Effective and Efficient PageRank-based Positioning for Graph Visualization

SHIQI ZHANG, National University of Singapore, Singapore and Southern University of Science and Technology, China
RENCHI YANG*, Hong Kong Baptist University, China
XIAOKUI XIAO†, National University of Singapore, Singapore
XIAO YAN and BO TANG†, Southern University of Science and Technology, China

Graph visualization is a vital component in many real-world applications (e.g., social network analysis, web mining, and bioinformatics) that enables users to unearth crucial insights from complex data. Lying in the core of graph visualization is the node distance measure, which determines how the nodes are placed on the screen. A favorable node distance measure should be informative in reflecting the full structural information between nodes and effective in optimizing visual aesthetics. However, existing node distance measures yield sub-par visualization quality as they fall short of these requirements. Moreover, most existing measures are computationally inefficient, incurring a long response time when visualizing large graphs. To overcome such deficiencies, we propose a new node distance measure, PDist, geared towards graph visualization by exploiting a well-known node proximity measure, *personalized PageRank*. Moreover, we propose an efficient algorithm Tau-Push for estimating PDist under both single- and multi-level visualization settings. With several carefully-designed techniques, Tau-Push offers non-trivial theoretical guarantees for estimation accuracy and computation complexity. Extensive experiments show that our proposal significantly outperforms 13 state-of-the-art graph visualization solutions on 12 real-world graphs in terms of both efficiency and effectiveness (including aesthetic criteria and user feedback). In particular, our proposal can interactively produce satisfactory visualizations within one second for billion-edge graphs.

CCS Concepts: • **Human-centered computing → Graph drawings**; • **Theory of computation → Graph algorithms analysis**; **Random walks and Markov chains**.

Additional Key Words and Phrases: graph visualization, personalized pagerank, approximate algorithm

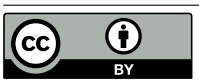
ACM Reference Format:
Shiqi Zhang, Renchi Yang, Xiaokui Xiao, Xiao Yan, and Bo Tang. 2023. Effective and Efficient PageRank-based Positioning for Graph Visualization. *Proc. ACM Manag. Data* 1, 1, Article 76 (May 2023), 27 pages. https://doi.org/10.1145/3588930

1 INTRODUCTION

Graph visualization is an effective way to help users comprehend and analyze complex relational data (e.g., social and biological networks). It has been identified as one of the most popular and

*Work done at the National University of Singapore.
†Corresponding authors (xkxiao@nus.edu.sg, tangb3@sustech.edu.cn)

Authors' addresses: Shiqi Zhang, s-zhang@comp.nus.edu.sg, National University of Singapore, Singapore, Singapore and Southern University of Science and Technology, Shenzhen, China; Renchi Yang, renchi@hkbu.edu.hk, Hong Kong Baptist University, Hong Kong, China; Xiaokui Xiao, xkxiao@nus.edu.sg, National University of Singapore, Singapore, Singapore; Xiao Yan, yanx@sustech.edu.cn; Bo Tang, tangb3@sustech.edu.cn, Southern University of Science and Technology, Shenzhen, China.



2836-6573/2023/5-ART76
https://doi.org/10.1145/3588930

challenging graph processing tasks in graph database systems and graph libraries [61]. In practice, graph visualization finds extensive applications, such as analyzing protein interactions in the organism [2], studying scholars' co-authoring behaviors [59], and capturing the massive hyperlinks among web pages [12]. This motivates a plethora of graph visualization solutions [1, 5, 15, 21, 23, 25, 26, 33, 36, 39, 49, 50, 53, 59, 70] and softwares [6, 7, 18, 62].

A central problem in graph visualization is calculating an effective layout, i.e., the coordinate of each node on the screen, which seeks to place closely-related nodes close and unrelated nodes far apart based on a node distance measure calculated from the graph. In most existing solutions, classic node distance measures, including the shortest distance [15, 25, 26, 39, 50, 64, 70] and the direct linkage [1, 5, 21, 23, 36, 49, 53], are widely used. However, such distance measures overlook the high-order structure information of the graph and fail to optimize some critical visual features, leading to sub-par visualizations (e.g., node overlapping and edge distortion). For example, the classic stress method [26] determines node positions by employing unrecognizable shortest distances, resulting in severe node overlapping.

Another long-standing challenge is visualizing large graphs. When handling large graphs, most existing solutions [1, 5, 33, 49, 50, 59, 64] adopt the *multi-level* scheme to avoid the poor readability and prohibitive overhead caused by visualizing all nodes in a *single-level* fashion. Specifically, the multi-level strategy organizes the nodes of the input graph G into a tree $\mathcal{H}$, such that (i) each leaf in $\mathcal{H}$ is a node in G, and (ii) each non-leaf node in $\mathcal{H}$, referred to as a *supernode*, has only a small number of children. The user can navigate through $\mathcal{H}$ and visualize any set $\mathcal{S}$ of nodes or supernodes that have the same parent. Utilizing the multi-level scheme, existing solutions still struggle to cope with large graphs as they entail significant overheads to compute the node distances. The reason is that these methods (e.g., [49, 50]) require calculating all pairwise distances (e.g., shortest distance) for the leaf nodes in the subtrees of two supernodes $\mathcal{V}_i$ and $\mathcal{V}_j$ before determining the distance between $\mathcal{V}_i$ and $\mathcal{V}_j$; otherwise, the visualization of supernodes will be uninformative in reflecting the underlying graph structure, which results in poor visualization quality.

To address the aforementioned challenges in effectiveness and efficiency, we propose a new graph-theoretic node distance measure dedicated to enhancing visualization quality, referred to as PDist. PDist takes inspiration from *personalized PageRank (PPR)* [55], a proximity measure quantifying the connectivity from a source node to a target node via random walks in a graph. Considering the requirements of graph visualization, PDist incorporates degree and symmetry information into PPR, and ameliorates PPR with transformation and truncation. Through such optimizations, PDist circumvents the visual issues (e.g., node overlapping and edge distortion) that may be caused by PPR, while accurately preserving the graph structure (e.g., degree and high-order proximity information). Notably, our analysis shows that PDist offers non-trivial visualization quality guarantees in terms of two widely-used aesthetic criteria.

Unfortunately, computing PDist for supernodes is challenging as it involves a multitude of leaf nodes underlying the supernodes. There are a plethora of approaches for PPR computation in the literature [45, 55, 76, 77], but they focus on single-source or top-k PPR queries rather than arduous all-pair queries in the case of PDist. As a result, these approaches are inefficient when adopted for PDist computation due to redundant graph traversal operations and random walk simulations. To this end, we propose Tau-Push, an efficient solution for computing approximate PDist. Compared to the PPR computation methods, Tau-Push achieves superior time complexity and empirical efficiency, while retaining strong accuracy guarantees. Under the hood, Tau-Push adopts a filter-refinement paradigm accommodating three carefully-designed techniques. First, Tau-Push computes a rough estimation of each PDist by *grouped forward graph traversal*. Next, Tau-Push identifies a set of failed target nodes $\mathcal{T}$ by leveraging the global PageRank of the graph.

Table 1. Frequently used notations.

Notation	Description
G=(V, E)	A graph G with node set V and edge set E.
n, m	The numbers of nodes and edges in G, respectively.
$d(v_i)$	The out-degree of node v_i in G.
$\mathbf{X}$	The position matrix of nodes in G.
$\mathcal{S}, k$	The level-(ℓ+1) supernode, the number of level-ℓ children in $\mathcal{S}$.
$\mathcal{V}_i, F(\mathcal{V}_i)$	The level-ℓ supernode, the set of leaf nodes in $\mathcal{V}_i$.
π, α	The PPR value, the restart probability in the random walk.
$\Delta[i, j]$	The PDist value of node pair (v_i, v_j) defined in Eq. (1).
$\pi_d(v_i, v_j)$	The DPPR value of node v_j w.r.t. node v_i defined in Eq. (1).
$\pi_d(\mathcal{V}_i, \mathcal{V}_j)$	The DPPR value of supernode pair $(\mathcal{V}_i, \mathcal{V}_j)$ defined in Eq. (2).
$\hat{\pi}_d(v_i, v_j)$	The approximate DPPR of node v_j w.r.t. node v_i (see Eq. (3)).
$r(v_i, v_j)$	The residue of node v_j w.r.t. node v_i (see Eq. (3)).
r_{max}, r^b_{max}	The forward and backward residue thresholds, respectively.
ϵ, δ	The approximation parameters in Definition 3.5.
τ_i	The DPR of level-ℓ supernode $\mathcal{V}_i$ in Eq.(4).

Finally, Tau-Push refines the PDist estimation of such nodes by performing a handful of graph traversal operations backward from $\mathcal{T}$. Note that our PDist and its computation algorithm Tau-Push are not limited to single- and multi-level graph visualizations, and could underpin other scenarios, including graph query result visualization [11] and incremental graph exploration [31].

Based on PDist and Tau-Push, we create our graph visualization solution PPRviz and experimentally evaluate PPRviz against 13 state-of-the-art methods using 12 real-world graphs. Our empirical results demonstrate that PPRviz outperforms the competitors in both effectiveness and efficiency. First and foremost, PPRviz obtains considerably better visualization quality in terms of aesthetic metrics and user satisfaction than the competitors. Moreover, PPRviz runs at a fraction of the computational cost of the competing methods for both pre-processing and online visualization. For instance, PPRviz takes only 1 second to visualize the (super) node sets from a graph with 3 billion edges, whereas none of the competitors can produce a visualization within 1000 seconds.

To summarize, this paper makes the following contributions:

- We propose PDist, a new node distance measure that not only fully captures the structural information of the input graph but also optimizes the aesthetic metrics.
- We devise Tau-Push, an algorithm for efficient PDist computation with three tailored techniques, which enables responsive visualizations even on billion-edge graphs.
- We conduct extensive experiments to demonstrate the superiority of PDist and Tau-Push over the state-of-the-art graph visualization methods in both effectiveness and efficiency.

2 PRELIMINARIES

In this section, we first elaborate on the procedure of graph visualization and then introduce the multi-level mechanism for visualizing large graphs. Finally, we discuss how to assess the visualization quality. Table 1 lists the frequently used notations in our paper.

2.1 Graph Visualization

Let $G = (V, E)$ be a graph, where V is a set of n nodes and E is a set of m edges. We assume that G is a directed and homogeneous graph, where the nodes and edges have no labels/attributes. We consider the task of visualizing the input graph G on the two-dimensional Euclidean space, where the graph G is laid out based on a position matrix $\mathbf{X} \in \mathbb{R}^{n\times 2}$ and $\mathbf{X}[i] \in \mathbb{R}^2$ (i.e., the i-th

row of $\mathbf{X}$) records the coordinate of node $v_i \in V$ on the screen. Following prior visualization methods [15, 21, 23, 25, 26, 36, 39, 49], we represent each node with a circle by placing the center of the circle on the coordinate of the node, and draw each directed edge using a straight arrowhead line. For undirected edges, we omit the arrowhead to avoid a cluttered display. Thus, $\|\mathbf{X}[i] - \mathbf{X}[j]\|$ is the distance between node v_i and v_j on the screen and $l(v_i, v_j) = \|\mathbf{X}[i] - \mathbf{X}[j]\|$ is the length of edge $(v_i, v_j) \in E$.

Given an input graph G, graph visualization typically consists of two phases: (i) *distance matrix computation* and (ii) *position matrix embedding*. In the first phase, a specific node distance matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ is computed, in which $\mathbf{D}[i, j]$ reflects the graph-theoretic distance between nodes v_i and v_j. For example, [23, 36, 53] directly employ the adjacent matrix as $\mathbf{D}$ and [25, 26, 39] use the all-pair shortest distances as $\mathbf{D}$. In the second phase, the distance matrix $\mathbf{D}$ is converted to a position matrix $\mathbf{X}$, such that the on-screen distance $\|\mathbf{X}[i] - \mathbf{X}[j]\|$ of node pair (v_i, v_j) is close to $\mathbf{D}[i, j]$ for all node pairs $(v_i, v_j) \in V \times V$. In literature, there exist several optimization techniques that transform the distance matrix into the position matrix, e.g., gradient descent [25], simulated annealing [17], and stress majorization [26].

