PPRviz: Effective and Efficient Graph Visualization based on Personalized PageRank

[Technical Report]

Shiqi Zhang
National University of Singapore
s-zhang@comp.nus.edu.sg

Renchi Yang
National University of Singapore
renchi@nus.edu.sg

Xiaokui Xiao
National University of Singapore
xkxiao@nus.edu.sg

Xiao Yan
Southern University of Science and Technology
yanx@sustech.edu.cn

Bo Tang
Southern University of Science and Technology
tangb3@sustech.edu.cn

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 real-world graphs in terms of both effectiveness and efficiency, and that PPRviz provides interactive visualizations within one second on billion-edge graphs.

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 [41], and road network design [39]. This motivates a plethora of graph visualization solutions [1, 6–8, 19, 24, 26, 29, 32–34, 44, 47, 51, 60, 65, 72, 77, 82] and software [9, 11, 20, 27].

Given a graph $G = (V, E)$ with $n$ nodes, the primary objective of existing solutions is to find two-dimensional coordinates for the nodes in $V$, such that the coordinates on the screen reflect the 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 existing methods [19, 24, 26, 29, 32–34, 44, 47, 51, 65, 77, 82] 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 clustering of nodes, and it incurs significant overheads as it requires computing the coordinates of a huge number of nodes. Multi-level methods [1, 7, 8, 12, 31, 44, 60, 61, 72, 75, 86] mitigate the above issue by recursively partitioning the input graph $G$ into a hierarchy of supergraphs, where each supernode in the level-$\ell$ 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 $\Delta \in \mathbb{R}^{k \times k}$ 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 $\Delta$ are preserved as much as possible. Therefore, the quality of the visualization highly depends on how $\Delta$ 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, 7, 8, 31, 60, 86] derive node distances based only on direct connections among nodes, ignoring all indirect connections via multiple hops. Furthermore, the methods in [61, 75] 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 $V_i$ and $V_j$, they require deriving either the distance between each pair of leaf nodes under $V_i$ and $V_j$ [61], or the coordinate of each such leaf node [60]. 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.
(PPR) [67], 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 distorted edges, and it significantly outperforms competitors in terms of aesthetic metrics and user studies. 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 the 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 circles and edges with lines [19, 26, 29, 33, 34, 47, 51, 60, 77]. Specifically, we place the center of a circle on the coordinate of a node, and draw directed edge using an arrowhead straight line. For undirected edges, we omit the arrowhead to avoid cluttered display. Thus, the layout of a graph $G$ is defined by a position matrix $X \in \mathbb{R}^{n \times 2}$, where $X[i] \in \mathbb{R}^2$ (i.e., the $i$-th row of $X$) records the coordinates of node $v_i \in V$ in the two-dimensional space, and $||X[i] - X[j]||$ 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 [13, 35, 83]. In particular, the position matrix $X$ 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 [15, 53, 66, 69, 70, 80], i.e., node distribution (ND) and uniform length coefficient variance (ULCV), as defined in the following.

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

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 (occupying same node positions) lead to an infinite ND score.

Definition 2.2 (ULCV). For a position matrix $X$, let $l_x$ (resp. $l_y$) be the standard deviation (resp. mean) of the edge lengths. The uniform length coefficient variance is $\text{ULCV}(X) = l_x/l_y$.

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 [35]. For this reason, some existing works [1, 7, 8, 12, 72, 75] proposed to interactively visualize large graphs in a multi-level manner to reduce the number of nodes in each visualization. As surveyed in [42, 85], multi-level methods consist of two phases: (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 node is at level-0 and that each non-leaf node is at level-$(l + 1)$ if its children are at level-$l$ ($l \geq 0$). During the interactive visualization phase, the user can select any supernode $S$ at level-$(l + 1)$, and ask for a visualization of the children of $S$.

For such visualizations, the key issue is that we need to carefully decide the graph layout matrix for the child nodes of $S$, especially when each child node is the supernode. In relation to this, a canonical approach is to measure the distance between two supernodes $V_i$ and $V_j$ based on the average distance between every node pair $v_i$ and $v_j$ [60, 78], where $v_i$ (resp. $v_j$) is a leaf node under $V_i$ (resp. $V_j$). This approach, however, presents a significant challenge in
3 SOLUTION OVERVIEW

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

- **Supergraph hierarchy construction:** Following prior works [1, 7, 8, 71, 75], PPRviz first constructs a supergraph hierarchy of the input graph in the preprocessing step. Take Figure 1(a) as an example. The bottom layer is the input graph, whose non-overlapping node partitions are organized as level-1 supernodes in the middle layer. Similarly, the clustering procedure is repeated to generate the top layer. To construct the supergraph hierarchy, we propose Louvain++, adopted from the community detection algorithm [17], to iteratively group nodes or supernodes at level- \( \ell \) of the hierarchy into supernodes at level- \((\ell + 1)\), under the constraint that each supernode \( S \) (resp. the coarsest supergraph) should have at most \( k \) children (resp. supernodes). This constraint ensures that the visualization of \( S \) ’s children does not contain too many nodes, which helps to avoid visual clutter [4, 25]. Note that \( k \) can be configured by users according to their needs.

- **Node distance matrix computation:** During interactive visualization, a user can select a level- \((\ell + 1)\) supernode \( S \) (marked by an arrow in Figure 1(b)). To visualize the intra-structure of \( S \), i.e., the shaded area in the middle layer, PPRviz calculates a node distance matrix \( \Delta \in \mathbb{R}^{k \times k} \) for the \( k \) children of \( S \) (Figure 1(c)), which utilizes information of related nodes in the original input graph, i.e., the shaded area in the bottom layer of Figure 1(b). In the degenerated case when the supergraph hierarchy has only one level (i.e., when the input graph is small), PPRviz would compute a matrix \( \Delta \in \mathbb{R}^{k \times k} \) for visualizing the entire graph.

- **Position matrix embedding:** Given the node distance matrix \( \Delta \in \mathbb{R}^{k \times k} \), PPRviz converts it into a position matrix \( X \in \mathbb{R}^{k \times 2} \) (see Figure 1(d)), by solving the following optimization problem [34]:

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

That is, it aims to ensure that the Euclidean distance \( ||X[i] - X[j]|| \) derived from the position matrix \( X \) is close to the node distance \( \Delta[i,j] \). Towards this end, PPRviz employs the standard method for solving Eq. (1), namely, the stress majorization technique [34]. The time complexity of this method is \( O(k^3) \) [34], which is insignificant as \( k \) is usually small to avoid visual clutter.

In what follows, we elaborate the node distance measure adopted in PPRviz and the algorithms used for the node distance matrix computation. Interested readers are referred to Appendix A for the details of PPRviz’s Louvain++ and stress majorization.

