	2.7E+04
Physician	**6.7E+04**	*8.7E+04*	7.6E+05	8.2E+05	1.5E+05	∞	2.5E+10	∞	∞	4.0E+09	9.4E+07	1.1E+05
FilmTrust	**9.1E+05**	7.1E+06	3.2E+08	1.4E+07	∞	∞	1.2E+17	∞	∞	1.4E+10	9.6E+07	*2.9E+06*
SciNet	**2.0E+06**	6.5E+12	2.3E+09	1.9E+08	9.9E+12	∞	1.1E+17	∞	∞	∞	6.6E+07	*2.2E+06*

Table 4: ULCV of PPRviz and the baselines, the best in bold and the second best in italic, "-" indicates undefined.

	PPRviz	OpenOrd/FR	LinLog	ForceAtlas	CMDS	PMDS	GFactor	SDNE	LapEig	LLE	Node2vec	SimRank
TwEgo	**0.22**	0.35	0.57	0.37	0.40	*0.23*	0.45	1.96	1.15	0.46	0.80	0.84
FbEgo	**0.39**	*0.42*	0.67	0.49	0.46	0.45	0.91	0.94	0.98	0.77	0.96	0.75
Wiki-ii	**0.35**	*0.41*	1.09	0.64	0.62	0.78	0.62	-	1.04	1.27	0.86	0.53
Physician	**0.45**	0.53	0.90	0.55	0.80	*0.47*	0.95	-	1.02	0.77	1.41	0.53
FilmTrust	**0.48**	*0.54*	1.99	0.96	1.05	0.69	0.64	-	1.70	0.87	0.89	1.78
SciNet	**0.34**	0.77	4.70	1.52	1.74	*0.74*	0.86	-	1.26	-	1.32	1.98

For PDist computation in PPRviz, we configure the relative error $\theta = 0.5$ and other parameters as discussed in Section 6, *e.g.*, $\delta = 1/(10 \cdot k)$. For the competitors, we follow the parameter settings in their original papers. Our competitors are representative. For example, FR, LinLog, ForceAtlas and OpenOrd are integrated into Gephi, a well-known graph visualization software [10].

Datasets and performance metrics. We use 12 real-world graphs in the experiments and their statistics are summarized in Table 2. We generate visualizations in a single-level fashion on the 6 smaller graphs, and report ND and ULCV to evaluate the visualization quality of PPRviz and the competitors. For the fair comparison, we follow NetworkX [36] and normalize each layout to the same scale. We omit crossing number (*i.e.*, the number of crossing edges), as it is reported to be ineffective when the graph is big or dense [52]. The 6 larger graphs are used to evaluate visualization efficiency, on which we report the *response time* and *total preprocessing time*. For the single-level methods, the response time is the time to visualize the entire graph. For PPRviz and the multi-level methods, the response time is obtained over 100 random zoom-in paths on each graph. Each path starts with the supergraph on the highest level (corresponds to the entire graph) and randomly selects a supernode in each level until reaching level-0 (*i.e.*, the original graph) to simulate interactive exploration. The preprocessing time is the time taken before visualization. We terminate a method if its response time (resp. preprocessing time) exceeds 100 seconds (resp. 12 hours). All experiments are conducted on a Linux machine with Intel Xeon(R) Gold 6240@2.60GHz CPU and 377GB RAM in single-thread mode.

7.2 Visualization Quality

In Tables 3 and 4, we report the ND and ULCV of PPRviz and the competitors on the 6 smaller graphs, respectively. Note that a lower score indicates better quality. We use the smaller graphs as most single-level methods cannot return visualization results for the larger graphs in a reasonable time. As OpenOrd applies FR to visualize each supergraph, we combine the two methods into a single column. We omit the results of KDraw as it can only visualize a single connected component and thus crashes on all 6 smaller graphs. Since LLE (resp. SDNE) places all nodes in the same position, ULCV is undefined (*i.e.*, zero divided by zero) for *SciNet* (resp. *Wiki-ii*, *Physician*, *FilmTrust* and *SciNet*) and marked with "-".

Table 3 shows that PPRviz consistently outperforms the competitors in ND expect the *TwEgo* and *FbEgo* graphs, on which PPRviz is the second best. In particular, the ND scores of ForceAtlas, FR and LinLog are five, two and three orders of magnitude larger than PPRviz on *SciNet*, respectively. Some competitors have an infinite ND (*e.g.*, PMDS, SDNE, and LapEig) as they suffer from serious node overlapping. Specifically, PMDS computes the position of a non-pivot node as the weighted combination of its connected pivot nodes, and thus degree-one non-pivot nodes connected to the same pivot will share the same position. As analyzed in Section 4.2, node embedding methods such as SDNE and LapEig suffer from node overlapping as they are designed for node classification and tend to put similar nodes in the same position. FR has the smallest ND scores on *TwEgo* and *FbEgo* graphs as nodes in the largest connected component are placed far apart, which leads to a large distortion in edge length (see Figure 3). Table 4 shows that PPRviz always performs the best in ULCV, which is attributed to our careful definition of DPPR. For instance, PPRviz is 14× better than LinLog on *SciNet*. The competitors have significantly larger ULCV than PPRviz because some edges are significantly longer than the others in their visualizations. Take SimRank as an example that edges in 2-cliques are very long while edges of large connected clusters are very short (see Section 4.2).

7.3 Visualization Efficiency

Response time. Figure 6 reports the response time on the 6 larger graphs. The results show that PPRviz outperforms all 13 competitors in efficiency and its response time is within 1 second for all 6 graphs. Even for the largest *Twitter* graph with about 40 million nodes and 3 billion edges, the response time of PPRviz is only 0.7 seconds. We will show in Section 7.4 that the high efficiency of

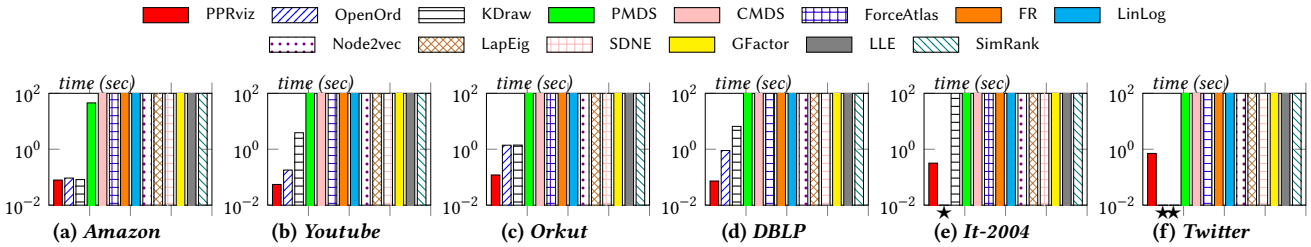


Figure 6: Response time of all methods (⋆ indicates failure to finish preprocessing within 12 hours).

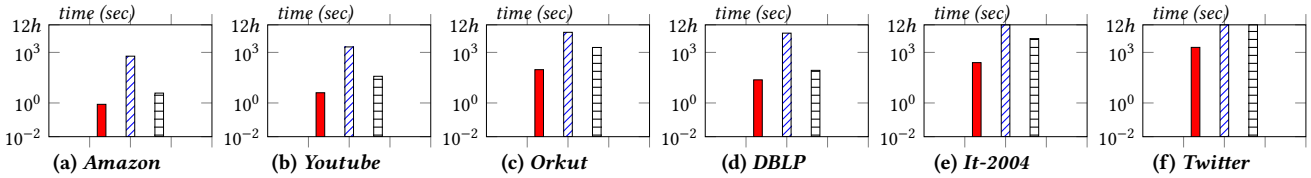


Figure 7: Preprocessing time of the multi-level methods: PPRviz, OpenOrd and KDraw.

PPRviz is attributed to our optimizations for level-ℓ DPPR computation, and without these optimizations, PPRviz would take more than 100 seconds on all tested graphs. The two multi-level methods, *i.e.*, OpenOrd and KDraw, become much slower than PPRviz when the graphs are large. In particular, KDraw uses the supernodes to approximate the repulsive forces from nodes outside $\mathcal{S}$. However, computing the forces inside a supernode $\mathcal{S}$ still incurs high complexity as a supernode can contain numerous nodes at a higher level. For OpenOrd, five stages are employed to determine the final layout of each supergraph, which require more iterations than PPRviz. We also found that OpenOrd and KDraw visualize fewer nodes than PPRviz on each level. For example, on the leaf level of the *Youtube* graph, PPRviz visualizes 25 nodes while OpenOrd and KDraw visualize only 9 and 2 nodes, respectively. Thus, the high efficiency of PPRviz is remarkable as it handles more nodes in each level than OpenOrd and KDraw.

The 5 conventional single-level methods are inefficient because they visualize all nodes of a graph in a single shot. Among them, CMDS, ForceAtlas, LinLog, and FR fail to terminate within 100 seconds for all graphs. Comparatively, PMDS can visualize the whole *Amazon* graph in 45 seconds because it only solves the positions of several pivot nodes, whose linear combinations determine positions of non-pivot nodes. The 5 graph embedding methods (*i.e.*, LapEig, SDNE, GFactor, LLE, and Node2vec) are generally inefficient as they need a large amount of computation to train the node embedding, *e.g.*, expensive matrix operations. SimRank fails to return visualization results for all graphs, as it is well known that computing SimRank is very expensive [46].