2.2 Multi-level Mechanism

Directly visualizing all nodes in a large graph usually results in a giant hairball with little discernible structure information, due to the sheer numbers of nodes and edges in the layout [27]. Thus, most existing methods [1, 5, 59, 64] and softwares [6, 18, 62] visualize large graphs in an interactive and *multi-level* manner, so as to cut down the number of nodes in each drawing. As surveyed in [31, 72], multi-level methods consist of two phases: (i) *preprocessing* and (ii) *interactive visualization*. In the preprocessing phase, a *supergraph* hierarchy is constructed such that nodes of the graph G are organized into a tree $\mathcal{H}$, where (i) each leaf is a node in G, and (ii) each non-leaf node, referred to as a *supernode*, contains k children. For convenience, we say that each leaf node is at level-0, and that each supernode $\mathcal{V}_i$ is at level-$(\ell + 1)$ if its children are at level-ℓ ($\ell \geq 0$). In the interactive visualization phase, users can select any supernode $\mathcal{S}$ at level-$(\ell + 1)$, and ask for a visualization of the k children of $\mathcal{S}$, where the corresponding position matrix $\mathbf{X} \in \mathbb{R}^{k \times 2}$ is derived following the visualization procedure described in the preceding section. For example, if $\mathcal{S}$ consists of leaf children, then multi-level methods visualize the subgraph of G induced by the nodes in $\mathcal{S}$. On the other hand, if $\mathcal{S}$ consists of supernode children, then they visualize a high-level graph where (i) each node represents a supernode $\mathcal{V}_i$ in $\mathcal{S}$ and (ii) each edge connects from supernode $\mathcal{V}_i$ to supernode $\mathcal{V}_j$ if G contains an edge from a leaf node in the sub-tree of $\mathcal{V}_i$ to another leaf node in the sub-tree of $\mathcal{V}_j$. Notice that the size constraint k is conducive to curtailing visual clutter [20] and can be configured by users according to their needs. Throughout this paper, we refer to visualizing the entire graph on one single layout (resp. multiple levels of layouts) as single-level (resp. multi-level) visualization.

Although the supergraph hierarchy can be constructed offline and the time complexity of position matrix embedding is reduced to $O(k^3)$ [26] for the children of $\mathcal{S}$, existing multi-level methods still incur high costs for the distance matrix computation of the children in $\mathcal{S}$, especially on graphs with many nodes and edges. A keen reader may propose to directly compute the distances of the children in $\mathcal{S}$ by treating $\mathcal{S}$ as a stand-alone graph. However, doing so overlooks the underlying structures of $\mathcal{S}$, and thus impairs visualization quality as pinpointed in [49, 50]. Instead, a canonical approach adopted by existing methods [49, 50, 66] 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, where v_i (resp. v_j) is a leaf node in $\mathcal{V}_i$ (resp. $\mathcal{V}_j$). In other words, this approach requires computing $k^\ell \times k^\ell$ pairs of distances for two level-ℓ supernodes, which poses a challenge in terms of computation efficiency.

2.3 Visualization Quality Assessment

Intuitively, a high-quality graph visualization should not only reflect the topological information of the input graph but also have good *readability* [8, 27]. In other words, the position matrix $\mathbf{X}$ in the Euclidean space should accurately reflect the structure of G, in the sense that well-connected nodes (resp. poorly-connected nodes) should be placed close (resp. far apart) to each other. As for good readability, it means that 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 several aesthetic criteria in the literature that quantify the readability of graph layouts. In this paper, we adopt the two most commonly-used aesthetic metrics [10, 40, 54, 57, 58, 68], i.e., *node distribution (ND)* and *uniform length coefficient variance (ULCV)*, defined as follows.

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

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 $\mathrm{ULCV}(\mathbf{X}) = l_\sigma / l_\mu$.

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 the same node positions) lead to an infinite ND score. ULCV measures the skewness of edge lengths. A large ULCV indicates that some edges are considerably longer or shorter than others, in which case the layout tends to look distorted.

3 PPR-BASED NODE DISTANCE

This section presents our node distance measure PDist for graph visualization. We first formally define PDist for the case of single-level visualization and then extend it to multi-level visualization.

3.1 Single-level PDist

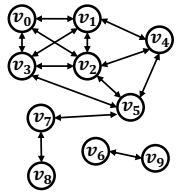
Definition. PDist is formulated based on *personalized PageRank (PPR)*, a node proximity measure defined as follows. Given a directed graph $G = (V, E)$, two nodes $v_i, v_j \in V$, and a *restart probability* α, the PPR $\pi(v_i, v_j)$ from v_i to v_j is defined as the probability that a *random walk with restart* (RWR) [69] 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, a large PPR $\pi(v_i, v_j)$ indicates that numerous paths exist from v_i to v_j; in other words, v_i is well connected to v_j. Based on PPR, we define PDist as follows.

*Definition 3.1 (*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

$$\mathbf{\Delta}[i, j] = \min\Big(\max\Big(1 - \log\big(\pi_d(v_i, v_j) + \pi_d(v_j, v_i)\big), 2\Big), 2\log n\Big), \tag{1}$$

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

The intuition behind PDist is that if node pair (v_i, v_j) has a high PPR value $\pi(v_i, v_j)$, then its PDist $\Delta[i, j]$ should be small. As such, PDist ensures that well-connected nodes are placed closely in a visualization. Moreover, PDist also accounts for the out-degrees of v_i and v_j in its definition to overcome the inherent limitation of PPR for visualization. To explain, we consider the example in Fig. 1, which shows a graph with nodes v_0–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



	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

Fig. 1. PPR and PDist in a toy graph.

node pairs (v_2, v_0) and (v_6, v_9) are both directly connected via an edge. This indicates that for adjacent node pairs in a graph, their PPR values could vary considerably. As a consequence, directly transforming PPR values into node distances would bring about a large variance in edge lengths in the visualization. To alleviate this issue, in our formulation of PDist, we scale each $\pi(v_i, v_j)$ by multiplying it with the out-degree of the source node v_i. It has been shown in previous work [82] that such a scaling contributes to an accurate measure of the strength of connections between nodes. In Fig. 1, 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. Moreover, the PDist of (v_0, v_8) is large, reflecting the fact that v_0 is far away from v_8 in the input graph.

To make the node distance symmetric, we formulate PDist based on $\pi_d(v_i, v_j) + \pi_d(v_j, v_i)$, since $\pi_d(v_i, v_j) \neq \pi_d(v_j, v_i)$ in general. Furthermore, PDist takes the inverse of the natural logarithm of DPPR $\pi_d(v_i, v_j) + \pi_d(v_j, v_i)$ and imposes an empirical truncation to narrow down the distance variance, thereby ensuring the distances between nodes lie in a reasonable range that can be well fitted into a canvas. In particular, the lower and upper bounds of PDist are set to 2 and $2 \log n$ for precluding node overlapping and blank space issues, respectively. The rationale of choosing $2 \log n$ as the upper bound is that the average PPR value between a node pair in a graph with n nodes is $1/n$ [63, 77] and PDist should focus on reflecting the proximity information of node pairs with above-average PPR values (i.e., relatively high relevance).

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

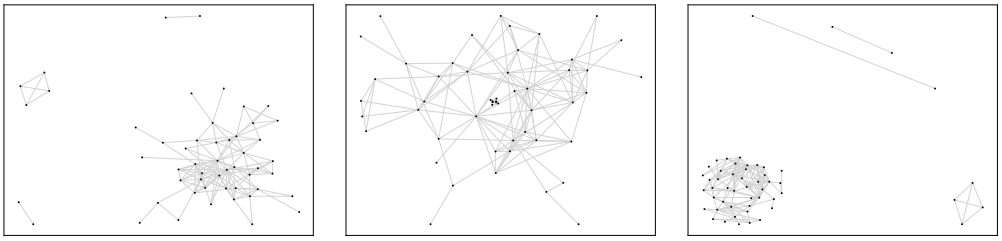
Theorem 3.2. *Given the* PDist *matrix* $\mathbf{\Delta}$ *of a graph* G*, suppose that* $||\mathbf{X}[i] - \mathbf{X}[j]|| = \mathbf{\Delta}[i, j]$ $\forall v_i, v_j \in V$*, then* $ND(\mathbf{X}) \leq 0.215e \cdot m + 0.0175n^2$*, where* e *is the Euler constant.*

Theorem 3.3. *Given the* PDist *matrix* $\mathbf{\Delta}$ *of a graph* G*, suppose that* $||\mathbf{X}[i] - \mathbf{X}[j]|| = \mathbf{\Delta}[i, j]$ $\forall v_i, v_j \in V$ *and the restart probability* $\alpha \leq \frac{1}{2} - \sqrt{\frac{1}{4} - \frac{1}{2e}}$*, then* $ULCV(\mathbf{X}) \leq \frac{\left(\log \frac{1}{2\alpha(1-\alpha)} - 1\right)}{4}$.

Note that the upper bound of the restart probability α in Theorem 3.3 (which is approximately 0.243) is not restrictive, as α is usually set to 0.15 or 0.2 [46, 74, 75, 79].

A case study. To verify the effectiveness of PDist, we use PDist and two classic node distance measures (i.e., the shortest distance and SimRank-based distance) to visualize the graph *FbEgo* via the same position matrix embedding algorithm [26]. Note that the SimRank-based distance is obtained by plugging SimRank [37] into Eq. (1). Fig. 2 displays the visualization results of the three distance measures. It can be observed that PDist yields a high-quality visualization result, which organizes the graph into a well-connected cluster and three cliques. In comparison, the widely-used shortest distance measure [15, 25, 26, 50] suffers from severe node overlapping and edge distortion issues. Regarding the visualization derived from SimRank-based distances, the edges of the cliques are distorted, as SimRank assigns node pairs within the large connected component

[1]We refer interested readers to [84] for all proofs.



ND=2.4E+03, ULCV=0.39 (a) PDist
ND=2.0E+04, ULCV=0.46 (b) Shortest distance [26]
ND=6.2E+03, ULCV=0.75 (c) SimRank [37]

Fig. 2. Visualization results of PDist and two other node distances measures on the *FbEgo* graph. Smaller aesthetic scores indicate better quality.

high similarity scores but 0 similarity scores to the 2-cliques. Besides the qualitative results, we also report ND and ULCV scores in Fig. 2, and we can find that PDist considerably outperforms the shortest distance and the SimRank-based distance in terms of both metrics, validating the superior aesthetic performance of PDist as proved in Theorem 3.2 and Theorem 3.3.

3.2 Multi-level PDist

Definition. Given a user-specified level-$(\ell+1)$ supernode $\mathcal{S}$, multi-level visualization requires calculating the level-ℓ PDist between the k level-ℓ children supernodes of $\mathcal{S}$. Recall from Definition 3.1 that a linchpin functioned in single-level PDist is the DPPR between two nodes. Thus, we extend the DPPR in Definition 3.1 to level-ℓ DPPR (Definition 3.4), and level-ℓ PDist can be derived accordingly by plugging Eq. (2) into Eq. (1).

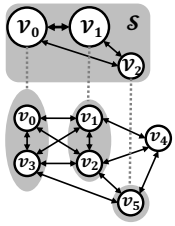
Definition 3.4 (Level-ℓ DPPR). For two level-ℓ supernodes $\mathcal{V}_i$ and $\mathcal{V}_j$, denote the set of leaf nodes in $\mathcal{V}_i$ as $F(\mathcal{V}_i)$, the level-ℓ DPPR $\pi_d(\mathcal{V}_i, \mathcal{V}_j)$ of $\mathcal{V}_j$ w.r.t. $\mathcal{V}_i$ is defined as

$$\pi_d(\mathcal{V}_i, \mathcal{V}_j) = \frac{\sum\limits_{v_s \in F(\mathcal{V}_i), v_t \in F(\mathcal{V}_j)} \pi_d(v_s, v_t)}{|F(\mathcal{V}_i)| \cdot |F(\mathcal{V}_j)|}, \tag{2}$$

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 of measuring the proximity between two supernodes by summarizing the structure of the underlying leaf nodes is also adopted in [49, 66, 71]. In particular, [49, 66] also consider the average of all pairwise distances of the leaf nodes.