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 present 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 \( \alpha \), 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) [81] 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 \( \alpha \), 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 \( \Delta \in \mathbb{R}^{n \times n} \) be the PDist matrix for all node pairs in a graph \( G \) and \( \Delta[i,j] \) be the PDist between nodes \( v_i \) and \( v_j \). We define \( \Delta[i,j] \) as

\[
\Delta[i,j] = 1 - \log \left( \pi_d(v_i, v_j) + \pi_d(v_j, v_i) \right),
\]

where \( \pi_d(v_i, v_j) = \pi(v_i, v_j) \cdot d(v_i) \) represents the degree-normalized PPR (DPRR) 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 \((v_0, v_9)\), \((v_2, v_0)\), and \((v_6, v_7)\). Observe that \(\pi(v_0, v_9) = 0.11 < \pi(v_2, v_0) = 0.44\), even though node pairs \((v_2, v_0)\) and \((v_6, v_7)\) are both directly connected 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 [93]. As a consequence, directly transforming PPR values into node distances would lead to a large variance in edge lengths in graph visualization. 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 [93] 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 \(v_0\) are direct neighbors of \(v_6\) and \(v_9\), respectively. Meanwhile, the PDist of \((v_6, v_7)\) is large, which reflects the fact that \(v_0\) is far away from \(v_9\) 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)), since \(\pi_d(v_i, v_j) \neq \pi_d(v_j, v_i)\) in general, whereas graph visualizations require a symmetric distance. Moreover, PDist takes the inverse of the natural logarithm of DPPR, so that it can produce a smaller distance for the node pair with better connectivity. For better visualization quality, we empirically truncate Eq. (2) to the range of \([2, 2\log n]\). In particular, we set the lower bound to 2 to avoid the negative value of Eq.(2) and node overlapping. Note that the average PPR value between a node pair is \(1/n\) for a graph with \(n\) nodes [10, 74, 90]. To avoid excessive blank space between nodes, we focus on the distance between nodes with above-average PPR values and set the upper bound to \(2\log n\).

**Bounds for aesthetic criteria.** The following theorems 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 \(\Delta\) of graph \(G\), suppose that \(|X[i] - X[j]| = \Delta[i, j]\) for \(v_i, v_j \in V\), then \(ND(X) \leq \frac{2(n - 1)}{2}.\)

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

Note that the upper bound of the restart probability \(\alpha\) in Theorem 4.3 (which is approximately 0.243) is not restrictive, as \(\alpha\) is usually set to 0.15 or 0.2 [10, 49, 58, 88, 89, 92].

### 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 [47], LinLog [65] and FR [29]); (ii) 2 stress methods (i.e., CMDS [34] and PMDS [19]); (iii) 5 graph embedding methods (i.e., GFactor [3], SDNE [87], LapEig [14], LLE [73] and Nodevec [37]); as well as (iv) a variant of PPRviz using SimRank [48] in PDist. The visualization results of a real-world graph Twitter 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, as explained in the following. Regarding more visualizations, please see Appendix A.

**Force-directed methods** model a graph as a force system, where adjacent nodes attract each other and all nodes repulse each other [22, 26, 29, 47, 60, 65]. 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. 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 [19, 32–34, 51, 52, 61, 77, 82]. In particular, they embed the pairwise shortest distance matrix into a two-dimensional coordinate matrix by optimization techniques, e.g., gradient descent [33] and stress majorization [34], such that 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 and drag down the readability. Specifically, CMDS and PMDS suffer from severe node overlapping, as illustrated in Figures 3(e) and 3(f). In addition, the node clusters are not as clear as PPRviz.

**Graph embedding methods** have been adapted for graph visualization in [36]. Following [36, 68, 79], we set the embedding dimension to 2 to obtain the position matrix \(X\). Figures (g)–(k) report the visualization results of 5 widely-adopted graph embedding methods [3, 14, 37, 73, 87] on the Twitter 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 any cluster structures. This is due to that graph embedding methods are designed for node classification or link prediction tasks, which requires similar nodes close to each other in the embedding space, without the consideration of any visualization quality.

**SimRank** [48] 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 are zero.

5 PPR-BASED SUPERNODE DISTANCE

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

5.1 Level-\(t\) \(PDist\) Definition

We first extend \(DPPR\) in Definition 4.1 to level-\(t\) \(DPPR\) (Definition 5.1), and derive level-\(t\) \(PDist\) by plugging Eq. (3) into Eq. (2).

**Definition 5.1 (Level-\(t\) \(DPPR\)).** For two level-\(t\) supernodes \(V'_{i}\) and \(V'_{j}\), denote the set of leaf nodes in \(V'_{i}\) as \(F(V'_{i})\), the level-\(t\) \(DPPR\) \(\pi_d(V'_{i}, V'_{j})\) of \(V'_{j}\) w.r.t. \(V'_{i}\) is defined as

\[
\pi_d(V'_{i}, V'_{j}) = \frac{\sum_{v_s \in F(V'_{i}), v_t \in F(V'_{j})} \pi_d(v_s, v_t)}{|F(V'_{i})||F(V'_{j})|},
\]

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

Intuitively, the level-\(t\) \(DPPR\) measures the connectivity from supernode \(V'_{i}\) to \(V'_{j}\) by taking the average of the \(DPPR\) values from the leaf nodes in \(V'_{i}\) to those in \(V'_{j}\). The idea that measure two supernodes by summarizing the structure of underlying leaf nodes is also adapted in [30, 60, 78, 84]. In particular, [60, 78] also consider the average of all pairwise distances of leaf nodes.

Compared with our level-\(t\) \(DPPR\), a simple and straightforward way is to take the level-\(t\) children of supernode \(S\) as a weighted graph and compute the \(DPPR\), called W-\(DPPR\), on it. More precisely, the graph is constructed by treating each supernode \(V'_{i} \in S\) as a node and merging edges between supernodes \(V'_{i}\) and \(V'_{j}\) in \(S\) as a weighted edge, regardless of the micro-structure of each supernode as well as edges linking nodes outside \(S\). 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 results in a 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 \(S\) with level-1 supernodes \(V_0, V_2, V_5\) as well as the level-\(t\) \(DPPR\) (\(\ell\)-\(DPPR\) in short) and W-\(DPPR\) values for supernode pairs \((V_1, V_0), (V_2, V_0)\) and \((V_2, V_1)\). Intuitively, for source supernode \(V_2\) \(V_1\) has better connectivity to it than \(V_0\) as the children of \(V_2\) and \(V_0\) share one common neighbor \(v_2\), whereas the children of \(V_2\) and \(V_1\) have two common neighbors \(v_3\) and \(v_4\). From the table in Figure 4, we observe that the W-\(DPPR\) values of \((V_2, V_0)\) and \((V_2, V_1)\) are equal, which is counter-intuitive. In contrast, level-\(t\) \(DPPR\) addresses this issue and accurately captures the structure of the original graph.

5.2 Efficiency Challenges

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

**Definition 5.2 (\((\theta, \sigma)\)-approximate level-\(t\) \(PDist\)).** Let \(\theta\) and \(\sigma\) be two constants, for any two supernodes \(V'_{i}, V'_{j} \in S\) and \(V'_{i} \neq V'_{j}\), \(\tilde{\Delta}[i, j]\) is a \((\theta, \sigma)\)-approximate level-\(t\) \(PDist\) \(\Delta[i, j]\) if it satisfies

- If \(\Delta[i, j] < \sigma\), \(|\Delta[i, j] - \tilde{\Delta}[i, j]| \leq \theta \cdot \Delta[i, j]\).
- If \(\Delta[i, j] \geq \sigma\), \(|\Delta[i, j] - \tilde{\Delta}[i, j]| \leq \theta \cdot \sigma\).

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

**Definition 5.3 ((\(\epsilon, \delta)\)-approximate level-\(t\) \(DPPR\)).** Let \(\epsilon\) and \(\delta\) be two constants, for any two supernodes \(V'_{i}, V'_{j} \in S\) and \(V'_{i} \neq V'_{j}\), \(\tilde{\pi}_d(V'_{i}, V'_{j})\) is an \((\epsilon, \delta)\)-approximate of level-\(t\) \(DPPR\) \(\pi_d(V'_{i}, V'_{j})\) if it satisfies the following conditions.