Preprocessing time. Figure 7 reports the preprocessing time of the multi-level methods, *i.e.*, PPRviz, OpenOrd, and KDraw, as the single-level methods do not have the preprocessing phase. All three methods conduct the hierarchical clustering from the graph. Furthermore, PPRviz also computes the DPR vector and some single-target DPPR scores. The results show that the processing time of PPRviz is two to three orders of magnitude shorter than OpenOrd and one order of magnitude shorter than KDraw. Moreover, both OpenOrd and KDraw cannot finish preprocessing for the largest *Twitter* graph within 12 hours while PPRviz takes only 33 minutes. OpenOrd costs much more time in preprocessing, as it computes the layout of the entire graph first and then conducts hierarchical clustering on the two-dimensional layout. For KDraw, its community detection algorithm [61] is more expensive than the Louvain+ algorithm in PPRviz.

7.4 PPRviz Variants

Besides our proposed PPRviz, other variants can be devised by replacing our proposed Tau-Push with FORA [88] and FORA-TP, which are introduced in Section 6.4. In this part, we report the performance of PPRviz and its variants.

Response time. We first evaluate the response time of each variant on the 4 large graphs in Table 5. The results show that PPRviz variant FORA incurs more than 100 seconds on all tested graphs. This is because FORA needs to compute DPPR from $O(n)$ leaf nodes for the top-level supergraph. Adopting the idea of push from supernode in Tau-Push, FORA-TP reduces the response time of FORA by at least three orders of magnitude. As analyzed in Table 1, Tau-Push can improve FORA-TP on the graph with massive edges, where the redundant random walks in FORA-TP turn to the main bottleneck. In particular, Tau-Push further speeds up FORA-TP by 3×, 2.3× and 4× for *Orkut*, *It-2004* and *Twitter* graphs respectively.

Preprocessing time. We show the preprocessing time of each variant in Table 6. Besides the clustering subroutine Louvain+, FORA, FORA-TP and Tau-Push require to pre-construct certain index for DPPR computation as explained in Section 6.4. In particular, FORA costs the shortest preprocessing time as it only computes some random walks from each node, and FORA-TP needs a longer time as it requires more random walks. Tau-Push is faster than FORA-TP on *Youtube*, *Orkut*, and *It-2004* graph but slower on the *Twitter* graph, since the complexity of DPR computation is proportional to the number of edges [66]. Overall, the main bottleneck of preprocessing is the conduction of Louvain+. For *Twitter* with about 3 billion edges, Tau-Push costs extra 10 minutes compared with FORA.

Index size. We compare the index size in Table 7, where each method stores both the supergraph hierarchy and the index for

Table 5: Response time of PPRviz variants (in seconds).

	Method		
Dataset	Tau-Push	FORA-TP	FORA
Youtube	0.06	0.07	>100
Orkut	0.12	0.36	>100
It-2004	0.32	0.73	>100
Twitter	0.71	2.76	>100

Table 6: Preprocessing time of PPRviz variants (in seconds).

	Method		
Dataset	Tau-Push	FORA-TP	FORA
Youtube	4.04	5.10	3.35
Orkut	94.56	104.94	79.89
It-2004	312.33	308.30	223.99
Twitter	1984.73	1485.64	1364.04

Table 7: Index size of PPRviz variants (in MiB).

	Method		
Dataset	Tau-Push	FORA-TP	FORA
Youtube	9	51	30
Orkut	25	237	102
It-2004	330	1520	1041
Twitter	338	1610	1075

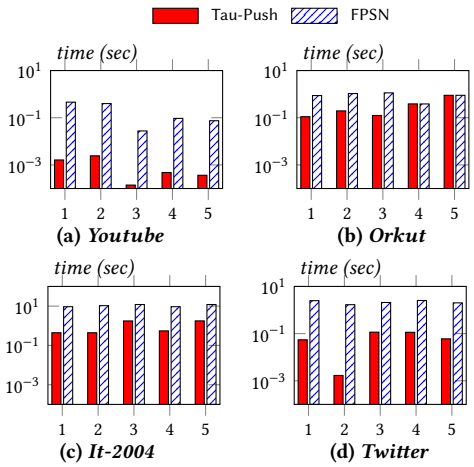


Figure 8: Tau-Push and FPSN Comparison.

DPPR computation. For FORA and FORA-TP, they require a much larger index than Tau-Push as they need to store many random walk samples. The index size of Tau-Push is very small compared with the original graph. More specifically, it only stores $O(n)$ supernode partitions, n DPR values and $O(k \cdot \sqrt{n \cdot k})$ extra DPPR values.

Ablation study. In Figure 8, we show that Tau-Push significantly improves the efficiency of FPSN when the cluster contains nodes with large DPR values. We select 5 level-1 clusters in each graph that contain leaf nodes with the largest DPR, and measure the average time to compute approximate level-0 DPPR from one source node to all target nodes. The speed of Tau-Push varies since the skewness of DPR value is different on each graph. For the first cluster in *Youtube*, the largest DPR value is 3 orders of magnitude larger than the others, and thus Tau-Push speeds up FPSN by about 300× through avoiding many rounds of push operations. For a similar reason, Tau-Push is over 1000× faster than FPSN in the second cluster in *Twitter*. The speedup of Tau-Push is less significant on *Orkut* and *It-2004* graphs, but can still reach 9× and 24× respectively.

8 RELATED WORK

Graph visualization. Conventional single-level graph visualization methods have been extensively studied [33, 40, 43, 83] and can be classified into two main categories: (i) *force-directed methods* [21, 24, 27, 45, 59, 64] and (ii) *stress methods* [18, 30–32, 49, 50, 60, 76, 80]. To improve efficiency, many optimizations are incorporated, *e.g.*, grid-based partitioning [27, 59], quad-tree [45, 64, 81], and dimensionality reduction [18, 50, 76]. Multi-level methods [1, 6, 7, 11, 29, 42, 59, 60, 71, 74, 84] mainly focus on designing clustering algorithms, and the single-level methods are directly applied to layout each cluster. For example, OpenOrd [59] clusters nodes based on their Euclidean distances in the graph layout and uses FR for visualization; KDraw [60] applies label propagation for clustering and uses [31] for visualization; Grouse/GrouseFlocks [6, 7] groups nodes based on certain graph structures. Different from existing works, our PPRviz defines node distance using PPR and provides high visualization quality. We also formulate level-ℓ DPPR to accurately capture the structure of the original graph when conducting visualization in a multi-level fashion. A related work uses PPR for node clustering but still relies on a force-directed method for visualization [21].

PPR computation. The efficient computation of PPR has been extensively studied [4, 15, 26, 38, 41, 47, 48, 57, 58, 73, 75, 86–92]. Among these works, BEAR [75] and BePI [48] improve the power iteration method [38] and achieve high efficiency by indexing several large matrices. However, the index space limits their applicability on large graphs. BiPPR [57] and HubPPR [87] combine random walks [26] with Backward-Push [58], and are subsequently improved by FORA [88]. We have shown that FORA is ineffective for level-ℓ DPPR. Similar to FORA [88], SpeedPPR [90] and DeltaPush [41] utilize random walk samplings to refine the estimation, and thus also suffer from the ineffective sample problem when used for level-ℓ DPPR. In essence, our level-ℓ DPPR computation problem is different from PPR computation as it aims at the aggregated PPRs between two clusters of nodes. Our Tau-Push enables PPRviz to significantly outperform existing visualization solutions in efficiency.

9 CONCLUSIONS

This paper proposes a multi-level visualization algorithm, PPRviz, for massive graphs. PPRviz formulates a PPR-based node distance PDist to determine graph layout, which preserves graph topology information and provides good visualization quality. Furthermore, PPRviz leverages an efficient subroutine Tau-Push to compute PDist efficiently. We conduct extensive experiments to compare PPRviz with 13 competitors on 12 real-world graphs, and the results show that PPRviz achieves high effectiveness and efficiency.

REFERENCES

[1] James Abello, Frank Van Ham, and Neeraj Krishnan. 2006. Ask-graphview: A large scale graph visualization system. *TVCG* (2006).

[2] Giuseppe Agapito, Pietro Hiram Guzzi, and Mario Cannataro. 2013. Visualization of protein interaction networks: problems and solutions. *BMC* (2013).

[3] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. 2013. Distributed large-scale natural graph factorization. In *WWW*.

[4] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *FOCS*.

[5] Daniel Archambault, Tamara Munzner, and David Auber. 2007. Topolayout: Multilevel graph layout by topological features. *TVCG* (2007).

[6] Daniel Archambault, Tamara Munzner, and David Auber. 2008. GrouseFlocks: Steerable exploration of graph hierarchy space. *TVCG* (2008).