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 on it (called W-DPPR). More precisely, the graph is constructed by treating each supernode $\mathcal{V}_i \in \mathcal{S}$ as a node and merging the edges between supernodes $\mathcal{V}_i$ and $\mathcal{V}_j$ in $\mathcal{S}$ as a weighted edge. Although W-DPPR can be computed very efficiently as it requires only an $O(k^2)$ cost, it ignores the micro-structures of each supernode and the paths containing nodes outside $\mathcal{S}$, resulting in sub-par visualization quality. To exemplify, we consider Fig. 3, which shows a graph with nodes v_0–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 to it than $\mathcal{V}_0$ as the children of $\mathcal{V}_2$ and $\mathcal{V}_0$ share one common neighbor v_2, whereas the children of $\mathcal{V}_2$ and $\mathcal{V}_1$ have two common neighbors v_3 and v_4. From the table in



	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

Fig. 3. A toy example of level-ℓ DPPR.

Fig. 3, we observe that the W-DPPR values of $(\mathcal{V}_2, \mathcal{V}_0)$ and $(\mathcal{V}_2, \mathcal{V}_1)$ are the same, which is counter-intuitive. In contrast, level-ℓ DPPR addresses this issue and accurately captures the structure of the original graph.

3.3 Efficiency Challenges

As per Eq.(1) and Eq.(2), the level-ℓ PDist between $\mathcal{V}_i$ and $\mathcal{V}_j$ requires computing the exact PPR values for all pairs of leaf nodes in $F(\mathcal{V}_i) \times F(\mathcal{V}_j)$, where $F(\mathcal{V}_i)$ (resp. $F(\mathcal{V}_j)$) signifies the set of leaf nodes of $\mathcal{V}_i$ (resp. $\mathcal{V}_j$). As pointed out in prior work [82], exact PPR computation is prohibitive as it entails summing up an infinite series. An option is to compute the near-exact result by performing power iterations (PI) [55] until the absolute error of PPR is less than 10^{-9} (the precision of float). That said, we still need to invoke PI for each of the $O(k^{\ell+1})$ leaf nodes in the specified level-$(\ell+1)$ supernode $\mathcal{S}$, where each invocation has $O(m)$ time complexity [55]. As a result, the near-exact computation of the level-ℓ PDist matrix incurs a high cost of $O(k^{\ell+1}m)$, which may be $O(mn)$ in the worst case (i.e., for the highest level supergraph).

To alleviate the above-said efficiency issue, we resort to computing approximate level-ℓ PDist. Towards this end, we define the concept of $(\epsilon, \delta)-$approximate level-ℓ DPPR.

Definition 3.5 ($(\epsilon, \delta)-$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 $\pi_d(\mathcal{V}_i, \mathcal{V}_j) \geq \delta$, $|\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) - \pi_d(\mathcal{V}_i, \mathcal{V}_j)| \leq \epsilon \cdot \pi_d(\mathcal{V}_i, \mathcal{V}_j)$.

The following Lemma 3.6 shows that we can convert the (ϵ, δ)-approximate level-ℓ DPPR into an approximate level-ℓ PDist. Accordingly, we focus on computing $(\epsilon, \delta)-$approximate level-ℓ DPPR in the rest of the paper.

Lemma 3.6. *Given constants θ and σ, by setting $\delta = \frac{e^{1-\sigma}}{2}$ and $\epsilon = 1 - \left(\frac{1}{e^2}\right)^{\theta}$, (ϵ, δ)-approximate level-ℓ* DPPR *results in an approximate level-ℓ* PDist $\widehat{\Delta}[i, j]$ *satisfying*

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

To compute $(\epsilon, \delta)-$approximate level-ℓ DPPR values w.r.t. a source supernode $\mathcal{V}_i$, a straightforward approach is to utilize existing single source PPR (SSPPR) approximation methods [4, 22, 32, 63, 77, 79]. Specifically, the state-of-the-art methods [45, 48, 63, 76, 77, 79] are built upon Forward-Push [4]. Given a source leaf node v_i, Forward-Push maintains an estimated DPPR $\widehat{\pi}_d(v_i, v_j)$ and a residue value $r(v_i, v_j)$ for each node v_j, where $r(v_i, v_i)$ is set to $d(v_i)$ and all $\widehat{\pi}_d(v_i, v_j)$ and other residue $r(v_i, v_j)$ are set to 0 initially. Then, Forward-Push starts a deterministic graph traversal from source node v_i, and at each step converts α portion of the residue of current node v_j (i.e.,

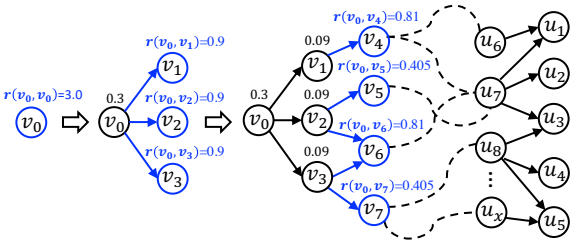


Fig. 4. An example of Forward-Push.

$r(v_i, v_j)$) into its estimated DPPR value $\widehat{\pi}_d(v_i, v_j)$ and distributes the remaining $1-\alpha$ part to v_j's *out-neighbors* evenly. For convenience, we refer to the operation of distributing a fraction of node v_j's residue to one of its out-neighbors as a *push* operation.

A running example. To exemplify, suppose that the restart probability $\alpha = 0.1$ and Forward-Push is performed starting from node v_0 in Fig. 4. Correspondingly, v_0 first converts $\alpha = 0.1$ fraction of initial residue $r(v_0, v_0) = d(v_0) = 3.0$ to $\widehat{\pi}(v_0, v_0) = 0.3$, and then evenly pushes the remaining residue (i.e., 2.7) to each of its out-neighbors, resulting in $r(v_0, v_1) = r(v_0, v_2) = r(v_0, v_3) = 0.9$. After that, nodes v_1, v_2, and v_3 continue such push operations and lead to $r(v_0, v_4) = r(v_0, v_6) = 0.81$ for nodes v_4, v_6 and $r(v_0, v_5) = r(v_0, v_7) = 0.405$ for nodes v_5, v_7.

In particular, the following invariant [4] holds during the course of push operations:

$$\pi_d(v_i, v_j) = \widehat{\pi}_d(v_i, v_j) + \sum_{v_k \in V} \frac{1}{d(v_k)} \cdot r(v_i, v_k) \cdot \pi_d(v_k, v_j), \tag{3}$$

where $\widehat{\pi}_d(v_i, v_j)$ is an estimation of $\pi_d(v_i, v_j)$ and the other term $\sum_{v_k \in V} \frac{1}{d(v_k)} \cdot r(v_i, v_k) \cdot \pi_d(v_k, v_j)$ can be regarded as the estimation error. Intuitively, an exact DPPR can be obtained when the residue of each node is eventually depleted, i.e., $r(v_i, v_k) = 0$ for all $v_k \in V$. In practice, Forward-Push computes approximate DPPR by conducting pushes until every residue value is less than a given threshold r_{max}, referred to as the *forward residue threshold*. When r_{max} is small, Forward-Push incurs significant computational overhead, as a large number of push operations are required to deplete residues. To address this issue, most of the state-of-the-art methods [45, 48, 63, 76, 77] follow the two-phase paradigm proposed in FORA [77]. In particular, they first invoke Forward-Push [4] with early termination conditions to derive rough approximations of DPPR values, and then refine results by exploiting random walk samplings [22] to estimate the error term in Eq. (3). To illustrate, consider the r.h.s. graph in Fig. 4. Instead of always using pushes, FORA-based methods would terminate at nodes v_4–v_7 and simulate random walks from v_4–v_7 as per their residues to probe the far-reaching nodes.

Unfortunately, FORA-based solutions still entail considerable computational overheads if applied directly to DPPR approximation for two reasons. First, although SSPPR approximation solutions can improve the efficiency of single-source DPPR computation, we need $O(k^{\ell+1})$ (up to n in the worst case) single-source DPPR queries for $O(k^{\ell+1})$ leaf nodes in the specified level-$(\ell+1)$ supernode $\mathcal{S}$, which is costly when ℓ is large. Second, most push operations or random walk samples in such methods are futile in the context of DPPR computation. This is due to the fact that these methods aim to obtain accurate DPPR estimates of *all* nodes w.r.t. the source. However, the majority of visited nodes by such operations are not in the supernode $\mathcal{S}$ that we are interested in visualizing. Notably, at level-1, roughly $\frac{n-k}{n}$ of nodes located outside $\mathcal{S}$, where k is a small integer (e.g., 25) and n can be up to millions. To illustrate, consider the example in Fig. 5. For simplicity, we assume that supernode $\mathcal{S}$ is at level-1, and it contains 12 leaf nodes v_0–v_{11}. As shown in the l.h.s. graph in Fig. 5,

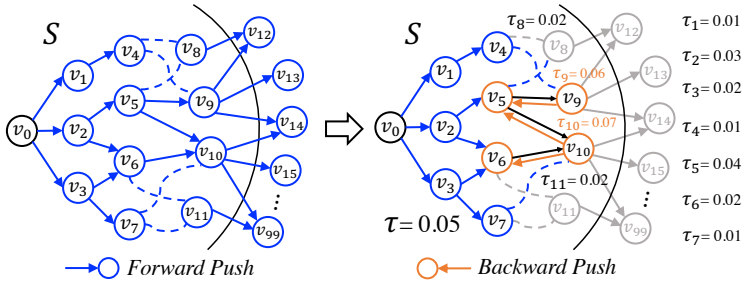


Fig. 5. Bidirectional push in Tau-Push.

Forward-Push iteratively performs push operations until the estimated DPPR values of all nodes, including 88 nodes v_{12}–v_{99} outside $\mathcal{S}$, satisfied the desired approximation.

4 Tau-Push ALGORITHM

In this section, we present Tau-Push, the efficient algorithm for estimating level-ℓ PDist via computing (ϵ, δ)–approximate level-ℓ DPPR. At a high level, Tau-Push overcomes the limitations of FORA-based methods through (i) a filter-refinement paradigm, which eliminates redundant push operations or random walks without affecting the accuracy guarantee, and (ii) a grouped push strategy, which reduces the number of leaf-level invocations from $O(k^{\ell+1})$ to $O(k)$. In what follows, we first illustrate the main idea of Tau-Push in Section 4.1, followed by the algorithmic details of two subroutines of Tau-Push in Section 4.2. We further provide theoretical results of Tau-Push in Section 4.3.

4.1 Main Idea and Algorithm

Tau-Push estimates level-ℓ DPPR values by pushing in a bidirectional fashion. In particular, Tau-Push first performs a small number of forward push operations from the source $\mathcal{V}_i$ to derive an accurate DPPR for each target node in the vicinity of $\mathcal{V}_i$. Next, it conducts a handful of pushes reversely along in-neighbors, referred to as *backward push* operations, from the rest of target nodes in $\mathcal{S}$ (i.e., the nodes far-reaching from $\mathcal{V}_i$), to obtain their approximate DPPR values. To explain the backward push, we consider the r.h.s. graph in Fig. 5. Given $\alpha = 0.1$ and node v_{10} with initial residue $r(v_{10}, v_{10}) = 1$, we will convert 0.1 fraction of $r(v_{10}, v_{10})$ to its estimated DPPR and propagate the remaining 0.9 to its in-neighbors v_5 and v_6, which receive $\frac{0.9}{d(v_5)} = 0.45$ and $\frac{0.9}{d(v_6)} = 0.9$, respectively. By combining forward and backward pushes, we can avoid the issue of "pushing too deeply" mentioned in Section 3.3, and hence, prune excessive pushes/samples for nodes outside $\mathcal{S}$, e.g., nodes v_{12}–v_{99} in Fig. 5. In other words, Tau-Push filters out the majority of nodes whose approximate DPPR values obtained by Forward-Push are sufficiently accurate, and only refines the DPPR estimation for each remaining node using backward pushes.