- If \(\pi_d(V'_{i}, V'_{j}) < \delta\), \(|\tilde{\pi}_d(V'_{i}, V'_{j}) - \pi_d(V'_{i}, V'_{j})| \leq \epsilon \cdot \delta\).
- If \(\pi_d(V'_{i}, V'_{j}) \geq \delta\), \(|\tilde{\pi}_d(V'_{i}, V'_{j}) - \pi_d(V'_{i}, V'_{j})| \leq \epsilon \cdot \pi_d(V'_{i}, V'_{j})\).
Lemma 5.4. With $\epsilon = 1 - \left( \frac{\pi_d(V_i, V_j) + \pi_d(V_j, V_i)}{\epsilon} \right)^{2\theta}$ and $\delta = \frac{e^\theta - 2}{4}$, $(\epsilon, \delta)$-approximate level-$t$ DPPR ensures that level-$t$ PDist is $(\theta, \sigma)$-approximate.

Lemma 5.4 indicates that the error bound $\epsilon$ 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-$t$ DPPR, which can be solved by extending the PPR approximation methods [5, 16, 28, 40, 43, 59, 74, 76, 88–92, 94].

**Deficiencies of existing solutions.** To estimate level-$t$ DPPR values w.r.t a source supernode $V_i$, a straightforward solution is to perform the well-studied single source PPR (SSPPR) approximation methods [5, 16, 28, 40, 43, 74, 90, 92] from each single leaf node in $V_i$ individually and combine their results in a degree-weighted fashion, which is rather inefficient. For instance, the level-$t$ supernode $V_i$ needs an $O(k^2n)$ cost to repeatedly invoke SSPPR queries from all $O(k^t)$ leaf nodes, and even an $O(mn)$ cost at the highest level. Moreover, most of the state-of-the-art methods [43, 74, 90, 92] are built on Forward-Push [5]. 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:

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

where $\pi_d(v_i, v_j)$ 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 $(\epsilon, \delta)$-approximation guarantee, existing methods either rely on numerous push operations to deplete residues or estimate the error term based on costly random walks. However, most of these operations are unnecessary since a large fraction of visited nodes are not in the interested supernode $S$. Particularly, when visualizing the $S$ at level-1, only $O(k/n)$ of random walk samples have their end nodes in $S$.

**6 Tau-Push ALGORITHM**

Motivated by the inefficiency issues, we present Tau-Push for level-$t$ DPPR approximation. We introduce the main idea of Tau-Push in Section 6.1, followed by the details 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-$t$ DPPR computation, and compare them with Tau-Push in Section 6.4.

**6.1 Overview**

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

The basic idea is to perform push operations [5, 59] (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 $S$, Tau-Push first invokes a core subroutine called Forward-Push from SuperNode (FPSN) to compute approximate level-$t$ DPPR values for each source supernode $V_i \subseteq S$ (Lines 1-2). After that, Tau-Push proceeds to refine the approximate level-$t$ 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 [5]. Recall from Section 5.2 that directly utilizing Forward-Push for level-$t$ DPPR approximation entails immensely expensive costs. Distinct from Forward-Push, FPSN starts push operations from all leaf nodes in the level-$t$ supernode simultaneously with the consideration of their degrees, such that $O(k^t)$ 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:

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

which essentially is a weighted global PageRank [67]. 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 $d(v_k) / m$. Note that if we perform push operations until every $r(v_i, v_k) \leq d(v_k) \cdot r_{\max}$, the error term in Eq. (4) is bounded by

$$\pi_d(v_i, v_j) - \pi_d(v_i, v_j) \leq m \cdot \rho_j \cdot r_{\max}.\quad (5)$$

Therefore, $(\epsilon, \delta)$-approximate DPPR $\pi_d(v_i, v_j)$ can be satisfied for a given $v_j$ by setting $r_{\max} = \frac{\epsilon \delta}{m \cdot \rho_j}$. For the level-$t$ supernode, it can be achieved by setting the initial residue of each leaf node to a well-designed value and extending DPR $\rho_j$ in a multi-level manner (i.e., $\tau_j$ in Algorithm 1), which are elaborated in Section 6.2.

Based on the above analysis, FPSN can return $(\epsilon, \delta)$-approximate DPPR $\pi_d(v_i, v_j)$ for each target node $v_j \in S' \subseteq S$ at level-0 by setting $r_{\max} = \frac{\epsilon \delta}{m \cdot \rho_{\max}}$ and $\rho_{\max} = \max_{v_i \in S'} (S' \rho_j)$. Therefore, the termination of FPSN can be facilitated by pre-computing DPR values, which only requires $O(m)$ running time [67] and $O(n)$ storage space. However, the cost of FPSN is dominated by the largest DPR value in $S$, which can be extremely large as the DPR values of scale-free networks are highly skewed [57]. 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 $S$ containing the largest DPR of $G$, the node with a small DPR value will keep

| Algorithm 1: Tau-Push |
|-----------------------|
| **Input:** Graph $G$, supernode $S$, constants $r_{\max}, r'_{\max}, \tau$. |
| **Output:** Estimated $\pi_d(V_i, V_j)$, $\forall V_i, V_j \in S$ and $V_i \neq V_j$. |
| 1. for each $V_i \in S$ do |
| 2. $\forall V_j \in S, V_j \neq V_i$, $\pi_d(V_i, V_j) \leftarrow$ FPSN ($V_i, r_{\max}, \tau$); |
| 3. for $V_j \in S$ where $\tau_j > \tau$, $\tau_j = \frac{\sum_{v_i \in V_j} \rho_{\max}}{|P(V_j)|}$ do |
| 4. $\forall V_i \in S, V_i \neq V_j$, $\pi_d(V_i, V_j) \leftarrow$ BPSN ($V_j, r'_{\max}$); |
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 [59] 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 \( v_j \in S \) at level-0, BPSN stops the reverse push when every \( r(v_k, v_j) \leq r'_{max} \) and returns \((\epsilon, \delta)\)-approximate DPR \( \bar{\pi}_d(V'_j, v_j) \) for each source leaf node \( v_i \in S \setminus v_j \), where \( r'_{max} = \frac{\epsilon \delta}{\delta_{max}} \) and \( \delta_{max} = \max_{v_i \in S \setminus v_j} d(v_i) \). 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 [57], 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 \( \bar{\pi}_d(V'_l, v_l) \), FPSN maintains two variables for each node \( v_k \in V \): (i) the reserve \( \bar{\pi}_d(V'_l, v_k) \), which is a lower bound of the partial level-\( t \) DPR

\[
\pi_d(V'_l, v_k) = \frac{\sum_{v_i \in F(V'_l)} \pi_d(v_i, v_k)}{|F(V'_l)|} ;
\]

(ii) the residue \( r(V'_l, v_k) \), which is a by-product. For initialization, the residue of each node \( v_j \in F(V'_l) \) is set to \( d(v_j)/d(V'_l) \) fraction of \( r_{init} \), where \( r_{init} = d(V'_l)/|F(V'_l)| \) and \( d(V'_l) \) is the sum of out-degrees of the leaf nodes in \( V'_l \). The reserve and residue values of the leaf nodes outside \( V'_l \) are all set to zero. During push operations, the reserve and residue 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 \( r(V'_l, v_k) \leq d(v_k) \cdot r'_{max} \) for all \( v_k \in V \), where \( r'_{max} \) is set to \( \frac{\epsilon \delta}{\delta_{max}} \). Finally, the approximation of partial level-\( t \) DPR, i.e., \( \bar{\pi}_d(V'_l, v_k) \), is aggregated into the approximation of level-\( t \) DPR in Line 9.