[7] Daniel W Archambault, Tamara Munzner, and David Auber. 2007. Grouse: Feature-Based, Steerable Graph Hierarchy Exploration.. In *EuroVis*.

[8] David Auber. 2004. Tulip—A huge graph visualization framework. In *GDS*.

[9] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. 2010. Fast incremental and personalized pagerank. *arXiv* (2010).

[10] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. 2009. Gephi: an open source software for exploring and manipulating networks. In *AAAI*.

[11] Vladimir Batagelj, Franz J Brandenburg, Walter Didimo, Giuseppe Liotta, Pietro Palladino, and Maurizio Patrignani. 2010. Visual analysis of large graphs using (x, y)-clustering and hybrid visualizations. *TVCG* (2010).

[12] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G Tollis. 1998. *Graph drawing: algorithms for the visualization of graphs*.

[13] Mikhail Belkin and Partha Niyogi. 2003. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.* (2003).

[14] Chris Bennett, Jody Ryall, Leo Spalteholz, and Amy Gooch. 2007. The aesthetics of graph visualization. *CAe* (2007).

[15] Pavel Berkhin. 2006. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Math.* (2006).

[16] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *J. Stat. Mech. Theory Exp.* (2008).

[17] Paolo Boldi and Sebastiano Vigna. 2004. The WebGraph Framework I: Compression Techniques. In *WWW*.

[18] Ulrik Brandes and Christian Pich. 2006. Eigensolver methods for progressive multidimensional scaling of large data. In *GD*.

[19] Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W Klau, Karsten Klein, and Petra Mutzel. 2011. The Open Graph Drawing Framework (OGDF). *Handbook of graph drawing and visualization* (2011).

[20] Fan Chung and Linyuan Lu. 2006. Concentration inequalities and martingale inequalities: a survey. *Internet Math.* (2006).

[21] Fan Chung and Alexander Tsiatas. 2012. Finding and visualizing graph clusters using PageRank optimization. *Internet Math.* (2012).

[22] Reuven Cohen, Keren Erez, Daniel Ben-Avraham, and Shlomo Havlin. 2001. Breakdown of the internet under intentional attack. *Phys. Rev. Lett.* (2001).

[23] Ron Davidson and David Harel. 1996. Drawing graphs nicely using simulated annealing. *TOG* (1996).

[24] Peter Eades. 1984. A heuristic for graph drawing. *Congr. Numer.* (1984).

[25] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. 2001. Graphviz—open source graph drawing tools. In *GD*.

[26] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. 2005. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Math.* (2005).

[27] Thomas MJ Fruchterman and Edward M Reingold. 1991. Graph drawing by force-directed placement. *SP&E* (1991).

[28] Yasuhiro Fujiwara, Yasutoshi Ida, Sekitoshi Kanai, Atsutoshi Kumagai, and Naonori Ueda. 2021. Fast Similarity Computation for t-SNE. In *ICDE*.

[29] Pawel Gajer, Michael T Goodrich, and Stephen G Kobourov. 2000. A multi-dimensional approach to force-directed layouts of large graphs. In *GD*.

[30] Emden R Gansner, Yifan Hu, and Shankar Krishnan. 2013. COAST: A convex optimization approach to stress-based embedding. In *GD*.

[31] Emden R Gansner, Yifan Hu, and Stephen North. 2012. A maxent-stress model for graph layout. *TVCG* (2012).

[32] Emden R Gansner, Yehuda Koren, and Stephen North. 2004. Graph drawing by stress majorization. In *GD*.

[33] Helen Gibson, Joe Faith, and Paul Vickers. 2013. A survey of two-dimensional graph layout techniques for information visualisation. *Inf. Vis.* (2013).

[34] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowl.-Based Syst.* (2018).

[35] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*.

[36] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX*. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[37] Jan-Henrik Haunert and Leon Sering. 2011. Drawing road networks with focus regions. *TVCG* (2011).

[38] Taher Haveliwala and Sepandar Kamvar. 2003. *The second eigenvalue of the Google matrix*. Technical Report. Stanford.

[39] Nathalie Henry, Jean-Daniel Fekete, and Michael J McGuffin. 2007. NodeTrix: a hybrid visualization of social networks. *TVCG* (2007).

[40] Ivan Herman, Guy Melançon, and M Scott Marshall. 2000. Graph visualization and navigation in Inf. Vis.: A survey. *TVCG* (2000).

[41] Guanhao Hou, Xingguang Chen, Sibo Wang, and Zhewei Wei. 2021. Massively parallel algorithms for personalized PageRank. *PVLDB* (2021).

[42] Yifan Hu. 2005. Efficient, high-quality force-directed graph drawing. *Mathematica* (2005).

[43] Yifan Hu and Lei Shi. 2015. Visualizing large graphs. *Wiley Interdiscip. Rev. Comput. Stat.* (2015).

[44] Weidong Huang, Peter Eades, and Seok-Hee Hong. 2009. Measuring effectiveness of graph visualizations: A cognitive load perspective. *Inf. Vis.* (2009).

[45] Mathieu Jacomy, Tommaso Venturini, Sebastien Heymann, and Mathieu Bastian. 2014. ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software. *PloS one* (2014).

[46] Glen Jeh and Jennifer Widom. 2002. Simrank: a measure of structural-context similarity. In *SIGKDD*.

[47] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *WWW*.

[48] Jinhong Jung, Namyong Park, Sael Lee, and U Kang. 2017. Bepi: Fast and memory-efficient method for billion-scale random walk with restart. In *SIGMOD*.

[49] Tomihisa Kamada, Satoru Kawai, et al. 1989. An algorithm for drawing general undirected graphs. *Inform. Process. Lett.* (1989).

[50] Marc Khoury, Yifan Hu, Shankar Krishnan, and Carlos Scheidegger. 2012. Drawing Large Graphs by Low-Rank Stress Majorization. In *CGF*.

[51] Moritz Klammler, Tamara Mchedlidze, and Alexey Pak. 2018. Aesthetic discrimination of graph layouts. In *GD*.

[52] Stephen G Kobourov, Sergey Pupyrev, and Bahador Saket. 2014. Are crossings important for drawing large graphs?. In *GD*.

[53] Jérôme Kunegis. 2013. KONECT – The Koblenz Network Collection. In *WWW*.

[54] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a social network or a news media?. In *WWW*.

[55] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[56] Nelly Litvak, Werner RW Scheinhardt, and Yana Volkovich. 2007. In-degree and PageRank: why do they follow similar power laws? *Internet Math.* (2007).

[57] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2016. Personalized pagerank estimation and search: A bidirectional approach. In *WSDM*.

[58] Peter Lofgren and Ashish Goel. 2013. Personalized pagerank to a target node. *arXiv* (2013).

[59] Shawn Martin, W Michael Brown, Richard Klavans, and Kevin W Boyack. 2011. OpenOrd: an open-source toolbox for large graph layout. In *VDA*.

[60] Henning Meyerhenke, Martin Nöllenburg, and Christian Schulz. 2017. Drawing large graphs by multilevel maxent-stress optimization. *TVCG* (2017).

[61] Henning Meyerhenke, Peter Sanders, and Christian Schulz. 2014. Partitioning complex networks via size-constrained clustering. In *SEA*.

[62] Mark EJ Newman. 2006. Modularity and community structure in networks. *PNAS* (2006).

[63] Constantin Niculescu and Lars-Erik Persson. 2006. *Convex functions and their applications*. Springer.

[64] Andreas Noack. 2005. Energy-based clustering of graphs with nonuniform degrees. In *GD*.

[65] Andreas Noack. 2007. Unified quality measures for clusterings, layouts, and orderings of graphs, and their application as software design criteria. (2007).

[66] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web*. Technical Report. Stanford InfoLab.

[67] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*.

[68] Helen C Purchase. 2002. Metrics for graph drawing aesthetics. *JVLC* (2002).

[69] Helen C Purchase, David Carrington, and Jo-Anne Allder. 2002. Empirical evaluation of aesthetics-based graph layout. *Empir. Softw. Eng.* (2002).

[70] Aaron Quigley and Peter Eades. 2000. Fade: Graph drawing, clustering, and visual abstraction. In *GD*.

[71] Jose Rodrigues, Hanghang Tong, Agma Traina, Christos Faloutsos, and Jure Leskovec. 2015. Gmine: a system for scalable, interactive graph visualization and mining. *arXiv* (2015).

[72] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* (2000).

[73] Jieming Shi, Renchi Yang, Tianyuan Jin, Xiaokui Xiao, and Yin Yang. 2019. Real-time top-k personalized pagerank over large graphs on gpus. *PVLDB* (2019).

[74] Lei Shi, Nan Cao, Shixia Liu, Weihong Qian, Li Tan, Guodong Wang, Jimeng Sun, and Ching-Yung Lin. 2009. HiMap: Adaptive visualization of large-scale online social networks. In *PacificVis*.