The linchpin to enabling the aforementioned filter-refinement optimization is the identification of nodes that requires backward pushes. In relation to this, we introduce the notion of *degree-normalized PageRank* (DPR). For each supernode $\mathcal{V}_j$, its DPR is defined as

$$\tau_j = \frac{1}{m \cdot |F(\mathcal{V}_j)|} \sum_{v_t \in F(\mathcal{V}_j)} \sum_{v_k \in V} \pi_d(v_k, v_t), \tag{4}$$

where $\pi_d(v_k, v_j)$ is the DPPR of v_j w.r.t. v_k and $F(\mathcal{V}_j)$ signifies the set of leaf nodes in $\mathcal{V}_j$. In particular, when $\mathcal{V}_j$ is a leaf node v_j (i.e., $F(\mathcal{V}_j) = v_j$), its DPR τ_j is the summation of DPPR values pertinent to v_j, which indicates the global importance of v_j in the entire graph. Building

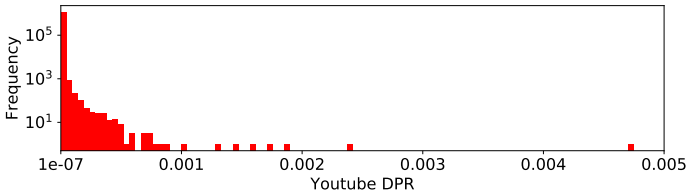


Fig. 6. The distributions of DPR on *Youtube*.

on our theoretical analysis, if the residue threshold r_{max} used in Forward-Push is set properly, the approximate DPPR values of nodes v_j with τ_j less than the given DPR threshold τ are guaranteed to be (ϵ, δ)-approximate (see Lemma 4.1). For example, the r.h.s. graph in Fig. 5 shows that, given $\tau = 0.05$, nodes v_1–v_8 and v_{11} can be filtered out as their DPR values are less than τ, and hence, we only need to conduct backward pushes from v_9 and v_{10}. We observe that DPR values of real-world graphs often follow the power law distribution, where only a few nodes have large values and the remaining nodes have nominal values. For instance, as shown in Fig. 6, the DPR values of over 99.98% of nodes on the well-known *Youtube* graph are less than 0.001, while the largest DPR is about 0.005. Hence, by checking DPR and choosing a proper τ, we can exclude the great majority of nodes in $\mathcal{S}$ for further refinement, thereby reducing the computational cost. It is worth mentioning that the DPR of each leaf node can be efficiently calculated in the preprocessing stage and used as the input to Tau-Push.

Notice that the supernode $\mathcal{S}$ considered above is a level-1 supernode with k leaf nodes. As for any level-$(\ell+1)$ supernode $\mathcal{S}$ containing $O(k^{\ell+1})$ leaf nodes, we need to invoke Forward-Push $O(k^{\ell+1})$ times for the computation of all-pair level-ℓ DPPR values in $\mathcal{S}$, which is rather time-consuming when ℓ is large. To alleviate this issue, we design Group Forward-Push (GFP), an improved and optimized Forward-Push equipped with a grouped push strategy: it starts push operations from all leaf nodes in the level-ℓ supernode simultaneously instead of separately. Analogously, we also propose Group Backward-Push (GBP) for accelerating backward push operations in level-ℓ supernodes.

Algorithm. Algorithm 1 shows the pseudo-code of Tau-Push, which consists of two main phases: GFP and GBP. Specifically, given a supernode $\mathcal{S}$ and constant k, Tau-Push starts by setting DPR threshold τ as $1/\sqrt{k \cdot n}$ and forward residue threshold as

$$r_{max} = \frac{\epsilon \cdot \delta}{m \cdot \tau} \tag{5}$$

at Lines 1-2. The setting of τ strikes a good balance between forward and backward phases, leading to an optimized worst-case time complexity (as analyzed later in Section 4.3). Next, for each supernode $\mathcal{V}_i \in \mathcal{S}$, Tau-Push invokes GFP with residue threshold r_{max} from $\mathcal{V}_i$ to derive $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$, a rough estimation of DPPR for each supernode $\mathcal{V}_j \in \mathcal{S}$ (Lines 3-4). Subsequently, Tau-Push calculates the residue threshold r^b_{max} used in GBP as per the following equation (Line 5):

$$r^b_{max} = \frac{\epsilon \cdot \delta}{\max_{\mathcal{V}_i \in \mathcal{S} \setminus \mathcal{V}_j} \frac{\sum_{v_s \in F(\mathcal{V}_i)} d(v_s)}{|F(\mathcal{V}_i)|}}, \tag{6}$$

where the denominator signifies the maximum average degree of leaf nodes in each supernode $\mathcal{V}_i$ of $\mathcal{S}$. Tau-Push identifies all supernodes $\mathcal{V}_j$ satisfying DPR $\tau_j > \tau$ and utilizes GBP with residue threshold r^b_{max} to refine the approximate DPPR value of every supernode pair $\mathcal{V}_i, \mathcal{V}_j$ in $\mathcal{S}$ (Lines 6-7). Eventually, for each supernode pair $\mathcal{V}_i, \mathcal{V}_j \in \mathcal{S}$, the level-$\ell$ approximate DPPR is converted to the desired level-ℓ approximate PDist (Lines 8-9).

Algorithm 1: Tau-Push

Input: Graph G, supernode $\mathcal{S}$ and constant k.
Output: Estimated $\widehat{\Delta}[i,j], \forall \mathcal{V}_i, \mathcal{V}_j \in \mathcal{S}$.

Set DPR threshold $\tau \leftarrow 1/\sqrt{k \cdot n}$;
Set forward residue threshold r_{max} by Eq. (5);
for *each supernode* $\mathcal{V}_i \in \mathcal{S}$ **do**
    $\forall \mathcal{V}_j \in \mathcal{S}$, $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) \leftarrow$GFP $(G, \mathcal{S}, \mathcal{V}_i, r_{max})$;
Set backward residue threshold r^b_{max} according to Eq. (6);
for *each supernode* $\mathcal{V}_j \in \mathcal{S}$ *such that* $\tau_j > \tau$ **do**
    $\forall \mathcal{V}_i \in \mathcal{S}$, $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) \leftarrow$GBP $(G, \mathcal{S}, \mathcal{V}_j, r^b_{max})$;
for *each supernode pair* $\mathcal{V}_i, \mathcal{V}_j \in \mathcal{S}$ **do**
    Convert DPPR $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ to PDist $\widehat{\Delta}[i,j]$ by Eq.(1);

Algorithm 2: GFP

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

1 Initialize approximate DPPR $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) \leftarrow 0 \ \forall \mathcal{V}_j \in \mathcal{S}$;
2 Initialize residues $r(\mathcal{V}_i, v_j) \leftarrow \frac{d(v_j)}{|F(\mathcal{V}_i)|} \ \forall v_j \in F(\mathcal{V}_i)$;
3 **while** $\exists v_k \in V$ *such that* $r(\mathcal{V}_i, v_k) > d(v_k) \cdot r_{max}$ **do**
4   **if** v_k *is in any supernode* $\mathcal{V}_j \in \mathcal{S}$ **then**
5         $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) \leftarrow \widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) + \frac{\alpha \cdot r(\mathcal{V}_i, v_k)}{|F(\mathcal{V}_j)|}$;
6   **for** *each out-neighbor* v_j *of* v_k **do**
7         $r(\mathcal{V}_i, v_j) \leftarrow r(\mathcal{V}_i, v_j) + (1-\alpha) \cdot \frac{r(\mathcal{V}_i, v_k)}{d(v_k)}$;
8     $r(\mathcal{V}_i, v_k) \leftarrow 0$;

4.2 GFP and GBP

In the following, we elaborate on the algorithmic details of GFP and GBP used in Tau-Push.

GFP. Algorithm 2 shows the pseudo-code of GFP. Akin to Forward-Push, GFP maintains two lists of variables in the course of pushes from source supernode $\mathcal{V}_i$: (i) the estimated DPPR $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$, $\forall \mathcal{V}_j \in \mathcal{S}$, and (ii) the residue $r(\mathcal{V}_i, v_k)$, $\forall v_k \in V$. Initially, all variables are set to 0 except that $r(\mathcal{V}_i, v_j)$ is set to $\frac{d(v_j)}{|F(\mathcal{V}_i)|}$ for each node v_j in the leaf node set $F(\mathcal{V}_i)$ of $\mathcal{V}_i$ (Lines 1-2). After that, GFP starts an iterative process to traverse G from nodes with non-zero residues, i.e., nodes in $F(\mathcal{V}_i)$. In particular, in each iteration, it inspects the residue of each node in V to identify a node v_k whose residue is greater than $d(v_k) \cdot r_{max}$. If such a node v_k exists, GFP first adds $\frac{\alpha \cdot r(\mathcal{V}_i, v_k)}{|F(\mathcal{V}_j)|}$ to the approximate DPPR $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$, where $\mathcal{V}_j$ is the supernode containing v_k (Lines 4-5). Then, it evenly distributes the remaining $(1-\alpha)$ fraction of residue to the out-neighbors of v_k (Lines 6-7). Afterwards, GFP resets the residue $r(\mathcal{V}_i, v_k)$ to 0 (Line 8) and proceeds to next iteration. Lemma 4.1 indicates the correctness of Algorithm 2.

Lemma 4.1. *Given a source supernode* $\mathcal{V}_i \in \mathcal{S}$ *and a* DPR *threshold* τ*, by setting* r_{max} *as in Eq.* (5)*, GFP returns* (ϵ, δ)*−approximate level-*ℓ DPPR $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ *for* $\mathcal{V}_j \in \mathcal{S}$ *with* DPR $\tau_j \leq \tau$.

Algorithm 3: GBP

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

1 Initialize approximate DPPR $\widehat{\pi}(\mathcal{V}_i, \mathcal{V}_j) \leftarrow 0$ $\forall \mathcal{V}_i \in \mathcal{S}$;
2 Initialize residues $r(v_i, \mathcal{V}_j) \leftarrow 1/|F(\mathcal{V}_j)|$, $\forall v_i \in F(\mathcal{V}_j)$;
3 **while** $\exists v_k \in V$ *such that* $r(v_k, \mathcal{V}_j) > r^b_{max}$ **do**
4 **if** v_k *is in any supernode* $\mathcal{V}_i \in \mathcal{S}$ **then**
5 $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) \leftarrow \widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j) + \frac{\alpha \cdot d(v_k) \cdot r(v_k, \mathcal{V}_j)}{|F(\mathcal{V}_i)|}$;
6 **for** *each in-neighbor* v_i *of* v_k **do**
7 $r(v_i, \mathcal{V}_j) \leftarrow r(v_i, \mathcal{V}_j) + (1-\alpha) \cdot \frac{r(v_k, \mathcal{V}_j)}{d(v_i)}$;
8 $r(v_k, \mathcal{V}_j) \leftarrow 0$;

GBP. GBP can be regarded as the backward counterpart of GFP. As shown in Algorithm 3, GBP initializes residue value $r(v_i, \mathcal{V}_j)$ of each leaf node v_i in target supernode $\mathcal{V}_j$ to $1/|F(\mathcal{V}_j)|$ (Line 2). Distinct from Forward-Push, GBP conducts the graph traversal from the target supernode $\mathcal{V}_j$, following the incoming edges of each node. To be more precise, GBP evenly pushes $(1-\alpha)$ fraction of residue $r(v_k, \mathcal{V}_j)$ to the in-neighbors of current node v_k (Lines 6-7), rather than out-neighbors in GFP. In addition, another two minor differences are: (i) the residue threshold r^b_{max} is defined as Eq. (6) (Line 3), and (ii) in each iteration, GBP increases the estimated DPPR $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ by $\frac{\alpha \cdot d(v_k) \cdot r(v_k, \mathcal{V}_j)}{|F(\mathcal{V}_i)|}$, where $\mathcal{V}_i$ is the supernode consisting of current node v_k (Lines 4-5). Lemma 4.2 proves the correctness of GBP.