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

![Figure 5: The distributions of DPR on Youtube and Twitter.](image)

\[ \bar{\pi}_d(V'_j, v_j) \leq \sum_{v_i \in F(V'_j)} \bar{\pi}_d(v_i, v_j) + r(V'_j, v_j \cdot v_j) ; \]

\[ \pi(V'_j, v_j) = \sum_{v_i \in F(V'_j)} \pi(v_i, v_j) + (1 - \alpha \cdot r(V'_j, v_j)) ; \]

\[ r(V'_j, v_j) = 0 ; \]

Algorithm 2: FPSN

**Input:** Graph \( G \), source \( V'_l \), supernode \( S \), constant \( r'_{max} \), \( \tau \)

**Output:** Estimated DPR \( \bar{\pi}_d(V'_l, v_l), \forall v_j \in S \)

```
1 \( r_{init} \leftarrow \frac{d(v_j)}{|F(V'_l)|} \) where \( d(V'_l) = \sum_{v_i \in F(V'_l)} d(v_i) ; \)
2 \( r(V'_j, v_j), \bar{\pi}_d(V'_j, v_j) \leftarrow 0, \forall v_j \in V ; \)
3 \( r(V'_j, v_j) \leftarrow \frac{d(v_j)}{|F(V'_l)|} \cdot r_{init}, \forall v_j \in F(V'_l) ; \)
4 while \( \exists v_k \in V \) such that \( r(V'_j, v_k) > d(v_k) \cdot r'_{max} \) do
5 \( \bar{\pi}_d(V'_j, v_k) \leftarrow \bar{\pi}_d(v_i, v_k) + \alpha \cdot r(V'_j, v_k) ; \)
6 for each \( v_j \in N(v_k) \) do
7 \( r(V'_j, v_j) \leftarrow r(V'_j, v_j) + (1 - \alpha) \cdot r(V'_j, v_k) ; \)
8 \( \bar{\pi}_d(V'_j, v_j) \leftarrow \sum_{v_i \in F(V'_j)} \bar{\pi}_d(v_i, v_j) / |F(V'_j)| ; \)
9 \( \forall v_j \in S ; \)
```

**Lemma 6.1.** By initializing residue values as Lines 1-3, Algorithm 2 satisfies the following invariant:

\[ \pi_d(V'_l, v_k) = \frac{\sum_{v_i \in F(V'_l)} \pi_d(v_i, v_k)}{|F(V'_l)|} ; \]

\( \pi(V'_j, v_j) = \sum_{v_i \in F(V'_j)} \pi(v_i, v_j) / |F(V'_j)| ; \)

\( \pi(V'_l, v_k) = \sum_{v_i \in F(V'_l)} \pi(V'_i, v_k) ; \)

(ii) residue \( r(v_k, v_j) \), which is a by-product. The residue values of the nodes are initialized as Lines 1-2 in Algorithm 3, and backward push operation [59] is performed for a node \( v_k \) with \( r(v_k, V'_j) \) following

\[
\left\{ \begin{array}{l}
r(v_k, v_j) = r(v_k, v_j) + \frac{1 - \alpha}{d(v_k, V'_j)} \cdot r(v_k, V'_j), \forall v_j \in N(v_k) \\
n(V'_j, v_k) = \frac{1 - \alpha}{d(V'_j, v_k)} \cdot r(v_k, V'_j) + r(v_k, v_j) \end{array} \right.
\]

where \( N(v_k) \) is the set of in-neighbors of \( v_k \). FPSN stops when every \( r(v_k, v_j) \leq r'_{max} \) and \( r'_{max} \) is set as \( \frac{\epsilon \delta}{\delta_{max}} \), in which \( d_{max} = \max_{v_k \in S} d(v_k) \) and \( d_l = \sum_{v_i \in F(V'_j)} d(v_i) / |F(V'_l)| \); finally, the partial DPR \( \pi(V'_j, v_j) \) is assembled into the required approximate DPR \( \bar{\pi}_d(V'_j, v_j) \) in Line 9 of Algorithm 3.

Similar to the analysis of FPSN, the correctness of BPSN can be derived by employing the invariant in [59].

**Lemma 6.3.** Given a target supernode \( V'_j \in S \), by setting \( r'_{max} = \frac{\epsilon \delta}{\delta_{max}} \), Algorithm 3 returns \((\epsilon, \delta)\)-approximate level-\( t \) DPR \( \bar{\pi}_d(V'_j, V'_j) \) for \( V'_l \in S \).
6.3 Theoretic Analysis

Correctness. Note that Lemmas 6.2 and 6.3 ensure the correctness from any source \( V_1 \) to target supernodes \( V_j \) with \( t_j \leq \tau \) and \( t_j > \tau \), respectively. Hence, the following theorem shows the correctness of Tau-Push in Algorithm 1.

**Theorem 6.4.** For any user-selected supernode \( S \) and threshold \( \tau \), by setting \( r_{\text{max}} \) and \( r_{\text{max}}' \) as Lemmas 6.2 and 6.3 respectively, Algorithm 1 returns \((e, \delta)\)-approximate level-\( \ell \) DPPR \( \tilde{\pi}_d(V_i, V_j) \) for \( V_i, V_j \in S \) and \( \tilde{\pi}_d(V_i, V_j) \).

Time complexity. The worst-case time complexity of Tau-Push is \( O\left(\frac{\text{km}^{1+\frac{m}{e}} + d_{\text{max}} m}{e \cdot \text{con}}\right) \). By setting \( \tau = 1/\sqrt{kn} \), the above complexity is minimized to \( O\left(\frac{kn \cdot \sqrt{kn}}{\sqrt{kn}}\right) \). By employing FPSN only, the worst-case complexity is \( O\left(\frac{\text{km}^{1+\frac{m}{e}}}{e \cdot \text{con}}\right) \), which is \( \sqrt{kn} \) times slower than Tau-Push. Furthermore, the expected complexity of Tau-Push for a random selected supernode \( S \) is \( O\left(\sum_{V_i \in S} \frac{d(V_i)}{|F(V_i)|} \cdot \frac{kn}{\text{con}}\right) \). Due to space limitation, we refer interested readers to Appendix A for the detailed complexity analysis and \( \tau \) setting.

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

6.4 Comparing Tau-Push with Alternatives

**FORA**. We follow the vanilla solution as elaborated in Section 5.2, where \( \text{FORA} \) is invoked from each leaf node \( v_i \in \mathcal{V} \). Specifically, \( \text{FORA} \) first utilizes Forward-Push [5] to derive rough approximations of the DPPR values, and then estimates the error term in Eq. (4) by exploiting random walk samplings [28]. Adapting the conclusions in [90], we can show that \( \text{FORA} \) yields approximate DPPR for each \( v_i \in \mathcal{V} \).

**Lemma 6.5.** For each source node \( v_i \in V \), by setting the initial residue value as \( r(v_i, v_i) = d(v_i) \) and performing \( \omega = \text{rsum} \cdot W \) random walks, \( \text{FORA} \) returns \((e, \delta)\)-approximate DPPR \( \pi_d(v_i, v_i) \)

\( \text{FORA} \)'s time complexity is about 5 faster than \( \text{FORA} \)'s.
7 EXPERIMENT EVALUATION

In Section 7.1, we introduce the experiment settings, followed by the evaluation of visualization quality in terms of metrics and user studies in Sections 7.2 and 7.3 respectively. Furthermore, we compare the efficiency of PPRviz and other methods in Section 7.4, followed by the evaluation of PPRviz and its variants in Section 7.5.