[75] Kijung Shin, Jinhong Jung, Sael Lee, and U Kang. 2015. Bear: Block elimination approach for random walk with restart on large graphs. In *SIGMOD*.
[76] Vin D Silva and Joshua B Tenenbaum. 2003. Global versus local methods in nonlinear dimensionality reduction. In *NIPS*.
[77] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*.
[78] Martyn Taylor and Peter Rodgers. 2005. Applying graphical design techniques to graph visualisation. In *Inf. Vis.*
[79] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. In *ICDM*. 613–622.
[80] Warren S Torgerson. 1952. Multidimensional scaling: I. Theory and method. *Psychometrika* (1952).
[81] Daniel Tunkelang. 1994. *A practical approach to drawing undirected graphs.* Technical Report.
[82] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *JMLR* (2008).
[83] Tatiana Von Landesberger, Arjan Kuijper, Tobias Schreck, Jörn Kohlhammer, Jarke J van Wijk, J-D Fekete, and Dieter W Fellner. 2011. Visual analysis of large graphs: state-of-the-art and future research challenges. In *CGF*.
[84] Chris Walshaw. 2000. A multilevel algorithm for force-directed graph drawing. In *GD*.
[85] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *SIGKDD*.
[86] Hanzhi Wang, Zhewei Wei, Junhao Gan, Sibo Wang, and Zengfeng Huang. 2020. Personalized PageRank to a Target Node, Revisited. In *SIGKDD*.
[87] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. HubPPR: effective indexing for approximate personalized pagerank. *PVLDB* (2016).
[88] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: simple and effective approximate single-source personalized pagerank. In *SIGKDD*.
[89] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. 2018. Topppr: top-k personalized pagerank queries with precision guarantees on large graphs. In *SIGMOD*.
[90] Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. 2021. Unifying the Global and Local Approaches: An Efficient Power Iteration with Forward Push. In *SIGMOD*.
[91] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S Bhowmick. 2020. Homogeneous network embedding for massive graphs via reweighted personalized PageRank. *PVLDB* (2020).
[92] Minji Yoon, Jinhong Jung, and U Kang. 2018. Tpa: Fast, scalable, and accurate method for approximate random walk with restart on billion scale graphs. In *ICDE*.

A APPENDIX

A.1 Algorithmic Details

Louvain+. To construct supergraph hierarchy, a solution is directly using multilevel community detection algorithms such as Louvain [16], which merges well-connected nodes based on modularity optimization [62]. However, Louvain boils down to two defects for visualization: (i) the number of communities (supernodes) in the highest level is too large as there is no merge that increases the modularity after some point, which causes visual clutter; (ii) the number of nodes in the communities are imbalanced. Specifically, low-level supernodes tend to contain many children, which leads to visual cluster while high-level supernodes usually contain only a few children and thus provide very limited structural information about the graph. To fix these, we extend Louvain to Louvain+.

Here we ignore the direction in the raw graph and take the undirected graph as the input for community detection. To generate a level-$(\ell+1)$ supergraph $G_{\ell+1}$, the detailed clustering strategy is that either (i) directly merge supernode $\mathcal{S}$ to its neighboring supernode $\mathcal{T}$ if $\mathcal{T}$ is the only neighbor; or (ii) merge $\mathcal{S}$ to its neighboring supernode $\mathcal{T}$ with the largest modularity gain $\widehat{Q}(\mathcal{S},\mathcal{T})$ if the size of $\mathcal{T}$ after this merge is less than k. The modularity gain $\widehat{Q}(\mathcal{S},\mathcal{T})$ after merging level-$(\ell+1)$ supernodes $\mathcal{S}$ and $\mathcal{T}$ is defined as

$$\widehat{Q}(\mathcal{S},\mathcal{T}) = \sum_{\mathcal{V}_i \in \mathcal{S}} q(\mathcal{V}_i,\mathcal{T}).$$

Note that $q(\mathcal{V}_i,\mathcal{T})$ is the modularity change of G_ℓ after moving a node $\mathcal{V}_i$ into supernode $\mathcal{T}$, which is defined by

$$q(\mathcal{V}_i,\mathcal{T}) = \left[\frac{w(\mathcal{T})+w_{cr}(\mathcal{V}_i,\mathcal{T})}{2m} - \left(\frac{w_{in}(\mathcal{T})+w_{in}(\mathcal{V}_i)}{2m}\right)^2\right] - \left[\frac{w(\mathcal{T})}{2m} - \left(\frac{w_{in}(\mathcal{T})}{2m}\right)^2 - \left(\frac{w_{in}(\mathcal{V}_i)}{2m}\right)^2\right],$$

where $w(\mathcal{T})$ is the number of leaf edges with both endpoints within $\mathcal{T}$, $w_{cr}(\mathcal{V}_i,\mathcal{T})$ is the number of leaf edges crossing supernodes $\mathcal{V}_i$ and $\mathcal{T}$, and $w_{in}(\mathcal{T})$ is the number of leaf edges incident to $\mathcal{T}$.

Stress Majorization. Stress majorization [32] is adapted to enable efficient optimization, Particularly, Eq. (1) is transformed via the following steps. First, the expansion of Eq. (1) is rewritten as follows:

$$Loss(\mathbf{X}|\boldsymbol{\Delta}) = \sum_{i<j} 1 + \sum_{i<j} \frac{||\mathbf{X}[i]-\mathbf{X}[j]||^2}{\boldsymbol{\Delta}[i,j]^2} - 2\sum_{i<j}\frac{||\mathbf{X}[i]-\mathbf{X}[j]||}{\boldsymbol{\Delta}[i,j]}. \quad (7)$$

In Eq. (7), the first term is a constant and the second term can be represented by $\text{trace}(\mathbf{X}^T\mathbf{L}^w\mathbf{X})$, where $\mathbf{L}^w$ is the weighted Laplacian matrix and defined as

$$\mathbf{L}^w[i,j] = \begin{cases} -\frac{1}{\boldsymbol{\Delta}[i,j]^2} & \text{if } i \neq j \\ \sum_{k\neq i}\frac{1}{\boldsymbol{\Delta}[i,k]^2} & \text{if } i = j \end{cases}.$$

According to the Cauchy-Schwartz inequality, we know that for any position matrix $\mathbf{Y}$, $||\mathbf{X}[i]-\mathbf{X}[j]||||\mathbf{Y}[i]-\mathbf{Y}[j]|| \geq (\mathbf{X}[i]-\mathbf{X}[j])^T(\mathbf{Y}[i]-\mathbf{Y}[j])$ holds. Hence, the third term can be bounded by

$$-2\sum_{i<j}\frac{||\mathbf{X}[i]-\mathbf{X}[j]||}{\boldsymbol{\Delta}[i,j]} \leq -2\sum_{i<j}\frac{(\mathbf{X}[i]-\mathbf{X}[j])^T(\mathbf{Y}[i]-\mathbf{Y}[j])}{\boldsymbol{\Delta}[i,j]||\mathbf{Y}[i]-\mathbf{Y}[j]||}.$$

Then the third term can be written as $-\text{trace}(2\mathbf{X}^T\mathbf{L}^Y\mathbf{Y})$, where the matrix $\mathbf{L}^Y$ is defined as

$$\mathbf{L}^Y[i,j] = \begin{cases} -\frac{1}{\boldsymbol{\Delta}[i,j]||\mathbf{Y}[i]-\mathbf{Y}[j]||} & \text{if } i \neq j \text{ and } \mathbf{Y}[i] \neq \mathbf{Y}[j] \\ -\sum_{k\neq i}\mathbf{L}^Y[i,k] & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}.$$

Hence, the loss function in Eq. (1) can be bounded by

$$\sum_{i<j} 1 + \text{trace}(\mathbf{X}^T\mathbf{L}^w\mathbf{X}) - \text{trace}(2\mathbf{X}^T\mathbf{L}^Y\mathbf{Y}). \quad (8)$$

To optimize Eq. (8), let $\mathbf{Y}$ be the position matrix from the previous iteration and $\mathbf{X}$ be the position matrix to be optimized in the current iteration. By setting the derivative of Eq. (8) to zero with respect to $\mathbf{X}$, the minimizer of the loss function satisfies

$$\mathbf{X} = (\mathbf{L}^w)^{-1}\mathbf{L}^Y\mathbf{Y}, \quad (9)$$

by which position matrix $\mathbf{X}$ is iteratively optimized and finally returned for visualization.

FORA-TP. The pseudo-code of FORA-TP is illustrated in Algorithm 4. Particularly, FORA-TP first performs Tau-Push, which returns the estimated value $\widehat{\pi}_d(\mathcal{V}_i,\mathcal{V}_j)$ for each $\mathcal{V}_j \in \mathcal{S}$ and the residue value $r(\mathcal{V}_i,v_k)$ for each $v_k \in V$ (Line 1). After that, ω times of random walk samplings are invoked. For each sampling, it starts from node v_k, where v_k is selected based on its residue $r(\mathcal{V}_i,v_k)$

Algorithm 4: FORA-TP

Input: Graph G, source $\mathcal{V}_i$, supernode $\mathcal{S}$, constants r_{max}, ω.
Output: $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$, $\forall \mathcal{V}_j \in \mathcal{S}$.