Lemma 4.2. *Given a target supernode* $\mathcal{V}_j \in \mathcal{S}$ *and a threshold* r^b_{max} *in Eq.* (6)*, GBP returns the* (ϵ, δ)*-approximate level-*ℓ DPPR $\widehat{\pi}_d(\mathcal{V}_i, \mathcal{V}_j)$ *for each source supernode* $\mathcal{V}_i \in \mathcal{S}$ *and* $\mathcal{V}_i \neq \mathcal{V}_j$.

4.3 Theoretical Results

Correctness. Combining Lemmata 4.1 and 4.2 leads to Theorem 4.3, which establishes the correctness of Tau-Push (Algorithm 1).

Theorem 4.3. *For any user-selected supernode* $\mathcal{S}$ *and threshold* τ*, by setting* r_{max} *and* r^b_{max} *as in Algorithm 1, 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* $\mathcal{V}_i \neq \mathcal{V}_j$.

Time complexity. The worst-case time complexity of Tau-Push is $O\left(\frac{k \cdot n \cdot m \cdot \tau}{\epsilon \cdot \delta} + \frac{m}{\tau \cdot \epsilon \cdot \delta}\right)$. By setting $\tau = 1/\sqrt{k \cdot n}$, the above complexity is minimized to $O\left(\frac{k \cdot m}{\epsilon} \cdot \sqrt{k \cdot n}\right)$. By employing GFP only, the worst-case complexity is $O\left(\frac{k^2 \cdot n \cdot m}{\epsilon}\right)$, which is $\sqrt{k \cdot n}$ times slower than Tau-Push. The expected complexity of Tau-Push for a randomly selected supernode $\mathcal{S}$ is $O\left(\sum_{\mathcal{V}_i \in \mathcal{S}} \frac{d(\mathcal{V}_i)}{|F(\mathcal{V}_i)|} \cdot \frac{k \cdot m}{\epsilon \cdot \delta \cdot n}\right)$.

Indexing scheme. We first store $O(n)$ DPR values as the index, which can be efficiently precomputed in a similar way to global PageRank by setting the k-th entry in the initial global PageRank as $\frac{d(v_k)}{m}$. Notice that GBP is only conducted from target $\mathcal{V}_j$ with $\tau_j > \tau = 1/\sqrt{k \cdot n}$ and is independent of the query supernode $\mathcal{S}$. Hence, the index space is $O\left(k \cdot \sqrt{k \cdot n}\right)$ since GBP

Table 2. Index space and time complexity of methods for approximate level-ℓ PDist computation.

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

estimates DPPR of $O\left(\sqrt{k \cdot n}\right)$ target nodes w.r.t. $O(k)$ source supernodes in $\mathcal{S}$. Overall, the index space of Tau-Push is $O\left(n + k \cdot \sqrt{k \cdot n}\right)$.

Tau-Push vs. FORA. We compare Tau-Push with FORA in terms of time complexity and index space for a randomly selected supernode $\mathcal{S}$. As summarized in Table 2, Tau-Push improves the time complexity of FORA by $\frac{n \cdot \sqrt{k \cdot m}}{k^3 \cdot \log n}$, where k is the number of supernodes/nodes to be visualized and is usually small as discussed in Section 2.2. For example, Tau-Push is about four orders of magnitude faster than FORA on the *Youtube* graph with 1 million nodes and 3 million edges. Besides that, we observe that the indexing space of FORA is usually 3–5× larger than that of Tau-Push in the experiments.

5 PPRviz

This section presents PPRviz, our framework for static graph visualization. Fig. 7 illustrates the procedure of PPRviz, which consists of two phases, i.e., (i) preprocessing and (ii) interactive visualization. In the preprocessing phase, it first constructs a supergraph hierarchy for the given graph G, then builds the index for the proposed Tau-Push. It is worth noting that the preprocessing phase is conducted only once for any queries on the given G. In the interactive visualization phase, PPRviz contains two stages, namely PDist matrix computation and position matrix embedding. As the index schema of Tau-Push has been explained in Section 4.3, we focus on illustrating the remaining stages as follows.

- **Supergraph hierarchy construction**: PPRviz first constructs a supergraph hierarchy of the input graph in the preprocessing step [1, 5, 64]. Take Fig. 7(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. We employ the Louvain algorithm [13] to construct the supergraph hierarchy. Specifically, Louvain first treats all level-ℓ supernodes and their relationships as a new graph and assigns each supernode to a stand-alone partition. After that, it builds the partitions of level-ℓ supernodes (i.e., supernodes at level-$(\ell + 1)$) by iteratively moving each supernode to the neighbor's partition with maximum modularity improvement, where the moduarlity [52] measures the density of links inside partitions as compared to links between partitions. Following the size requirement in Section 2.2, we adapt Louvain under the constraint that each supernode $\mathcal{S}$ (resp. the coarsest supergraph) should have at most k children (resp. supernodes) and call this adaption Louvain+. Note that k can be configured by users according to their needs.
- **PDist matrix computation**: During interactive visualization, a user can select a level-$(\ell + 1)$ supernode $\mathcal{S}$ (marked by an arrow in Fig. 7(b)). To visualize the intra-structure of $\mathcal{S}$, i.e., the shaded area in the middle layer, PPRviz leverages Tau-Push in Algorithm 1 to calculate the PDist matrix $\mathbf{\Delta} \in \mathbb{R}^{k \times k}$ for the k children of S (Fig. 7(c)), which reflects the theoretical graph distance between the children by utilizing information of the related nodes in the input graph, i.e., the shaded area in the bottom layer of Fig. 7(b).
- **Position matrix embedding**: Given the PDist 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 Fig. 7(d)), where $\mathbf{X}[i] \in \mathbb{R}^2$ represents the two-dimensional

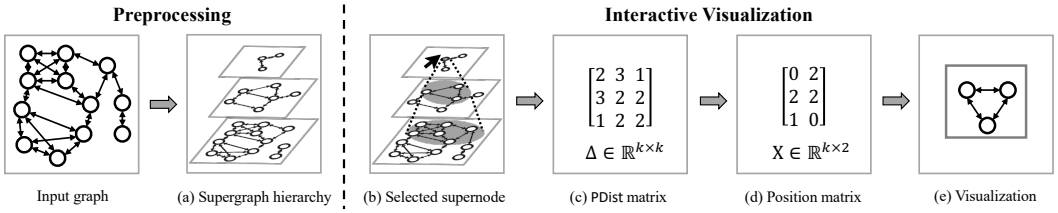


Fig. 7. Procedure of PPRviz.

coordinate of the i-th child node on the screen. In particular, the position matrix embedding step solves the following optimization problem [26]:

$$\arg\min_{\mathbf{X}} L(\mathbf{X}|\Delta) = \sum_{i<j} \left(1 - \frac{||\mathbf{X}[i] - \mathbf{X}[j]||}{\Delta[i,j]}\right)^2. \tag{7}$$

Intuitively, 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 PDist $\Delta[i,j]$. Towards this end, PPRviz employs the standard method for solving Eq. (7), namely, the *stress majorization* technique [26]. The time complexity of this method is $O(k^3)$ [26], which is insignificant as k is usually small to avoid visual clutter [35].

Applications and future works. The proposed framework PPRviz is not limited to multi-level visualization and is applicable for visualizing static homogeneous graphs in other scenarios. For example, PPRviz can visualize the entire graph or the subgraph returned by the graph query in a single-level fashion [11], where PPRviz skips the supergraph hierarchy construction stage and sets $k = n$ for the visualization phase. Furthermore, PPRviz can be employed for incremental exploration [31], where users can move a focal area over the entire graph to explore the subgraph they are interested in. Notice that PPRviz can easily cope with the small dynamic graphs (including subgraphs, motifs, etc.) returned by database queries because our proposed Tau-Push algorithm is highly efficient and can be real-time responsive when a new visualization request is made. Regarding the scenarios on large dynamic or attributed graphs, even though existing solutions [78, 85] are applicable for the supergraph hierarchy construction and position embedding stages of PPRviz, the proposed PDist or corresponding estimation algorithm Tau-Push fails to extend to these scenarios trivially. In particular, under the dynamic setting, it is still unclear how to efficiently estimate the proposed PDist with a rigorous theoretical accuracy guarantee. In addition, PPRviz can be extended to visualize attributed or knowledge graphs by treating node/edge attributes as additional nodes [81]. However, such a method requires a new distance measure considering both topologies and attributes. These challenges motivate us to design new approaches in future works.

6 RELATED WORK

Graph visualization. Conventional single-level graph visualization methods have been extensively studied [27, 31, 34, 72] and can be classified into two main categories: (i) *force-directed methods*, e.g., FR [23], LinLog [53], ForceAtlas [36] and others [16, 21, 36, 49]; and (ii) *stress methods*, e.g., CMDS [26], PMDS [15] and others [25, 39, 50, 70]. In particular, force-directed methods model a graph as a force system, where adjacent nodes attract each other and all nodes repulse each other. The position matrix is derived by minimizing the composite forces in the entire system. Stress methods utilize the shortest distance as the node distance to guide node placement. They embed the shortest distance matrix into a position matrix by optimization techniques, e.g., gradient descent [25] and stress majorization [26]. To boost efficiency, many optimizations are incorporated, e.g., grid-based partitioning [23, 49], quad-tree [36, 53], and dimensionality reduction [15]. Moreover,

as surveyed by [28], graph embedding methods, e.g., GFactor [3], SDNE [73], LapEig [9], LLE [60] and Node2vec [29], can also be applied for visualization by treating the embedding matrix with dimension being 2 as the position matrix [28, 56, 67]. Multi-level methods [1, 5, 33, 49, 50, 59, 64] mainly focus on designing graph clustering algorithms, and the single-level methods are directly employed to layout each cluster. For example, OpenOrd [49] clusters nodes based on their Euclidean distances in the graph layout and uses FR for visualization; KDraw [50] applies label propagation for clustering and uses [25] for visualization; GrouseFlocks [5] groups nodes based on certain graph structures. Compared with our PDist distance matrix, the distance measures employed in existing methods fail to preserve the topological information comprehensively and hence dampen visualization quality. Specifically, force-directed methods only consider the direct links in the graph. Stress methods consider the shortest path from the source node to the target node but ignore other intricate paths. A related work [16] uses the force-directed method for visualization after determining the edge weights by PPR; however, this method is still inherently a force-directed method, hence suffering from the corresponding limitations.

PPR computation. The efficient computation of PPR has been extensively studied [4, 22, 32, 38, 45–47, 63, 65, 74–77, 79, 82, 83]. Among these works, BEAR [65] and BePI [38] improve the power iteration method [55] and achieve high efficiency by indexing several large matrices. However, the index space limits their applicability to large graphs. BiPPR [46] and HubPPR [75] combine random walks [22] with Backward-Push [47], and are subsequently improved by FORA [77]. However, FORA and its improved solutions [32, 45, 76, 79] suffer from numerous push operations and ineffective sample problems as illustrated in Section 3.2. This is because our level-ℓ PDist computation aims at the aggregated PPRs between two clusters, which is essentially different from PPR computation for a single source node. Another line of work computes PPR using the idea of particle filtering [24], which performs a deterministic graph traversal from a set of source nodes. Nevertheless, it is non-trivial to determine the initial particle distribution for level-ℓ DPPR, and it does not offer quality guarantees. In contrast, Tau-Push can significantly outperform existing visualization and PPR solutions in efficiency with quality guarantees.