7.1 Experiment Settings

Competitors and parameter settings. We compare PPRviz with 13 competitors from different categories. (i) 5 single-level graph layout methods: FR [29], LinLog [65], ForceAtlas [47], CMDS [34], PMDS [19]; (ii) 5 single-level graph embedding methods: GFactor [3], SDNE [87], LapEig [14], LLE [73], Node2vec [57]; (iii) an adaptation of PPRviz with SimRank [48] as introduced in Section 4.2; (iv) 2 multi-level methods: OpenOrd [60] and KDraw [61]. For PPRviz, we set the maximum number of nodes in a cluster to 25 (i.e., $k = 25$) as suggested in [46]. For the fair comparison, we follow [6–8] to modify OpenOrd and KDraw such that only the partial view of the supergraph is visualized. Note that PPRviz does not allow cluster size constraint and KDraw employs a complicated method to determine cluster size. As such, we can only set the maximum number of supernodes in the coarsest supergraph (instead of all levels) to $K$. However, we observe that PPRviz shows more supernodes than OpenOrd and KDraw in each level. For PDist computation in PPRviz, we configure the relative error $\theta = 0.5$ and other parameters as discussed in Section 6. For the competitors, we follow the parameter settings in their original papers. Our competitors are representative. For example, FR, ForceAtlas and OpenOrd are integrated into Gephi, a well-known visualization software [11].

Datasets and performance metrics. We use 12 real-world graphs in the experiments and their statistics are summarized in Table 4. 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 [38] and normalize each layout to the same scale. 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 the average visualization time for the children of each supernode over 100 random zoom-in paths. 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 2 and 3, 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 visualizations for the larger

---

Table 2: ND of PPRviz and the baselines, the best in bold and the second best in italic, \(\infty\) indicates infinity.

| Dataset  | 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| oo   | 3.1E+08| oo   | oo     | oo  | oo       | 4.6E+02 |
| FbEgo    | 2.4E+03| 1.1E+03    | 9.5E+03| 1.3E+04    | 2.0E+04| oo   | 3.6E+12| oo   | oo     | oo  | 3.9E+07  | 1.2E+05 |
| Wiki-ii  | 2.7E+04| 2.7E+04    | 1.4E+04| 4.8E+04    | 4.9E+04| oo   | 9.2E+11| oo   | 7.5E+29| 2.5E+06| 2.7E+04 |
| Physician| 6.7E+04| 7.3E+04    | 6.7E+04| 8.2E+04    | 6.5E+04| oo   | 2.5E+10| oo   | 4.0E+09| 9.4E+07| 1.1E+05 |
| FilmTrust| 9.1E+05| 7.1E+06    | 3.2E+08| 1.4E+07    | oo   | oo   | 1.2E+17| oo   | 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| oo   | 1.1E+17| oo   | oo     | oo  | 6.6E+07  | 2.2E+06 |

Table 3: ULCV of PPRviz and the baselines, the best in bold and the second best in italic, “-” indicates undefined.

| Dataset  | 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   | 0.94 | 1.04   | 1.27 | 0.86     | 0.53    |
| Physician| 0.45   | 0.53       | 0.90   | 0.55       | 0.80 | 0.47 | 0.95   | 1.67 | 1.02   | 0.77 | 1.41     | 0.53    |
| FilmTrust| 0.48   | 0.54       | 1.99   | 0.96       | 1.05 | 0.69 | 0.64   | 1.31 | 1.70   | 0.87 | 0.89     | 1.78    |
| SciNet   | 0.34   | 0.77       | 4.70   | 1.52       | 1.74 | 0.74 | 0.86   | 1.72 | 1.26   | 1.32 | 1.98     |         |

Table 4: Dataset statistics \((K = 10^{3}, M = 10^{6}, B = 10^{9})\)

| Dataset | $n$ | $m$ | Description                        |
|---------|-----|-----|------------------------------------|
| TwEgo   | 23  | 52  | Ego network [56]                  |
| FbEgo   | 52  |     | Ego network [56]                  |
| Wiki-ii | 186 | 632 | Authorship network [54]           |
| Physician| 241 | 1.8K| Social network [54]               |
| FilmTrust| 874 | 2.6K| User trust network [54]           |
| SciNet  | 1.5K| 5.4K| Collaboration network [54]         |
| Amazon  | 334.9K| 1.9M| Product network [56]              |
| Youtube | 1.1M| 6.0M| Social network [56]               |
| Orkut   | 3.1M| 234.4M| Social network [56]              |
| DBLP    | 5.4M| 17.2M| Collaboration network [54]         |
| It-2004 | 41.3M| 2.3B| Crawled network [18]              |
| Twitter | 41.7M| 3.0B| Social network [55]               |
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 places all nodes in the same position, ULCV is undefined (i.e., zero divided by zero) for SciNet and marked with “-“.

Table 2 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 3 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. In contrast, the competitors have significantly larger ULCV than PPRviz, as analyzed in Section 4.2.

Besides ND and ULCV, we also compare PPRviz with the competitors using another popular aesthetic metric [15, 53, 80]. Due to page limit, the results are reported in Appendix A, which shows that PPRviz performs the best or second best in most cases.

### 7.3 User Study

In this part, we evaluate the visualization quality of PPRviz qualitatively by a user study. We aim to answer two questions (i) how does the visualization quality of PPRviz compare with the competitors and (ii) how does the approximate level-ℓ DPPR computation in PPRviz affect visualization quality for observers. We recruited 30 participants with 6 females and 24 males, among which 28 individuals are aged 20 to 30 and 2 individuals are aged 30 to 40.

#### Settings.

To answer the first question, we generate 5 visualizations using PPRviz, FR, CMDS, Node2vec and SimRank (called a group) on the 6 small graphs. According to ND and ULCV results in Section 7.2, FR, CMDS and Node2vec are the best-performing algorithms in force-directed methods, stress methods and graph embedding methods, respectively. We randomly shuffle the 5 visualizations in each group when presenting to participants and the participants conduct the following two tasks.

- **Task 1 (T1):** rank the 5 visualizations in each group from the highest to the lowest readability by 1-5, where high readability means that the nodes and edges are shown clearly.

- **Task 2 (T2):** rank the 5 visualizations in each group from the best to the worst structure representation by 1-5, where a good structure representation makes it easy to observe clusters and strongly connected nodes.

To answer the second question, we compare the visualizations generated by PPRviz and a variant called PPRviz-PI, which uses Power Iteration (PI) [67] to compute DPPR. We set the absolute error of PI to $10^{-9}$ (smaller than the precision of float) for PPRviz-PI such that the visualizations produced by PPRviz-PI can be considered as exact. We use the FilmTrust and SciNet graphs (as PI does not scale to large graphs) and set 3 different values for $k$ (i.e., 15, 20 and 25), which produces 6 level-2 supergraphs. For each supergraph, we generate 2 visualizations using PPRviz and PPRviz-PI (called a group). We also randomly shuffle the 2 visualizations in each group when presenting to a participant.

- **Task 3 (T3):** for 2 visualization in a group, select the one with better visualization quality or the difference is hard to tell.