1 $\forall \mathcal{V}_j \in \mathcal{S}$, $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) \leftarrow$ Tau-Push $(\mathcal{V}_i, r_{max})$;
2 $r_{sum} \leftarrow \sum_{v_j \in V} r(\mathcal{V}_i, v_j)$;
3 **repeat**
4 Perform a random walk from a node v_k with probability $\frac{r(\mathcal{V}_i, v_k)}{r_{sum}}$. Let v_t be the ending node of the random walk;
5 $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ increases by $\frac{r_{sum}}{\omega \cdot |F(\mathcal{V}_j)|}$ for $v_t \in F(\mathcal{V}_j)$;
6 **until** ω *times*;

(Line 4). Suppose that the sampling stops at $v_t \in F(\mathcal{V}_j)$, then the estimated value $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ is increased by $\frac{r_{sum}}{\omega \cdot |F(\mathcal{V}_j)|}$ (Line 5).

A.2 More Visualizations

In Figures 9 to 13, we report the visualization results of PPRviz and other competitors on *FbEgo*, *Wiki-ii*, *Physician*, *FilmTrust* and *SciNet* graphs.

A.3 Proofs

Proof of Theorem 4.2. Note that PDist is at least 2. Hence the node distribution is, $\mathrm{ND}(\mathbf{X}) = \sum_{i<j} \frac{1}{\Delta[i,j]^2} \leq \frac{n(n-1)}{8}$.

Proof of Theorem 4.3. We first need the following lemma.

Lemma A.1. *Given the* PDist *matrix* Δ *of graph* G*, suppose that* $||\mathbf{X}[i] - \mathbf{X}[j]|| = \Delta[i,j]$ *for* $v_i, v_j \in V$*, then any edge length* $l(v_i, v_j)$ *is bounded by* $l(v_i, v_j) \leq 1 - \log(2\alpha(1-\alpha))$.

Based on Lemma A.1 and Definition 4.1, the edge length is in range $[2, 1 - \log 2\alpha(1-\alpha)]$, where the teleport probability $\alpha \leq \frac{1}{2} - \sqrt{\frac{1}{4} - \frac{1}{2e}}$. $l_\sigma \leq (\log \frac{1}{2\alpha(1-\alpha)} - 1)/2$ based on Popoviciu's inequality [63]. Since $l_\mu \geq 2$, hence, $\mathrm{ULCV}(\mathbf{X}) \leq (\log \frac{1}{2\alpha(1-\alpha)} - 1)/4$.

Proof of Lemma A.1. For any random walk starting from v_i, the probability that the walk moves to a certain one-hop neighbor v_j is $(1-\alpha)/d(v_s)$. Since the walk stops at v_j with probability α, then the PPR value of a neighbor is bounded by

$$\pi(v_i, v_j) \geq \frac{\alpha(1-\alpha)}{d(v_i)}.$$

By Definition 4.1,

$$\Delta[i,j] \leq 1 - \log(2\alpha(1-\alpha)).$$

Since $\Delta[i,j] = ||X[i] - X[j]||$, the edge length $l(v_i, v_j)$ is equal to $\Delta[i,j]$ between v_i and its one-hop neighbor v_j. Hence, the proof completes.

Proof of Lemma 5.4. When DPPR is larger than δ, denote $c = (\frac{\pi_d(\mathcal{V}_i, \mathcal{V}_j) + \pi_d(\mathcal{V}_j, \mathcal{V}_i)}{e})^\theta$. There exists two cases to satisfy the relative error part of (θ, σ)-approximate PDist, there exists two cases.

- For $\widehat{\Delta}[i,j] - \Delta[i,j] \leq \theta \cdot \Delta[i,j]$, the following approximation should be satisfied

$$\pi_d(\mathcal{V}_i, \mathcal{V}_j) - \widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) \leq (1-c) \cdot \pi_d(\mathcal{V}_i, \mathcal{V}_j).$$

- For $\Delta[i,j] - \widehat{\Delta}[i,j] \leq \theta \cdot \Delta[i,j]$, the following approximation should be satisfied

$$\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) - \pi_d(\mathcal{V}_i, \mathcal{V}_j) \leq (\frac{1}{c} - 1) \cdot \pi_d(\mathcal{V}_i, \mathcal{V}_j).$$

Recall that $\Delta[i,j] \geq 2$ and $\pi_d(\mathcal{V}_i, \mathcal{V}_j) \leq 1/e$ accordingly, then $c < 1$ and $\frac{1}{c} - 1 > 1 - c$. By setting

$$\epsilon = 1 - c = 1 - (\frac{\pi_d(\mathcal{V}_i, \mathcal{V}_j) + \pi_d(\mathcal{V}_j, \mathcal{V}_i)}{e})^\theta,$$

the relative error part of (θ, σ)-approximate PDist holds. When DPPR is not larger than δ,

$$|\Delta[i,j] - \widehat{\Delta}[i,j]| \leq \theta \cdot (1 - \log 2\delta) = \theta \cdot \sigma.$$

Proof of Lemma 6.1 For each $v_s \in F(\mathcal{V}_i)$, denote $\widehat{\pi}_d(\mathcal{V}_i, v_s, v_k)$ and $r(\mathcal{V}_i, v_s, v_k)$ as the lower bound and residue value, which is firstly initialized from v_s and then distributed to v_k. Initially, $\widehat{\pi}_d(\mathcal{V}_i, v_s, v_s)$ is set to $\frac{d(v_s)}{F(\mathcal{V}_i)}$ (Line 3). According to the proof in [4], the following equation holds for each $v_t \in V$ in graph traversal,

$$\frac{\pi_d(v_s, v_t)}{|F(\mathcal{V}_i)|} = \widehat{\pi}_d(\mathcal{V}_i, v_s, v_t) + \sum_{v_k \in V} r(\mathcal{V}_i, v_s, v_k) \cdot \pi(v_k, v_t).$$

Note that for each $v_t \in V$, $\sum_{v_s \in F(\mathcal{V}_i)} \widehat{\pi}_d(\mathcal{V}_i, v_s, v_t) = \widehat{\pi}_d(\mathcal{V}_i, v_j)$ and residue value holds the same relationships. By summing up all $v_s \in F(\mathcal{V}_i)$, the above equation turns to

$$\pi_d(\mathcal{V}_i, v_t) = \widehat{\pi}_d(\mathcal{V}_i, v_t) + \sum_{v_k \in V} r(\mathcal{V}_i, v_k) \cdot \pi(v_k, v_t).$$

By Eq. (3), the DPPR between supernodes $\mathcal{V}_i$ and $\mathcal{V}_j$ is $\pi_d(\mathcal{V}_i, \mathcal{V}_j) = \frac{\sum_{v_t \in F(\mathcal{V}_j)} \pi_d(\mathcal{V}_i, v_t)}{|F(\mathcal{V}_j)|}$. Then the above equation can be transformed to

$$\pi_d(\mathcal{V}_i, \mathcal{V}_j) = \widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) + \sum_{v_t \in F(\mathcal{V}_j)} \sum_{v_k \in V} \frac{r(\mathcal{V}_i, v_k)}{|F(\mathcal{V}_j)|} \cdot \pi(v_k, v_t).$$

Proof of Lemma 6.2 Based on Lemma 6.1, for any supernode $\mathcal{V}_i, \mathcal{V}_j \in \mathcal{S}$, we can express the approximation error as

$$\pi_d(\mathcal{V}_i, \mathcal{V}_j) - \widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) = \sum_{v_t \in F(\mathcal{V}_j)} \sum_{v_k \in V} \frac{r(\mathcal{V}_i, v_k)}{|F(\mathcal{V}_j)|} \cdot \pi(v_k, v_t).$$

As Algorithm 1 terminates when the residue value $r(\mathcal{V}_i, v_k) \leq r_{max} \cdot d(v_k)$ for all v_k, the error term on the RHS can be bounded by

$$\frac{r_{max}}{|F(\mathcal{V}_j)|} \cdot \sum_{v_t \in F(\mathcal{V}_j)} \sum_{v_k \in V} d(v_k) \cdot \pi(v_k, v_t).$$

Thus, $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ is (ϵ, δ)−approximate if we set

$$r_{max} = \frac{|F(\mathcal{V}_j)| \cdot \epsilon \cdot \delta}{\sum_{v_t \in F(\mathcal{V}_j)} \sum_{v_k \in V} d(v_k) \cdot \pi(v_k, v_t)}.$$

Based on Eq. (5), for a target $\mathcal{V}_j \in \mathcal{S}$, the correctness of FPSN is guaranteed by setting

$$r_{max} = \frac{\epsilon \cdot \delta}{m \cdot \tau_j},$$

where $\tau_j = \sum_{v_s \in F(\mathcal{V}_j)} \rho_s / |F(\mathcal{V}_j)|$. With $r_{max} = \frac{\epsilon \cdot \delta}{m \cdot \tau}$, FPSN is (ϵ, δ)−approximate for target supernode $\mathcal{V}_j \in \mathcal{S}$ with $\tau_j \leq \tau$. The proof completes.

Proof of Lemma 6.3 Similar to the invariant property of conventional Backward-Push [58], BPSN has the following property.