7 EXPERIMENT EVALUATION

We introduce the experiment settings in Section 7.1. Section 7.2 evaluates the visualization quality of our PDist and the competing methods. Sections 7.3 and 7.4 compare the efficiency of Tau-Push against the visualization and PPR-based solutions. Interested readers are referred to [84] for additional results and analysis on visualization and Tau-Push's variants.

7.1 Experiment Settings

Competitors and parameter settings. We compare PPRviz with 13 representative graph visualization methods from different categories:

- single-level force-directed methods: FR [23], LinLog [53], ForceAtlas [36];
- single-level stress methods: CMDS [26], PMDS [15];
- single-level embedding methods: GFactor [3], SDNE [73], LapEig [9], LLE [60], Node2vec [29];
- an SimRank-based adaptation mentioned in Section 3.1;
- multi-level methods: OpenOrd [49] and KDraw [50].

We follow the parameter settings of all competitors as recommended in their respective papers. For PPRviz, we set the maximum number of nodes in a cluster to 25 (i.e., $k = 25$) as suggested in [35]. For a fair comparison, we follow [5, 59] and modify OpenOrd and KDraw such that only the partial view of the clusters $\mathcal{S}$ in the zoom-in path is visualized. Since OpenOrd does not allow

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

Dataset	n	m	Description
TwEgo	23	52	Ego network[44]
FbEgo	52	146	Ego network[44]
Wiki-ii	186	632	Authorship network[41]
Physician	241	1.8K	Social network[41]
FilmTrust	874	2.6K	User trust network[41]
SciNet	1.5K	5.4K	Collaboration network[41]
Amazon	334.9K	1.9M	Product network [44]
Youtube	1.1M	6.0M	Social network [44]
Orkut	3.1M	234.4M	Social network [44]
DBLP	5.4M	17.2M	Collaboration network [41]
It-2004	41.3M	2.3B	Crawled network [14]
Twitter	41.7M	3.0B	Social network [42]

cluster size constraint and KDraw employs a complicated method to determine cluster size, we set the maximum number of supernodes in the coarsest supergraph (instead of all levels) to k. We observe that PPRviz usually shows more nodes than OpenOrd and KDraw in a visualization, and hence the efficiency of PPRviz is not caused by processing fewer nodes. Additionally, we compare our proposed Tau-Push against 4 PPR approximation solutions PI [55], FORA [77], FORA+ [76] and ResAcc [45]. For all methods, we set $\epsilon = 1 - 1/e$ and $\delta = 1/(10k)$ by default. For FORA-based competitors FORA, FORA+, and ResAcc, we set the initial residue $r(v_i, v_i) = d(v_i)$ for the source each v_i and follow the settings of other parameters as suggested in the original papers, where the correctness of PDist estimation is still satisfied. The experiment source code and the implementation of PPRviz are available at https://github.com/jeremyzhangsq/PPRviz-reproducibility/.

Datasets and performance metrics. We use the 12 real-world graphs in Table 3 for the experiments. We generate visualizations in a single-level fashion on 6 smaller graphs, and use ND and ULCV to evaluate the visualization quality of PPRviz and the competitors. For a fair comparison, we follow NetworkX [30] 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 1000 seconds (resp. 12 hours). All experiments are conducted on a Linux machine with Intel Xeon(R) Gold 6240@2.60GHz CPU and 80GB RAM in single-thread mode. Note that the memory utilized by PPRviz is comparable to that used by storing the input graph.

7.2 Visualization Quality

We evaluate the visualization quality of PPRviz and the competitors quantitatively via two popular aesthetic metrics, and qualitatively via a user study and a case study on 6 smaller graphs.

7.2.1 **Aesthetic Metrics.** We report the ND and ULCV scores of all approaches in Table 4 and Table 5, respectively. Since OpenOrd applies FR to visualize each supergraph, we treat them as one method and report their results in one column. The results of KDraw are omitted as it fails to process graphs with multiple connected components. We use "-" and ∞ to indicate undefined and

Table 4. ND of PPRviz and baselines, the best in bold and the second best in italic, ∞ indicates infinity.

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

Table 5. ULCV of PPRviz and baselines, the best in bold and the second best in italic, "-" indicates undefined.

	TwEgo	*FbEgo*	*Wiki-ii*	*Physician*	*FilmTrust*	*SciNet*
PPRviz	**0.22**	**0.39**	**0.35**	**0.45**	**0.48**	**0.34**
OpenOrd/FR	0.35	*0.42*	*0.41*	0.53	*0.54*	0.77
LinLog	0.57	0.67	1.09	0.90	1.99	4.70
ForceAtlas	0.37	0.49	0.64	0.55	0.96	1.52
CMDS	0.40	0.46	0.62	0.80	1.05	1.74
PMDS	*0.23*	0.45	0.78	*0.47*	0.69	*0.74*
GFactor	0.45	0.91	0.62	0.95	0.64	0.86
SDNE	1.96	0.94	0.94	1.67	1.31	1.72
LapEig	1.15	0.98	1.04	1.02	1.70	1.26
LLE	0.46	0.77	1.27	0.77	0.87	-
Node2vec	0.80	0.96	0.86	1.41	0.89	1.32
SimRank	0.84	0.75	0.53	0.53	1.78	1.98

infinity scores, respectively. Table 4 shows that PPRviz consistently outperforms the competitors in ND on all graphs except *TwEgo* and *FbEgo*, where PPRviz has comparable performance to the best method FR. Note that FR achieves the smallest ND scores on *TwEgo* and *FbEgo* because it places nodes of the largest connected component far apart from each other, which causes the edge distortion issue. In particular, the ND scores of ForceAtlas, FR and LinLog are 5, 2, and 3 orders of magnitude larger than PPRviz on *SciNet*, respectively. Regarding stress methods, we find that CMDS and PMDS yield infinite ND scores, indicating severe node overlapping issues. This is because they use the shortest distance between two nodes as the node distance, which only takes a few discrete values and thus fails to distinguish from different node pairs. Furthermore, PMDS computes the position of a non-pivot node as the weighted combination of its connected pivot nodes. Thus, degree-one non-pivot nodes connected to the same pivot will share the same position. The graph embedding methods, especially SDNE and LapEig, have worse ND scores than PPRviz and other competitors, as embedding-based methods are specially designed for machine learning tasks without considering visualization quality. From Table 5, we can see that PPRviz always performs the best in ULCV. For instance, on *SciNet*, PPRviz is 14× better than LinLog. The superior performance of PPRviz is attributed to our carefully designed PDist.

7.2.2 **User Study.** In the second set of experiments, we conduct a user study to evaluate the visualization quality of PPRviz. This study aims to answer two questions: (i) does PPRviz output better visualization results than the competitors, and (ii) does the approximate level-ℓ PDist computation

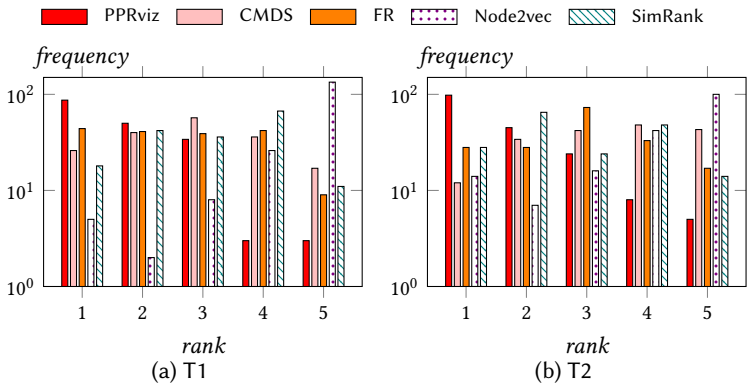


Fig. 8. Results of T1 and T2, frequency of selected ranking.

Table 6. Results of T3, frequency of being selected.

	Tau-Push	PI	No difference
Frequency	54	43	**83**

in PPRviz affect visualization quality from the perspective of human 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.

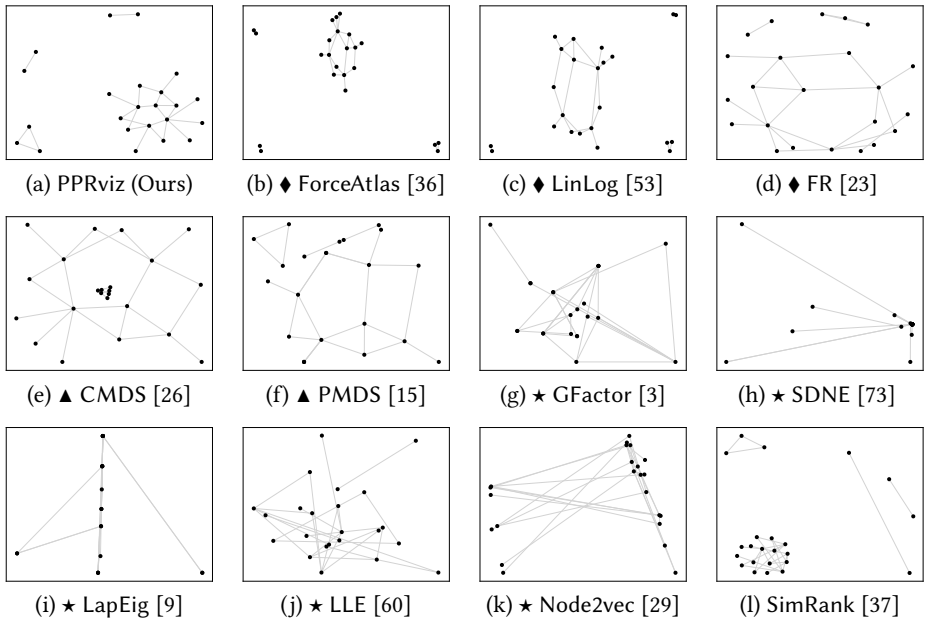
To answer the first question, we generate 6 groups of visualizations, each of which is obtained on one of the 6 small graphs using PPRviz and 4 representative competitors, including FR, CMDS, Node2vec, and SimRank, that achieve relatively good performance in Section 7.2.1. In each group, the 5 visualizations generated by different methods are randomly shuffled before presenting to the participants. Following the assessment paradigm in graph drawing [19, 43, 80], the participants are asked to rank the visualizations in terms of readability and cluster structure, which are identified as two paramount tasks by [8, 27] and are described as follows.

- Task 1 (T1): rank the 5 visualizations in each group from the highest (1) to the lowest readability (5), where high readability means that the graph elements are well displayed.
- Task 2 (T2): rank the 5 visualizations in each group from the best (1) to the worst (5) in terms of the structure layout, where a good layout means that clusters and strongly connected nodes can be clearly observed.

To answer the second question, we compare the visualizations generated by employing Tau-Push in PPRviz with those by employing the near-exact solution PI in PPRviz. Since PI does not scale to large graphs, we consider two small graphs: *FilmTrust* and *SciNet*. For both methods, we vary k in $\{15, 20, 25\}$. For each supergraph, we generate 2 visualizations using Tau-Push and PI and organize them as a group. The visualizations in each group are displayed to the participants without telling them the methods, and the participants are asked to conduct the following task.

- Task 3 (T3): for the 2 visualizations in a group, select the one with superior visualization quality.