#### Result.

The results of the user studies are reported in Table 5. Note that for each task, we collected 180 instances (30 participants×6 visualization groups). Table 5(a) shows that PPRviz has smaller average ranking than all competitors in both readability and structural preservation, which is in line with the ND and ULCV results in Table 2 and Table 3. Moreover, PPRviz ranks the first much more often than the competitors. Table 5(b) shows that the participants cannot tell the quality difference for the visualizations generated by PPRviz and PPRviz-PI in most cases. Besides, the number of times that PPRviz and PPRviz-PI are considered to provide better visualizations are similar. These results suggest that the approximate level-ℓ DPPR computation does not affect visualization quality.

### 7.4 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 0.63 seconds. We will show in Section 7.5 that the high efficiency of 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 $S$. However, computing the forces inside a supernode $S$ 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. Recall in Section 7.1 that PPRviz visualizes more nodes than OpenOrd and KDraw in 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 visualizations for all graphs, as it is well known that computing SimRank is very expensive [48].

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 DPR 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 clustering algorithm [62] is more expensive than Louvain+ in PPRviz.

Vary cluster size. Table 6 reports the preprocessing time and response time of PPRviz by varying the maximum number of nodes (i.e., the cluster size limit k) on the largest test graph Twitter. We exclude KDraw and OpenOrd from this experiment because configuring k is difficult for them as discussed in Section 7.1. PPRviz’s preprocessing time drops when k increases because Louvain+ organizes more nodes into a supernode with larger k, resulting in fewer supernodes in each level of the supergraph and fewer levels of the supergraph hierarchy. For interactive visualization, PPRviz’s response time increases with k and the main reasons is that more pairwise PDist are computed with more nodes in each visualization. Furthermore, the response time of PPRviz is only 2.10 seconds when k = 100, above which the intra-structure of a supernode becomes dense and overburdens human perception [46].

7.5 PPRviz Variants

Besides our proposed PPRviz, other variants can be devised by replacing our proposed Tau-Push with FORA [90] 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 7. 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 7. 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

Table 6: Time of PPRviz on Twitter by varying k (in seconds).

| k   | 5  | 10 | 25 | 50  | 100 |
|-----|----|----|----|-----|-----|
| Preprocessing | 2267.65 | 2114.39 | 1934.48 | 1842.45 | 1796.87 |
| Response   | 0.28 | 0.43 | 0.63 | 1.56 | 2.10 |

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.
Table 7: Performance of PPRviz variants.

| Dataset  | Response time (seconds) | Preprocessing time (seconds) | Index size (MiB) |
|----------|------------------------|-----------------------------|------------------|
|          | Tau-Push | FORA-TP | FORA | Tau-Push | FORA-TP | FORA | Tau-Push | FORA-TP | FORA |
| Youtube  | 0.06     | 0.07    | >100 | 4.04     | 5.1     | 3.35 | 9        | 51      | 30   |
| Orkut    | 0.12     | 0.36    | >100 | 94.56    | 104.94  | 79.89 | 25       | 237     | 102  |
| It-2004  | 0.32     | 0.73    | >100 | 312.33   | 308.3   | 223.99 | 330      | 1520    | 1041 |
| Twitter  | 0.63     | 2.76    | >100 | 1984.73  | 1485.64 | 1364.04 | 338      | 1610    | 1075 |

Figure 8: Tau-Push and FPSN Comparison.

number of edges [67]. 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 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, $n$ DPPR 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 DPPR values. We select 5 level-1 clusters in each graph that contain leaf nodes with the largest DPPR, 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 DPPR value is different on each graph. For the first cluster in Youtube, the largest DPPR 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 [35, 42, 45, 85] and can be classified into two main categories: (i) force-directed methods [22, 26, 29, 47, 60, 65] and (ii) stress methods [19, 32–34, 51, 52, 61, 77, 82]. To improve efficiency, many optimizations are incorporated, e.g., grid-based partitioning [29, 60], quad-tree [47, 65, 83], and dimensionality reduction [19, 52, 77]. Multi-level methods [1, 7, 8, 12, 31, 44, 60, 61, 72, 75, 86] mainly focus on designing clustering algorithms, and the single-level methods are directly applied to layout each cluster. For example, OpenOrd [60] clusters nodes based on their Euclidean distances in the graph layout and uses FR for visualization; KDraw [61] applies label propagation for clustering and uses [33] for visualization; Grouse/GrouseFlocks [7, 8] 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 [22] uses PPR for clustering but still relies on the force-directed visualization.

**PPR computation.** The efficient computation of PPR has been extensively studied [5, 16, 28, 40, 43, 49, 50, 58, 59, 74, 76, 88–94]. Among these works, BEAR [76] and BelPI [50] improve the power iteration method [40] and achieve high efficiency by indexing several large matrices. However, the index space limits their applicability on large graphs. BiPPR [58] and HubPPR [89] combine random walks [28] with Backward-Push [59], and are subsequently improved by FORA [90]. We have shown that FORA is ineffective for level-ℓ DPPR. Similar to FORA [90], SpeedPPR [92] and Delta-Push [43] 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. 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. Regarding future works, we will extend PPRviz for incremental exploration, which not only allows users to change cluster size during the online phase but also shows other parts of current supergraph.
Shiqi Zhang, Renchi Yang, Xiaokui Xiao, Xiao Yan, and Bo Tang

[71] Aaron Quigley and Peter Eades. 2000. Fade: Graph drawing, clustering, and visual abstraction. In GD. 197–210.

[72] Jose Rodrigues, Hanghang Tong, Agma Traina, Christos Faloutsos, and Jure Leskovec. 2015. Gmine: a system for scalable, interactive graph visualization and mining. PVLDB 4 (2015), 1195–1198.

[73] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. Science 290, 5500 (2000), 2323–2328.

[74] Jieming Shi, Renchi Yang, Tianyuan Jin, Xiaokui Xiao, and Yin Yang. 2019. Real-time top-k personalized pagerank over large graphs on gpus. PVLDB 13, 1 (2019), 15–28.

[75] 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. 41–48.

[76] Kijung Shin, Jinhong Jung, Sael Lee, and U Kang. 2015. Bear: Block elimination approach for random walk with restart on large graphs. In SIGMOD. 1571–1585.

[77] Vin D Silva and Joshua B Tenenbaum. 2003. Global versus local methods in nonlinear dimensionality reduction. In NIPS, Vol. 15. 705–712.

[78] Robert R Sokal. 1958. A statistical method for evaluating systematic relationships. Univ. Kansas, Sci. Bull. 38 (1958), 1409–1438.

[79] Jian Tang, Meng Qu, Mingze Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In WWW. 1067–1077.

[80] Martyn Taylor and Peter Rodgers. 2005. Applying graphical design techniques to graph visualisation. In Inf. Vis. 651–656.

[81] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. In ICDM. 613–622.

[82] Warren S Torgerson. 1952. Multidimensional scaling: I. Theory and method. Psychometrika 17, 4 (1952), 401–419.

[83] Daniel Tunkelang. 1994. A practical approach to drawing undirected graphs. Technical Report.

[84] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. JMLR 9 (2008), 2579–2605.

[85] Tatiana Von Landesberger, Arjan Kuiper, Tobias Schreck, Jörn Kohlhammer, Jarko J van Wijk, J-D Fekete, and Dieter W Fekete. 2011. Visual analysis of large graphs: state-of-the-art and future research challenges. In CGF, Vol. 30. 1719–1749.

[86] Chris Waldshaw. 2000. A multilevel algorithm for force-directed graph drawing. In GD. 171–182.

[87] Daixin Wang, Pang Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In SIGKDD. 1225–1234.