Proposition 1. BPSN *satisfies the following invariant:*

$$\pi_d(v_s, \mathcal{V}_j) = \widehat{\pi}_d(v_s, \mathcal{V}_j) + d(v_s) \cdot \sum_{v_k \in V} \pi(v_s, v_k) \cdot r(v_k, \mathcal{V}_j). \quad (10)$$

Based on Eq.(10) and $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) = \sum_{v_s \in F(\mathcal{V}_i)} \frac{\widehat{\pi}_d(v_s, \mathcal{V}_j)}{|F(\mathcal{V}_i)|}$, the approximation error is

$$\pi_d(\mathcal{V}_i, \mathcal{V}_j) - \widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) = \sum_{v_s \in F(\mathcal{V}_i)} \frac{d(v_s)}{|F(\mathcal{V}_i)|} \cdot \sum_{v_k \in V} \pi(v_s, v_k) \cdot r(v_k, \mathcal{V}_j).$$

Note that $\sum_{v_k \in V} \pi(v_s, v_k) = 1$ and BPSN stops when $r(v_k, \mathcal{V}_j) \leq r'_{max}$ for all $v_k \in V$. Hence, the error term on the RHS is upper-bounded by

$$\sum_{v_s \in F(\mathcal{V}_i)} \frac{d(v_s) \cdot r'_{max}}{|F(\mathcal{V}_i)|}.$$

For a source supernode $\mathcal{V}_i$, the approximation quality is met with

$$r'_{max} = \frac{\epsilon \cdot \delta}{d_i}.$$

Hence, correctness is guaranteed to all source supernodes $\mathcal{V}_i \in F(\mathcal{S})$ by setting $r'_{max} = \frac{\epsilon \cdot \delta}{d_{max}}$ where $d_{max} = \max_{\mathcal{V}_i \in \mathcal{S}} d_i$.

Proof of Theorem 6.6 Recall that the input values of sampling phase satisfy (see Lemma 6.1)

$$\pi_d(\mathcal{V}_i, \mathcal{V}_j) = \widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) + \sum_{v_t \in F(\mathcal{V}_j)} \sum_{v_k \in V} \frac{r(\mathcal{V}_i, v_k)}{|F(\mathcal{V}_j)|} \cdot \pi(v_k, v_t).$$

For any sampled node v_k, let $X_{k,j}$ be a Bernoulli variable that takes value 1 if the random walk starting from v_k stops at any node of $F(\mathcal{V}_j)$. By definition,

$$\mathbb{E}[X_{k,j}] = \sum_{v_t \in F(\mathcal{V}_j)} \pi(v_k, v_t).$$

Recall that v_k is selected with probability of $\frac{r(\mathcal{V}_i, v_k)}{r_{sum}}$ and the estimation is refined by $\frac{r_{sum}}{\omega \cdot |F(\mathcal{V}_j)|}$, hence,

$$\mathbb{E}[\frac{1}{\omega} \sum^{\omega} \frac{r_{sum}}{|F(\mathcal{V}_j)|} \cdot \frac{r(\mathcal{V}_i, v_k)}{r_{sum}} \cdot X_{k,l}] = \sum_{v_t \in F(\mathcal{V}_j)} \frac{r(\mathcal{V}_i, v_k)}{|F(\mathcal{V}_j)|} \cdot \pi(v_k, v_t).$$

Denote $\psi_k = \frac{r(\mathcal{V}_i, v_k)}{|F(\mathcal{V}_j)|} \cdot \pi(v_k, v_t)$, then

$$\mathbb{E}[\sum_{v_k \in V} \psi_k] = \sum_{v_t \in F(\mathcal{V}_j)} \sum_{v_k \in V} \frac{r(\mathcal{V}_i, v_k)}{|F(\mathcal{V}_j)|} \cdot \pi(v_k, v_t).$$

which is exactly the second term of Lemma 6.1. Thus, it is an unbiased estimator of $\pi_d(\mathcal{V}_i, \mathcal{V}_j)$.

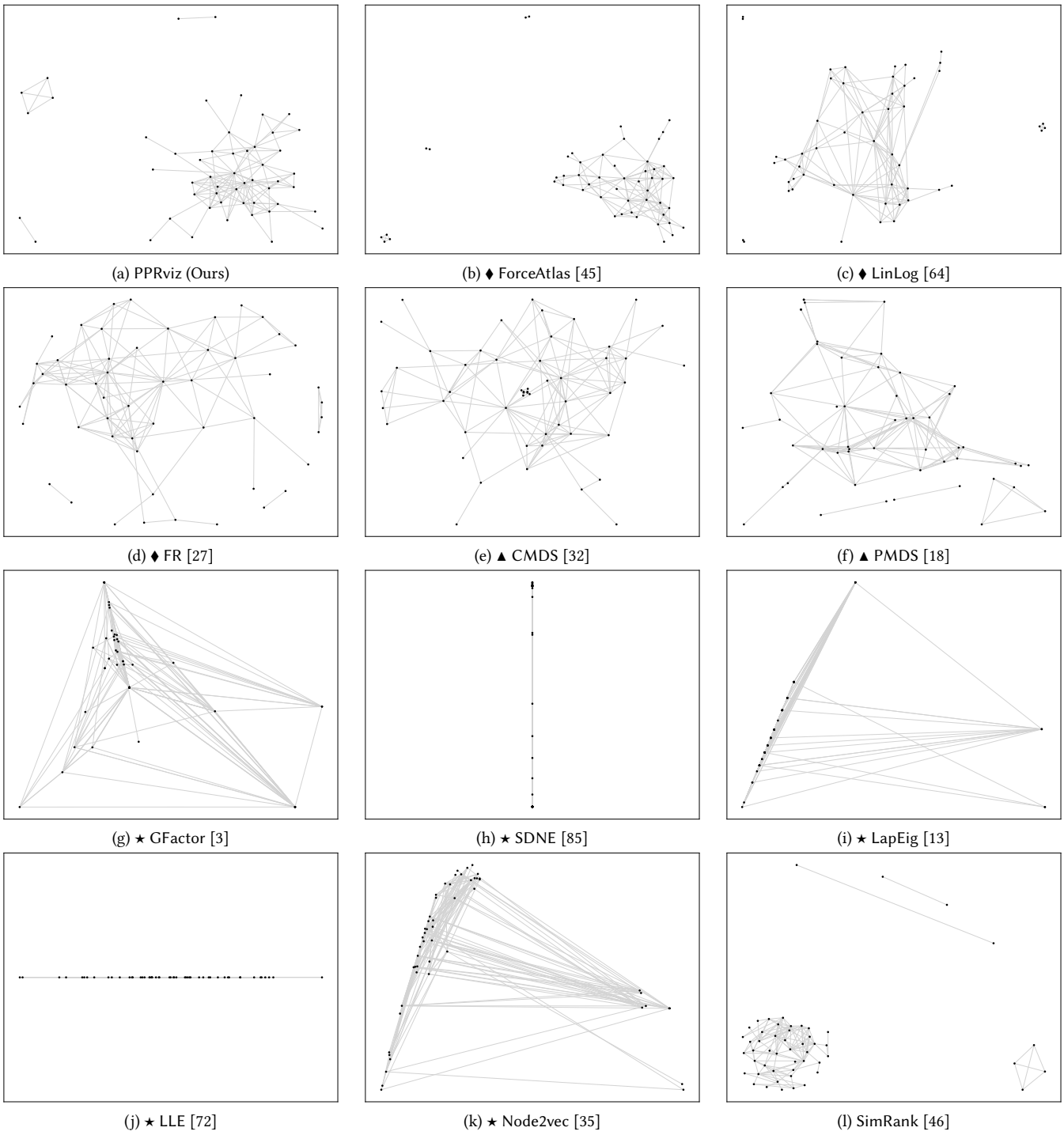
Next, by applying Chernoff Bound [20], we can derive that, for any $\mathcal{V}_i, \mathcal{V}_j \in \mathcal{S}$,

- when $\pi_d(\mathcal{V}_i, \mathcal{V}_j) < \delta$, by setting $\omega = \frac{r_{sum}}{\gamma} \cdot W$, we have

$$\Pr[|\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) - \pi_d(\mathcal{V}_i, \mathcal{V}_j)| \geq \epsilon \cdot \delta]$$
$$\leq \exp(-\frac{\omega \cdot (\epsilon \cdot \delta)^2}{\frac{r_{sum}}{|F(\mathcal{V}_j)|}(\frac{2}{3}(\epsilon \cdot \delta) + 2\delta)}) \leq p_f.$$

- when $\delta \leq \pi_d(\mathcal{V}_i, \mathcal{V}_j)$, by setting $\omega = \frac{r_{sum}}{\gamma} \cdot W$, we have

$$\Pr[|\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) - \pi_d(\mathcal{V}_i, \mathcal{V}_j)| \geq \epsilon \cdot \pi_d(\mathcal{V}_i, \mathcal{V}_j)]$$
$$\leq \exp(-\frac{\omega \cdot (\epsilon \cdot \pi_d(\mathcal{V}_i, \mathcal{V}_j))^2}{\frac{r_{sum}}{|F(\mathcal{V}_j)|}(\frac{2}{3}(\epsilon \cdot \pi_d(\mathcal{V}_i, \mathcal{V}_j)) + 2\pi_d(\mathcal{V}_i, \mathcal{V}_j))}) \leq p_f.$$