Note that for each task, we collected 180 instances (30 participants $\times$ 6 groups). Fig. 8(a) (resp. Fig. 8(b)) reports the ranking frequency of the visualization generated by each approach in T1 (resp. T2). In particular, the rankings of PPRviz are concentrated on top-1 and top-2 for both tasks. This result demonstrates PPRviz's sound quality in terms of readability and cluster inspection, which is consistent with our ND and ULCV results reported in Section 7.2.1. Regarding the results of T3, we find in Table 6 that the participants cannot tell the quality difference between the visualizations



(a) PPRviz (Ours) (b) ♦ ForceAtlas [36] (c) ♦ LinLog [53] (d) ♦ FR [23]

(e) ▲ CMDS [26] (f) ▲ PMDS [15] (g) ★ GFactor [3] (h) ★ SDNE [73]

(i) ★ LapEig [9] (j) ★ LLE [60] (k) ★ Node2vec [29] (l) SimRank [37]

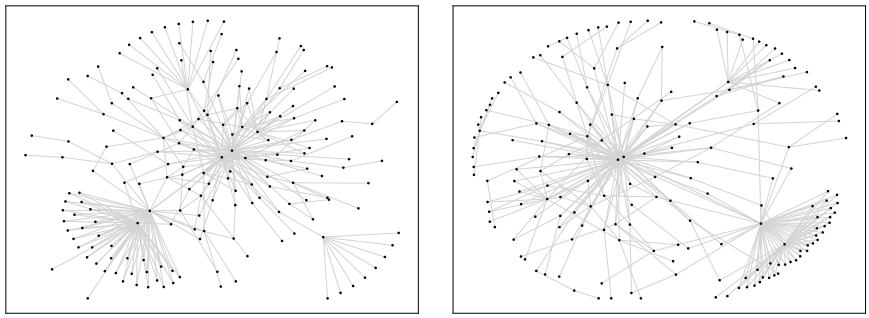
Fig. 9. Visualization results and aesthetic metrics for the *TwEgo* graph: force-directed methods are marked with ♦; stress methods are marked with ▲; graph embedding methods are marked with ★.

generated by Tau-Push and PI in most cases, and the times the two methods are selected as the best are comparable, demonstrating that the approximate level-ℓ PDist computation in PPRviz does not affect its visualization quality.

7.2.3 **Case Study.** At last, we conduct a case study to compare the visualizations of PPRviz and other competitors. In Fig. 9, we report the results of PPRviz and all competitors on *TwEgo*. Besides that, we also compare the results of PPRviz and the best competitor FR on two large graphs *Wiki-ii* and *FilmTrust*, which are shown in Fig. 10 and Fig. 11, respectively. We find that the comparison results of visualization are consistent with those of metrics in Section 7.2.1. More concrete, PPRviz yields a high-quality visualization, which clearly organizes the graph into a well-connected cluster and several cliques. In contrast, the competitors suffer from issues such as node overlapping and edge distortion. Furthermore, as shown in Fig. 9(d) and Fig. 11(b), the structures of small-size clusters yielded by the best competitor FR are hard to recognize. This is because nodes in the small-size clusters are more likely to be affected by the repulsive force from the large cluster, making all of them huddle together.

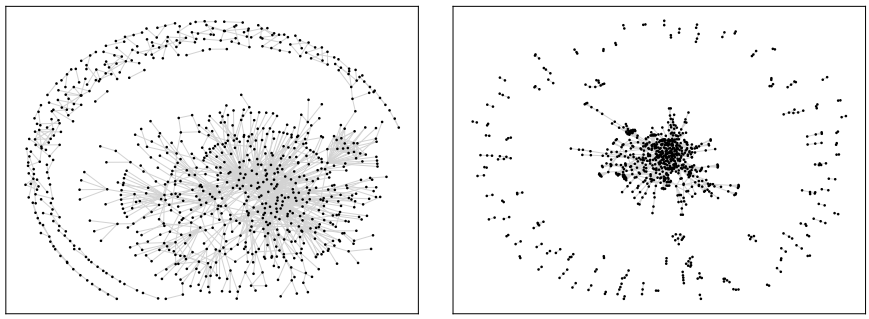
7.3 Visualization Efficiency

Response time. Fig. 12 shows the response time on the 6 larger graphs, including *Amazon*, *Youtube*, *Orkut*, *DBLP*, *It-2004*, and *Twitter*. We only report the results of PPRviz, OpenOrd, KDraw, PMDS, and Node2vec, and omit the rest of the competitors as they fail to terminate within 1000 seconds for all datasets. We can observe that PPRviz consistently outperforms all competitors in terms of response time. Specifically, PPRviz outputs the visualization result within 1 second on all datasets. For example, PPRviz costs 0.63 seconds on the *Twitter* graph 3 billion edges, while all competing methods take more than 1000 seconds. Regarding the competitors, PMDS only computes the positions of several pivot nodes and determines the positions of other nodes by linear combinations



(a) PPRviz (Ours) (b) ♦ FR [23]

Fig. 10. Visualization of PPRviz and FR on *Wiki-ii*.



(a) PPRviz (Ours) (b) ♦ FR [23]

Fig. 11. Visualization of PPRviz and FR on *FilmTrust*.

of the pivot nodes. However, it can only visualize the entire *Amazon* graph in 45 seconds. In addition, Node2vec incurs costly random walk simulations, rendering it rather inefficient on large graphs and can only return the visualization for *Amazon* in 845 seconds. The two multi-level methods, i.e., KDraw and OpenOrd, have comparable performance to PPRviz on *Amazon*, i.e., the smallest one among the 6 graphs, but turn to be markedly inferior to PPRviz when graph size increases. The reason is that KDraw requires computing the forces between the leaf nodes inside a supernode which is rather costly for high-level supergraphs, and OpenOrd takes many iterations to determine the layout in its five-stage design.

Preprocessing time. Fig. 13 plots 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 hierarchical clustering for the input graph. However, PPRviz also computes the DPR vector and some single-target PDist scores. The results show that the processing time of PPRviz is two to three orders of magnitude faster than OpenOrd and one order of magnitude faster than KDraw. Moreover, both OpenOrd and KDraw cannot finish preprocessing for the largest *Twitter* graph within 12 hours while PPRviz takes only 33 minutes. This is because OpenOrd computes the layout of the entire graph first and then conducts hierarchical clustering on the two-dimensional layout. For KDraw, its clustering algorithm [51] is more expensive than Louvain+ in PPRviz.

Vary cluster size. Table 7 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

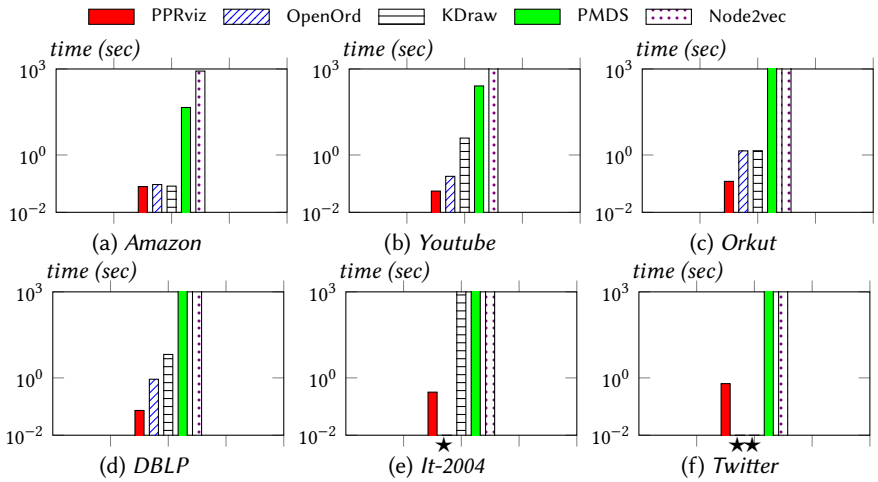


Fig. 12. Response time of selected methods (⋆ indicates failure to finish preprocessing within 12 hours).

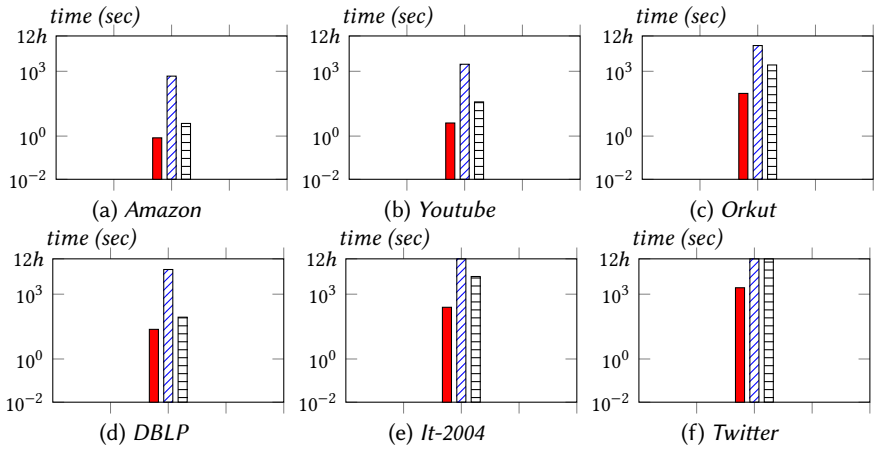


Fig. 13. Preprocessing time of the multi-level methods: PPRviz, OpenOrd and KDraw.

Table 7. 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

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-ℓ supergraph and fewer levels of the supergraph hierarchy. For interactive visualization, PPRviz's response time increases with k, and the main reason is that more pairwise PDist are computed with more nodes in each visualization. Furthermore, the response time of PPRviz is only 2.10 seconds even with $k = 100$, above which the intra-structure of a supernode becomes dense and overburdens human perception [35].

Table 8. Response time of PPRviz variants (in seconds).

	PI	FORA	FORA+	ResAcc	Tau-Push	GFRA	GFP (τ_{max})
Youtube	-	-	-	-	0.06	0.07	0.06
Orkut	-	-	-	-	0.12	0.36	0.12
It-2004	-	-	-	-	0.32	0.73	0.33
Twitter	-	-	-	-	0.63	2.76	0.66

Table 9. Preprocessing time of PPRviz variants (in seconds).

	PI	FORA	FORA+	ResAcc	Tau-Push	GFRA	GFP (τ_{max})
Youtube	2.46	3.35	3.17	2.46	4.04	5.1	3.99
Orkut	72.61	79.89	78.75	72.61	94.56	104.94	94.53
It-2004	169.27	223.99	200.21	169.27	312.33	308.3	69.18
Twitter	1296.17	1364.04	1360.08	1296.17	1984.73	1485.64	1914.92

Table 10. Index size of PPRviz variants (in MiB).

	PI	FORA	FORA+	ResAcc	Tau-Push	GFRA	GFP (τ_{max})
Youtube	5	51	30	5	9	51	9
Orkut	13	237	95	13	25	237	25
It-2004	172	1520	1022	172	330	1520	330
Twitter	177	1610	1052	177	338	1610	335

7.4 Tau-Push Performance

Comparing with other solutions. To validate the efficiency of the Tau-Push algorithm for PDist computation, we replace Tau-Push with 4 alternative solutions as mentioned in Section 7.1, and then compare PPRviz against these variants in terms of response time, preprocessing time, and index size on the 4 largest graphs including *Youtube*, *Orkut*, *It-2004*, and *Twitter*. In terms of response time, we use "-" to indicate that an approach fails to terminate within 1000 seconds. Table 8 shows that all PPRviz variants incur more than 1000 seconds on all tested graphs. This is because they need to compute PDist from $O(n)$ leaf nodes for the top-level supergraph. As shown in Table 9, all methods achieve comparable performance regarding the preprocessing time, since supergraph hierarchy construction dominates preprocessing cost. The two PPRviz variants using PI and ResAcc have slightly shorter preprocessing time as they only need to construct the supergraph hierarchies. For the same reason, they require much less space to store the indices as shown in Table 10. In contrast, FORA (resp. Tau-Push) precomputes random walk samples (resp. DPR values and results for GBP) as indices. Compared with FORA, our Tau-Push costs less space for indices because Tau-Push eliminates the need to store random walks by precomputing DPR to guide the termination of push operations. Specifically, besides $O(n)$ supernode partitions, PPRviz only requires storing n DPR values and $O(k \cdot \sqrt{n \cdot k})$ extra PDist values, which in total are insignificant compared with the size of the input graph.