[88] Hanzhi Wang, Zhouwei Wei, Junhao Gan, Sibo Wang, and Zengfeng Huang. 2020. Personalized PageRank to a Target Node, Revisited. In SIGKDD. 505–514.

[89] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: simple and effective approximate single-source personalized pagerank. In SIGKDD. 505–514.

[90] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: simple and effective approximate single-source personalized pagerank. In SIGKDD. 505–514.

[91] 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. 441–456.

[92] Hao Wu, Junhao Gan, Zhouwei Wei, and Rui Zhang. 2021. Unifying the Global and Local Approaches: An Efficient Power Iteration with Forward Push. In SIGMOD. 1996–2008.

[93] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S Bhowmick. 2020. Homogeneous network embedding for massive graphs via reweighted personalized PageRank. PVLDB 13, 5 (2020), 670–683.

[94] Minji Yoon, Junhong Jung, and U Kang. 2018. Tpa: Fast, scalable, and accurate method for approximate random walk with restart on billion scale graphs. In ICDE. 1132–1143.
A APPENDIX

A.1 Algorithmic Details

Louvain+. To construct supergraph hierarchy, a solution is directly using multilevel community detection algorithms such as Louvain [17], which merges well-connected nodes based on modularity optimization [63]. 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-$(t + 1)$ supergraph $G_{t+1}$, the detailed clustering strategy is that either (i) directly merge supernode $S$ to its neighboring supernode $T$ if $T$ is the only neighbor; or (ii) merge $S$ to its neighboring supernode $T$ with the largest modularity gain $\bar{Q}(S, T)$ if the size of $T$ after this merge is less than $k$. The modularity gain $\bar{Q}(S, T)$ after merging level-$(t + 1)$ supernodes $S$ and $T$ is defined as

$$\bar{Q}(S, T) = \sum_{V_i \in S} q(V_i, T).$$

Note that $q(V_i, T)$ is the modularity change of $G_t$ after moving a node $V_i$ into supernode $T$, which is defined by

$$q(V_i, T) = \frac{w(T) + w_{cr}(V_i, T)}{2m} - \left(\frac{w_{in}(T) + w_{in}(V_i)}{2m}\right)^2 - \left(\frac{w_{in}(T)}{2m}\right)^2 - \left(\frac{w_{in}(V_i)}{2m}\right)^2,$$

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

**Stress Majorization.** Stress majorization [34] 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:

$$\text{Loss}(X|\Delta) = \sum_{i < j} \left(1 + \sum_{i \in j} \frac{|X[i] - X[j]|}{\Delta[i,j]} - 2 \sum_{i \in j} \frac{|X[i] - X[j]|}{\Delta[i,j]} \right).$$

In Eq. (7), the first term is a constant and the second term can be represented by trace$(X^TL^W X)$, where $L^W$ is the weighted Laplacian matrix and defined as

$$L^W[i,j] = \begin{cases} 
\frac{1}{\Delta[i,j]} & \text{if } i \neq j \\
\frac{1}{\Delta[k]} & \text{if } i = j 
\end{cases}.$$

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

$$-2 \sum_{i < j} \frac{|X[i] - X[j]|}{\Delta[i,j]} \cdot \frac{|Y[i] - Y[j]|}{\Delta[i,j]} \leq -2 \sum_{i,j} \frac{|X[i] - X[j]|}{\Delta[i,j]} \cdot \frac{|Y[i] - Y[j]|}{\Delta[i,j]} \cdot |X[i] - Y[i]|.$$
Recall that $\Delta[i, j] \geq 2$ and $\pi_d(V_i, V_j) \leq 1/e$ accordingly, then $c < 1$ and $\frac{1}{c} - 1 > 1 - c$. By setting
$$e = 1 - c = 1 - \frac{\pi_d(V_i, V_j) + \pi_d(V_j, V_i)}{e},$$
the relative error part of $(\theta, \sigma)$-approximate PDist holds. When DPPR is not larger than $\delta$,
$$|\Delta[i, j] - \tilde{\Delta}[i, j]| \leq \theta \cdot (1 - 10 \log 2) = \theta \cdot \sigma.$$

**Proof of Lemma 6.1** For each $v_i \in F(V_i)$, denote $\tilde{\pi}_d(V_i, v_0, v_k)$ and $r(V_i, v_0, v_k)$ as the lower bound and residue value, which is firstly initialized from $v_0$ and then distributed to $v_k$. Initially, $r(V_i, v_0, v_k)$ is set to $d(\text{sink})$ (Line 3). According to the proof in [5], the following equation holds for each $v_i \in V$ in graph traversal,
$$\pi_d(v_i, v_j) = \tilde{\pi}_d(V_i, v_0, v_j) + \sum_{v_k \in V} r(V_i, v_j, v_k) \cdot \pi(v_k, v_i).$$

Note that for each $v_i \in V$, $\sum_{v_k \in F(V_i)} \tilde{\pi}_d(V_i, v_k, v_j) = \tilde{\pi}_d(V_i, v_j)$ and residue value holds the same relationships. By summing up all $v_0 \in F(V_i)$, the above equation turns to
$$\pi_d(V_i, v_j) = \tilde{\pi}_d(V_i, v_j) + \sum_{v_k \in V} r(V_i, v_i, v_k) \cdot \pi(v_k, v_i).$$

By Eq. (3), the DPPR between supernodes $V_i$ and $V_j$ is $\pi_d(V_i, V_j) = \sum_{v_i \in F(V_i)} \pi_d(v_0, v_i)$. Then the above equation can be transformed to
$$\pi_d(V_i, V_j) = \tilde{\pi}_d(V_i, V_j) + \sum_{v_i \in F(V_i)} \sum_{v_k \in V} r(V_i, v_i, v_k) \cdot \pi(v_k, v_i).$$

**Proof of Lemma 6.2** Based on Lemma 6.1, for any supernode $V_i \in S$, we can express the approximation error as
$$\pi_d(V_i, V_j) - \tilde{\pi}_d(V_i, V_j) = \sum_{v_i \in F(V_i)} \sum_{v_k \in V} r(V_i, v_i, v_k) \cdot \pi(v_k, v_i).$$

As Algorithm 1 terminates when the residue value $r(V_i, v_0, v_k) \leq r_{\text{max}} \cdot d(v_k)$ for all $v_k$, the error term on the RHS can be bounded by
$$\sum_{v_i \in F(V_i)} \sum_{v_k \in V} r(V_i, v_i, v_k) \cdot \pi(v_k, v_i) \leq d_{\text{sum}} \cdot \sum_{v_i \in F(V_i)} \pi(v_0, v_i).$$

Thus, $\tilde{\pi}_d(V_i, V_j)$ is $(\epsilon, \delta)$-approximate if we set
$$r_{\text{max}} = \frac{\epsilon \cdot \delta}{|F(V_i)|}.$$

Based on Eq. (5), for a target $V_j \in S$, the correctness of FPSN is guaranteed by setting
$$r_{\text{max}} = \frac{\epsilon \cdot \delta}{|F(V_j)|},$$
where $r_{\text{max}} = \frac{\epsilon \cdot \delta}{t_{\text{max}}}$. FPSN is $(\epsilon, \delta)$-approximate for target supernode $V_j \in S$ with $t_{\text{max}} = \tau$. The proof completes.

**Proof of Lemma 6.3** Similar to the invariant property of conventional Backward-Push [59], BPSN has the following property.