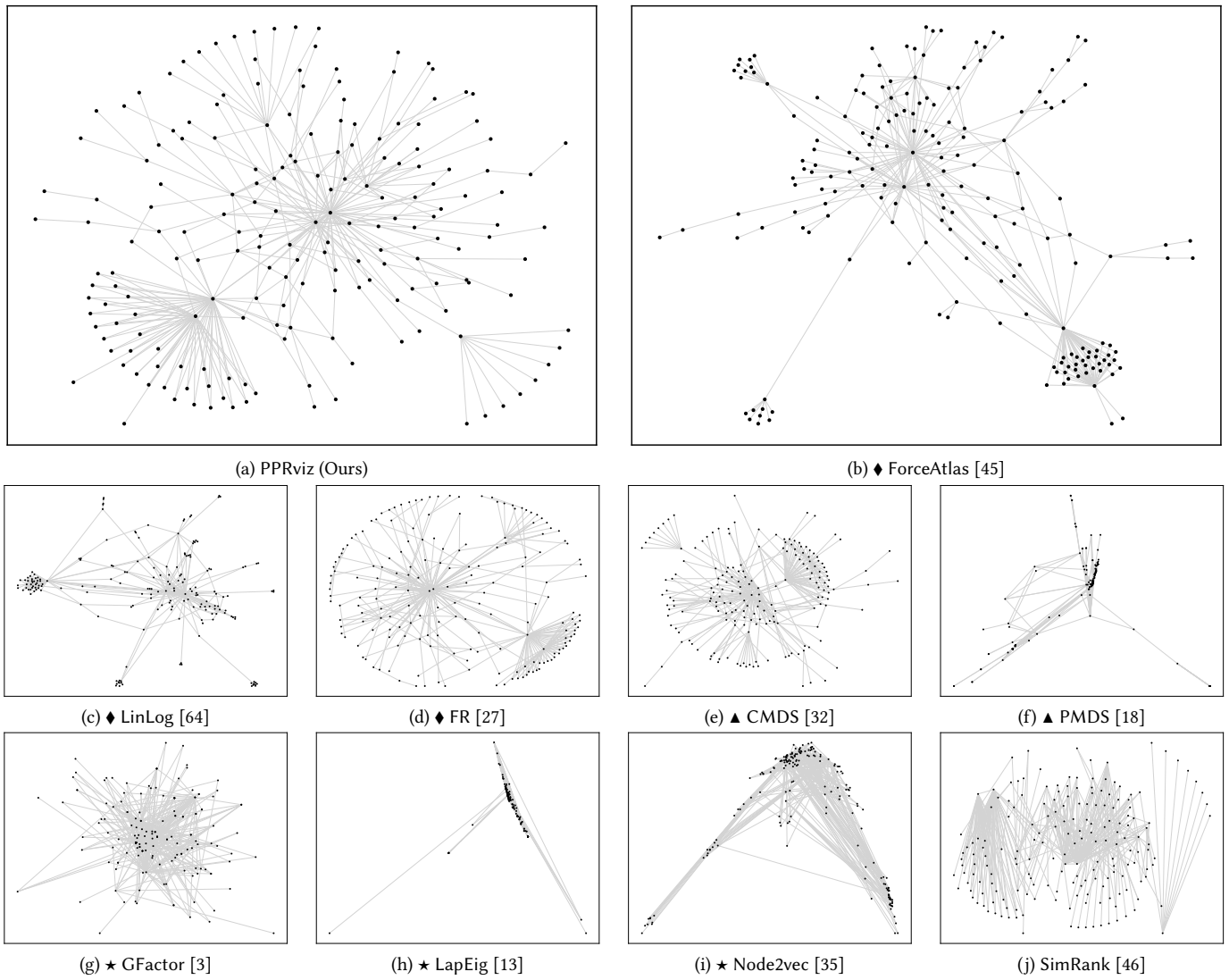
(a) PPRviz (Ours) (b) ♦ ForceAtlas [45] (c) ♦ LinLog [64]

(d) ♦ FR [27] (e) ▲ CMDS [32] (f) ▲ PMDS [18]

(g) ★ GFactor [3] (h) ★ SDNE [85] (i) ★ LapEig [13]

(j) ★ LLE [72] (k) ★ Node2vec [35] (l) SimRank [46]

Figure 9: Visualization results for the *FbEgo* graph: force-directed methods are marked with ♦; stress methods are marked with ▲; graph embedding methods are marked with ★.

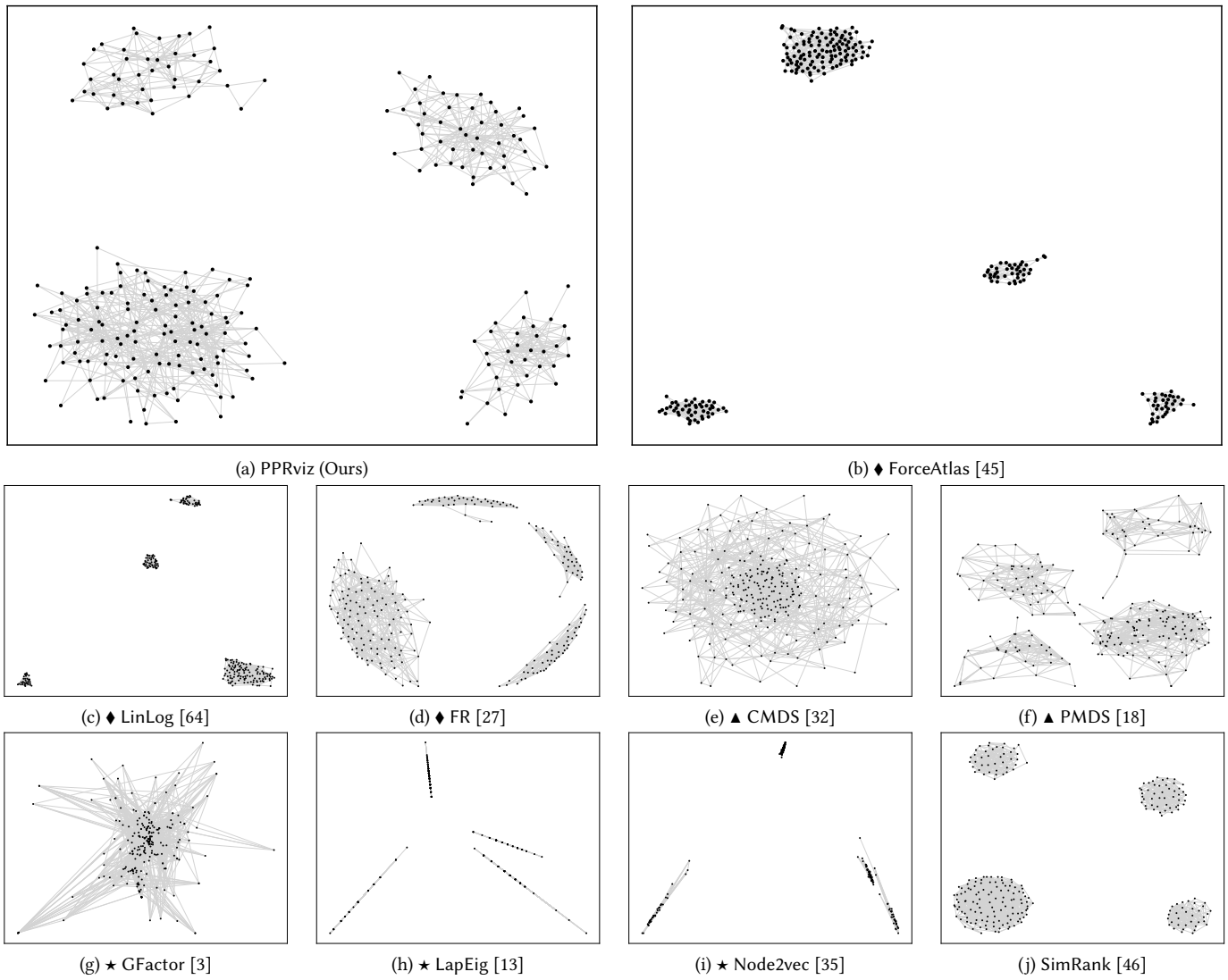


(a) PPRviz (Ours) (b) ♦ ForceAtlas [45]

(c) ♦ LinLog [64] (d) ♦ FR [27] (e) ▲ CMDS [32] (f) ▲ PMDS [18]

(g) ⋆ GFactor [3] (h) ⋆ LapEig [13] (i) ⋆ Node2vec [35] (j) SimRank [46]

Figure 10: Visualization results for the *Wiki-ii* graph: force-directed methods are marked with ♦; stress methods are marked with ▲; graph embedding methods are marked with ⋆.