Proposition 1. BPSN satisfies the following invariant:
$$\pi_d(v_i, V_j) = \tilde{\pi_d}(v_i, V_j) + d(v_i) \cdot \sum_{v_k \in V} \pi(v_i, v_k) \cdot r(v_k, V_j).$$  (10)
Time for the worst case. The threshold $\tau$ determines how quickly FPSN can be terminated and how many BPSN are performed. A small $\tau$ engenders a low cost for FPSN and a high cost for BPSN, and vice versa. Therefore, the appropriate setting of $\tau$ should balance the workloads of FPSN and BPSN in Tau-Push. First, note that the complexity of FPSN for a given leaf source $v_i \in S$ is $O \left(\frac{d(v_i)}{\epsilon \delta} \right)$ [5], 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 [23], the largest out-degree is $O \left(\frac{n \sqrt{\ln n}}{\epsilon \delta} \right)$, 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 $O \left(\frac{knm}{\epsilon \delta} \right)$. Next, for a leaf target $v_j$, the worst time complexity of BPSN is $O \left(\frac{\sum_{v_i \in V} d(v_i) \cdot m_i}{\epsilon \delta \cdot r_{max}} \right)$ [88] and can be simplified as $O \left(\frac{m \cdot r_{max}}{\epsilon \delta \cdot r_{max}} \right)$ using Eq. (5). As $r_{max} = O(1)$ and $r'_{max} = \frac{\epsilon \delta}{\max} \cdot \delta$, the time complexity of BPSN can be re-written as $O \left(\frac{d_{max} \cdot m}{\epsilon \delta \cdot \delta} \right)$. Since there are at most $O(1/\tau)$ target nodes with average DPR larger than $\tau$, the time complexity of Lines 3 - 4 of Algorithm 1 is $O \left(\frac{1}{\tau} \cdot \frac{d_{max} \cdot m}{\epsilon \delta} \right)$.

T setting. Based on the aforementioned analysis, the worst-case time complexity of Tau-Push is $O \left(\frac{knm}{\epsilon \delta} + \frac{d_{max} \cdot m}{\epsilon \delta \cdot \delta} \right)$ [88]. Since the degree and DPR follows the same power law [57], the probability that a random-select node has out-degree larger than $d_{max}$ is $1/\tau$. Hence, following the proof by [23], $d_{max} = O(\log n)$ for the scale-free networks. Therefore, the time complexity turns into $O \left(\frac{knm}{\epsilon \delta} \right)$. For the single-source approximation in FPSN, we set $\delta = \frac{1}{O(\tau)}$ [74] as nodes in the same supernode have good connectivity (i.e., large DPRV) and we focus on the top-$O(k)$ DPRV 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 $\delta = O(n/k)$ for BPSN as empirically $\sum_{v_i \in V} d(v_i)/(r_{max}/2)$ is comparable to the random level-1 supernode $S$ and $\rho$ should be comparable to $\tau$. With these configurations, by setting $\tau = 1/\sqrt{kn}$, the worst-case time complexity of Tau-Push in Algorithm 1 is minimized to $O \left(\frac{kn \cdot \sqrt{kn}}{\epsilon \delta} \right)$.

For a random supernode $S$. In scale-free networks, both DPR and DPRV values follow the power law [10, 57, 58, 74, 91], hence, the i-th largest DPRV value is $O \left(\frac{1}{\log n} \right)$ [74]. As there are $n/k$ supernodes at level-$i$ and the worst-case complexity is $O \left(\frac{k \cdot (\log n)}{n \cdot \log n} \right)$, which is $\sqrt{kn}$ times slower than Tau-Push.

FORA and FORA-TP

FORA. [90] is invoked from each leaf node $v_f \in V_f$. In particular, FORA first utilizes Forward-Push [5] to derive rough approximations of the DPRV values, and then estimates the error term in Eq. (4) by exploiting random walk samplings [28]. Based on Lemma 6.5, plugging the approximate DPRV into Eq. (3) yields $(\epsilon, \delta)$-approximate level-$\ell$ DPRV.

However, FORA has high computation complexity. Specifically, the Forward-Push phase costs $O\left(\frac{d(v_f)}{r_{max}}\right)$ [5], and the random walk phase costs $O\left(\frac{n \cdot m}{\log n} \cdot W\right)$ as $\sum_{v_i \in V} d(v_i)/(m \cdot W)$ to balance the time complexities of two phases. Thus, FORA costs $O\left(\sqrt{d(v_f)} \cdot m \cdot W\right)$ for a source leaf node $v_f \in V_f$. By summarizing the time complexities for each $v_f \in V_f$ and $v_i \in S$, the time complexity for computing all pairwise approximate level-$\ell$ DPRV in a supernode $S$ is $O \left(\sum_{v_f \in V} \sum_{v_i \in E(V_f)} \sqrt{d(v_f)} \cdot m \cdot W\right)$, which is prohibitively high, as a high-level supernode $V_f$ can contain many leaf nodes.

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 $\hat{\tau}_d(V_i, V_j)$ for each $V_i \in S$ and the residue value $\hat{\tau}_d(V_i, v_k)$ for each $v_k \in V_f$ (Line 1). After that, $\alpha$ 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 $\hat{\tau}_d(V_i, v_k)$.
Table 8: AR of PPRviz and the baselines, the best in bold and the second best in italic, “-” indicates undefined.

|          | PPRviz | OperOrd/FR | LinLog | ForceAtlas | CMDS  | PMDS  | GFactor | SDNE  | LapEig | LLE   | Node2vec | SimRank |
|----------|--------|------------|--------|------------|-------|-------|---------|-------|--------|-------|----------|---------|
| TwEgo    | 0.00E+00 | 1.95E+00  | 1.21E+00 | 9.26E-03  | 0.00E+00 | 5.00E+00 | 1.40E+00 | 6.00E+00 | 2.51E+01 | 3.87E+00 | 3.45E+00 |
| FbEgo    | 4.09E+01 | 5.60E+01  | 6.83E+01 | 4.54E+01  | 4.01E+02 | 4.70E+02 | 5.14E-02 | 6.44E+00 | 2.71E+00 | 9.81E+01 | 2.97E+00 |
| Wiki-ii  | 4.80E+02 | 8.87E+02  | 4.34E+02 | 5.20E+02  | 4.89E+02 | 8.15E+02 | 6.41E+02 | 8.09E+02 | 2.70E+03 | 7.41E+02 | 6.43E+01 |
| Physician| 6.72E+02 | 4.08E+02  | 4.70E+02 | 5.20E+02  | 4.34E+02 | 5.14E+02 | 6.44E+00 | 2.71E+00 | 9.81E+01 | 2.97E+00 | 6.43E+01 |
| FilmTrust| 7.51E+02 | 8.03E-02  | 7.89E+02 | 8.09E+02  | 8.15E+02 | 6.41E+02 | 6.44E+00 | 2.71E+00 | 9.81E+01 | 2.97E+00 | 6.43E+01 |
| SciNet   | 7.51E+02 | 7.50E+02  | 1.57E+03 | 1.29E+03  | 1.20E+03 | 7.52E+03 | 1.14E+04 | 1.00E+04 | 1.10E+03 | 6.43E+01 | 6.43E+01 |

More visualization results. In Figures 9 to 13, we report the visualization results of PPRviz and other competitors on FbEgo, Wiki-ii, Physician, FilmTrust and SciNet graphs. Specifically, we enlarge the results of PPRviz and a promising competitor ForceAtlas for users comparison.
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 ★.
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 ★.
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 ★.
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 ★.
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 ★.