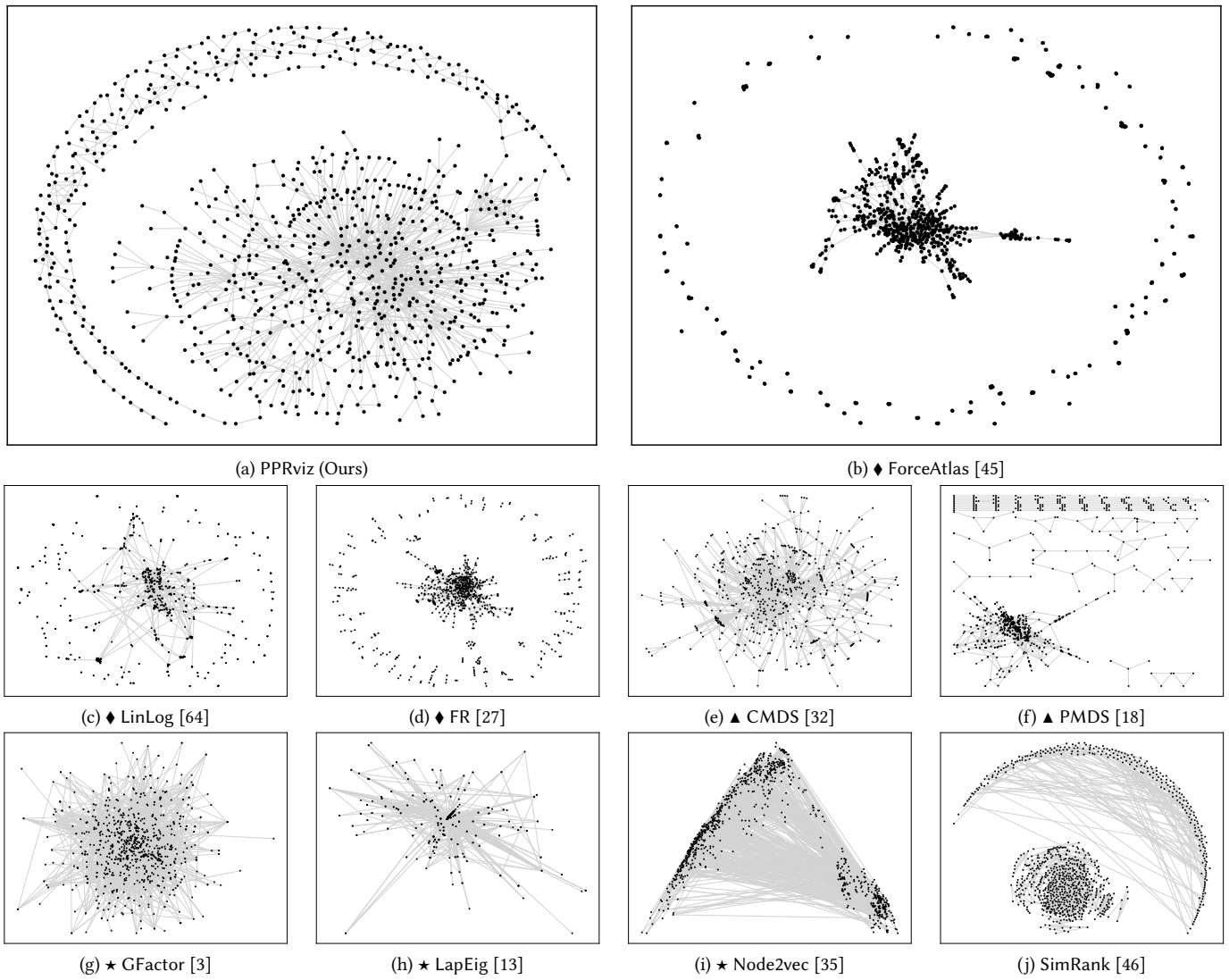


(a) PPRviz (Ours) (b) ♦ ForceAtlas [45]

(c) ♦ LinLog [64] (d) ♦ FR [27] (e) ▲ CMDS [32] (f) ▲ PMDS [18]

(g) ⋆ GFactor [3] (h) ⋆ LapEig [13] (i) ⋆ Node2vec [35] (j) SimRank [46]

Figure 11: Visualization results for the *Physician* graph: force-directed methods are marked with ♦; stress methods are marked with ▲; graph embedding methods are marked with ⋆.

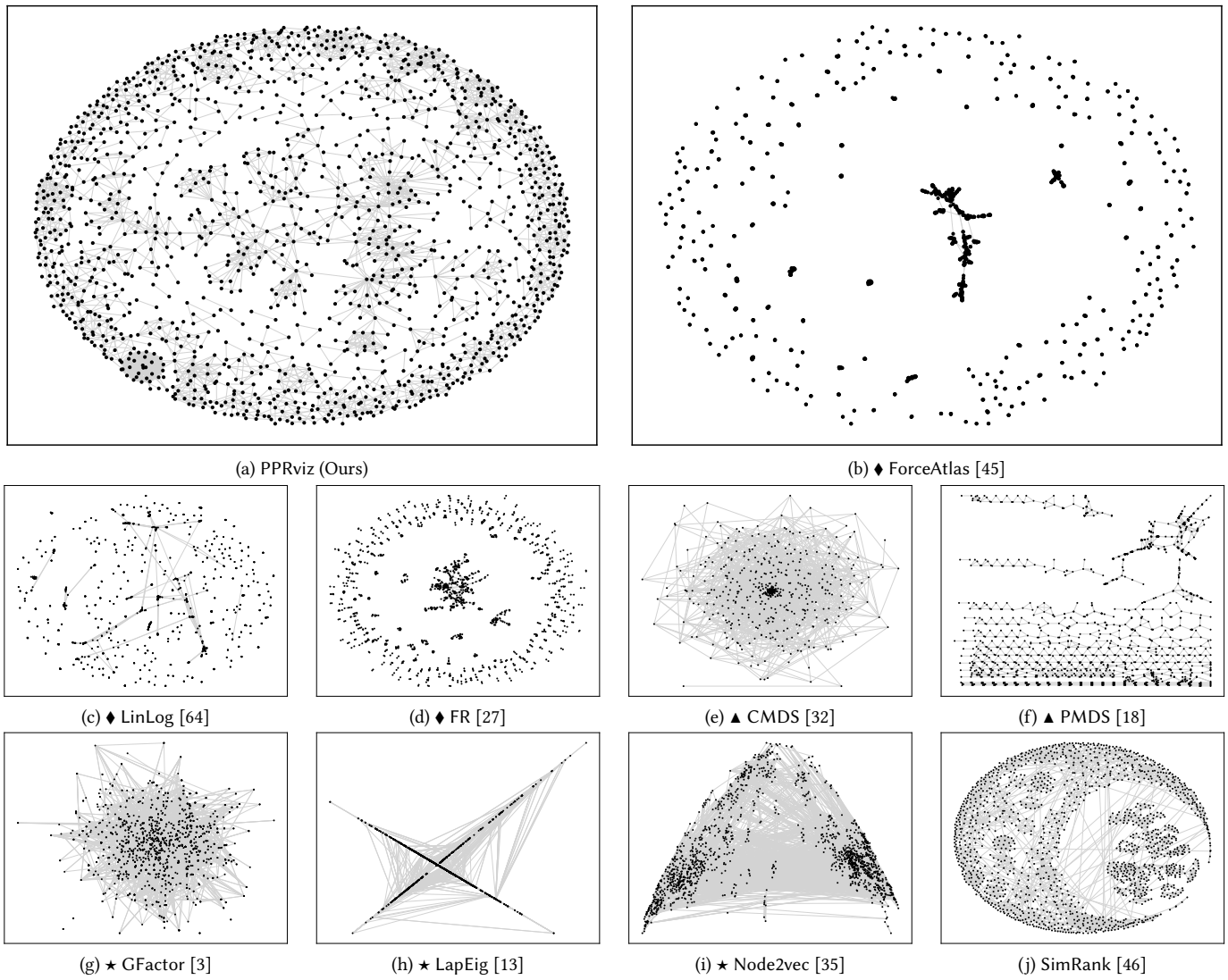


(a) PPRviz (Ours) (b) ♦ ForceAtlas [45]

(c) ♦ LinLog [64] (d) ♦ FR [27] (e) ▲ CMDS [32] (f) ▲ PMDS [18]

(g) ⋆ GFactor [3] (h) ⋆ LapEig [13] (i) ⋆ Node2vec [35] (j) SimRank [46]

Figure 12: Visualization results for the *FilmTrust* graph: force-directed methods are marked with ♦; stress methods are marked with ▲; graph embedding methods are marked with ⋆.



(a) PPRviz (Ours) (b) ♦ ForceAtlas [45]

(c) ♦ LinLog [64] (d) ♦ FR [27] (e) ▲ CMDS [32] (f) ▲ PMDS [18]

(g) ⋆ GFactor [3] (h) ⋆ LapEig [13] (i) ⋆ Node2vec [35] (j) SimRank [46]

Figure 13: Visualization results for the *SciNet* graph: force-directed methods are marked with ♦; stress methods are marked with ▲; graph embedding methods are marked with ⋆.