Ablation study. In this set of experiments, we verify the effectiveness of the three techniques in Tau-Push, i.e., the grouped push strategy, DPR-guided termination trick, and the filter-refinement optimization delineated in Section 4.1. First, to demonstrate the effectiveness of the grouped push strategy, we replace Forward-Push in FORA with GFP and call this variant GFRA. In particular, for each level-ℓ supernode, GFRA first invokes GFP to roughly estimate level-ℓ DPPR values w.r.t. the given source supernode and then refines the under-estimations by random walk samplings. After conducting a theoretical analysis, we generate a sufficient amount of random walks to ensure the correctness and optimize the time complexity by balancing the overhead of two stages for a fair

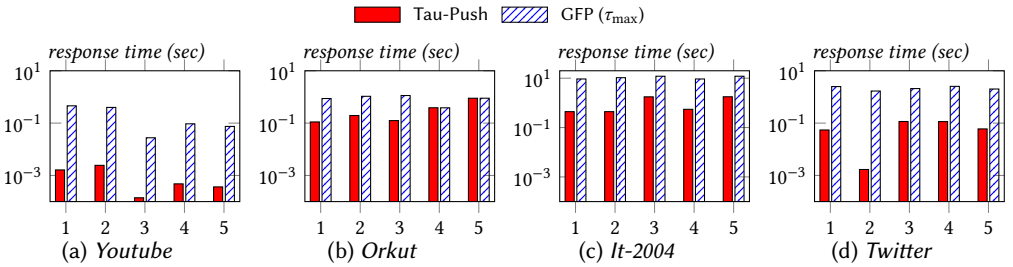


Fig. 14. Comparison of Tau-Push and GFP ($\tau_{\max}$)

comparison. Akin to the analysis in Table 2, we can show that the time complexity of GFRA to compute all pairwise approximate level-ℓ PDist in a supernode $\mathcal{S}$ is $O\left(\frac{k \cdot \log n \cdot \sqrt{m}}{\epsilon}\right)$, which improves the time complexity of FORA by $n/\sqrt{k}$. As shown in Table 8, the response time of GFRA is at least four orders of magnitude faster than FORA by adopting the grouped push strategy in our Tau-Push.

Note that if we set $\tau = \max_{\mathcal{V}_j \in \mathcal{S} \setminus \mathcal{V}_i} \tau_j$ for Eq.(5), all values returned by GFP are $(\epsilon, \delta)-$approximate level-ℓ DPPR. Here, we call this variant GFP (τ_{max}) and compare GFRA against it. We find in Table 8 that GFP (τ_{max}) improves GFRA on the *Orkut*, *It-2004*, and *Twitter* graphs that have massive edges, with 3×, 2.3×, and 4× speedups, respectively. Furthermore, as shown in Table 10, GFP (τ_{max}) incurs much less space overhead for index storage. The reason is that on such graphs, GFRA requires a multitude of excessive random walks, while GFP (τ_{max}) eliminates random walks and enables early termination by leveraging DPR as described in Section 4.1.

Finally, we evaluate the filter-refinement optimization by comparing Tau-Push with the initial solution GFP (τ_{max}). Fig. 14 shows that Tau-Push significantly reduces the response time of GFP when the cluster contains nodes with large DPR values by employing GBP. The speedup of Tau-Push varies since the skewness of DPR value is different on each graph. For the first cluster on *Youtube*, the largest DPR value is 3 orders of magnitude larger than the others, and thus Tau-Push speeds up GFP by about 300× by avoiding many rounds of push operations. For a similar reason, Tau-Push is over 1000× faster than GFP 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 CONCLUSIONS

This paper proposes a PPR-based node distance PDist and its fast computation algorithm Tau-Push for massive graph visualization, whose performance is extensively evaluated by comparing with 13 competitors on 12 real-world graphs. The results show that our proposal achieves high effectiveness and efficiency. Regarding future works, we plan to extend PDist and Tau-Push to support dynamic or attributed graph visualization.

ACKNOWLEDGMENTS

This work was partially supported by A*STAR, Singapore (Grant No. A19E3b0099), by Guangdong Basic and Applied Basic Research Foundation (Grant No. 2021A1515110067), and by Shenzhen Fundamental Research Program (Grant No. 20220815112848002) and a research gift from Huawei. Bo Tang is also affiliated with the Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology, Shenzhen, China.

REFERENCES

[1] James Abello, Frank Van Ham, and Neeraj Krishnan. 2006. Ask-graphview: A large scale graph visualization system. *TVCG* 12, 5 (2006), 669–676.

[2] Giuseppe Agapito, Pietro Hiram Guzzi, and Mario Cannataro. 2013. Visualization of protein interaction networks: problems and solutions. *BMC* 14, 1 (2013), 1–30.

[3] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. 2013. Distributed large-scale natural graph factorization. In *WWW*. 37–48.

[4] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *FOCS*. 475–486.

[5] Daniel Archambault, Tamara Munzner, and David Auber. 2008. GrouseFlocks: Steerable exploration of graph hierarchy space. *TVCG* 14, 4 (2008), 900–913.

[6] David Auber. 2004. Tulip—A huge graph visualization framework. In *GDS*. 105–126.

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

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

[9] Mikhail Belkin and Partha Niyogi. 2003. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.* 15, 6 (2003), 1373–1396.

[10] Chris Bennett, Jody Ryall, Leo Spalteholz, and Amy Gooch. 2007. The aesthetics of graph visualization. *CAe* (2007), 57–64.

[11] Sourav S Bhowmick, Kai Huang, Huey Eng Chua, Zifeng Yuan, Byron Choi, and Shuigeng Zhou. 2020. AURORA: Data-driven construction of visual graph query interfaces for graph databases. In *SIGMOD*. 2689–2692.

[12] Nikos Bikakis, John Liagouris, Maria Krommyda, George Papastefanatos, and Timos Sellis. 2016. GraphVizdb: A scalable platform for interactive large graph visualization. In *ICDE*. 1342–1345.

[13] 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, 10 (2008), P10008.

[14] Paolo Boldi and Sebastiano Vigna. 2004. The WebGraph Framework I: Compression Techniques. In *WWW*. 595–602.

[15] Ulrik Brandes and Christian Pich. 2006. Eigensolver methods for progressive multidimensional scaling of large data. In *GD*. 42–53.

[16] Fan Chung and Alexander Tsiatas. 2012. Finding and visualizing graph clusters using pagerank optimization. *Internet Math.* (2012), 86–97.

[17] Ron Davidson and David Harel. 1996. Drawing graphs nicely using simulated annealing. *TOG* 15, 4 (1996), 301–331.

[18] Wouter De Nooy, Andrej Mrvar, and Vladimir Batagelj. 2018. *Exploratory social network analysis with Pajek: Revised and expanded edition for updated software*. Vol. 46. Cambridge university press.

[19] Fan Du, Nan Cao, Yu-Ru Lin, Panpan Xu, and Hanghang Tong. 2017. isphere: Focus+ context sphere visualization for interactive large graph exploration. In *CHI*.

[20] Christian A Duncan, Michael T Goodrich, and Stephen G Kobourov. 1998. Balanced aspect ratio trees and their use for drawing very large graphs. In *GD*. 111–124.

[21] Peter Eades. 1984. A heuristic for graph drawing. *Congr. Numer.* 42 (1984), 149–160.

[22] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. 2005. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Math.* 2, 3 (2005), 333–358.

[23] Thomas MJ Fruchterman and Edward M Reingold. 1991. Graph drawing by force-directed placement. *SP&E* 21, 11 (1991), 1129–1164.

[24] Denis Gallo, Matteo Lissandrini, and Yannis Velegrakis. 2020. Personalized page rank on knowledge graphs: Particle Filtering is all you need!. In *EDBT*. 447–450.

[25] Emden R Gansner, Yifan Hu, and Stephen North. 2012. A maxent-stress model for graph layout. *TVCG* 19, 6 (2012), 927–940.

[26] Emden R Gansner, Yehuda Koren, and Stephen North. 2004. Graph drawing by stress majorization. In *GD*. 239–250.

[27] Helen Gibson, Joe Faith, and Paul Vickers. 2013. A survey of two-dimensional graph layout techniques for information visualisation. *Inf. Vis.* 12, 3-4 (2013), 324–357.

[28] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowl.-Based Syst.* 151 (2018), 78–94.

[29] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. 855–864.

[30] 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).

[31] Ivan Herman, Guy Melançon, and M Scott Marshall. 2000. Graph visualization and navigation in information visualization: A survey. *TVCG* 6, 1 (2000), 24–43.

[32] Guanhao Hou, Xingguang Chen, Sibo Wang, and Zhewei Wei. 2021. Massively parallel algorithms for personalized pagerank. *PVLDB* 14, 9 (2021), 1668–1680.
[33] Yifan Hu. 2005. Efficient, high-quality force-directed graph drawing. *Mathematica* 10, 1 (2005), 37–71.
[34] Yifan Hu and Lei Shi. 2015. Visualizing large graphs. *Wiley Interdiscip. Rev. Comput. Stat.* 7, 2 (2015), 115–136.
[35] Weidong Huang, Peter Eades, and Seok-Hee Hong. 2009. Measuring effectiveness of graph visualizations: A cognitive load perspective. *Inf. Vis.* 8, 3 (2009), 139–152.
[36] 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* 9, 6 (2014), e98679.
[37] Glen Jeh and Jennifer Widom. 2002. Simrank: a measure of structural-context similarity. In *SIGKDD*. 538–543.
[38] 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*. 789–804.
[39] Tomihisa Kamada, Satoru Kawai, et al. 1989. An algorithm for drawing general undirected graphs. *Inform. Process. Lett.* 31, 1 (1989), 7–15.
[40] Moritz Klammler, Tamara Mchedlidze, and Alexey Pak. 2018. Aesthetic discrimination of graph layouts. In *GD*. 169–184.
[41] Jérôme Kunegis. 2013. KONECT – The Koblenz Network Collection. In *WWW*.
[42] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a social network or a news media?. In *WWW*. 591–600.
[43] Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, and Nathalie Henry. 2006. Task taxonomy for graph visualization. In *BELIV*.
[44] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.
[45] Dandan Lin, Raymond Chi-Wing Wong, Min Xie, and Victor Junqiu Wei. 2020. Index-free approach with theoretical guarantee for efficient random walk with restart query. In *ICDE*. 913–924.
[46] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2016. Personalized pagerank estimation and search: A bidirectional approach. In *WSDM*. 163–172.
[47] Peter Lofgren and Ashish Goel. 2013. Personalized pagerank to a target node. *arXiv* (2013).
[48] Siqiang Luo, Xiaokui Xiao, Wenqing Lin, and Ben Kao. 2019. Baton: Batch one-hop personalized pageranks with efficiency and accuracy. *TKDE* 32, 10 (2019), 1897–1908.
[49] Shawn Martin, W Michael Brown, Richard Klavans, and Kevin W Boyack. 2011. OpenOrd: an open-source toolbox for large graph layout. In *VDA*, Vol. 7868. 786806.
[50] Henning Meyerhenke, Martin Nöllenburg, and Christian Schulz. 2017. Drawing large graphs by multilevel maxent-stress optimization. *TVCG* 24, 5 (2017), 1814–1827.
[51] Henning Meyerhenke, Peter Sanders, and Christian Schulz. 2014. Partitioning complex networks via size-constrained clustering. In *SEA*. 351–363.
[52] Mark EJ Newman. 2006. Modularity and community structure in networks. *PNAS* 103, 23 (2006), 8577–8582.
[53] Andreas Noack. 2005. Energy-based clustering of graphs with nonuniform degrees. In *GD*. 309–320.
[54] Andreas Noack. 2007. Unified quality measures for clusterings, layouts, and orderings of graphs, and their application as software design criteria. (2007).
[55] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The pagerank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.
[56] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. 701–710.
[57] Helen C Purchase. 2002. Metrics for graph drawing aesthetics. *JVLC* 13, 5 (2002), 501–516.
[58] Helen C Purchase, David Carrington, and Jo-Anne Allder. 2002. Empirical evaluation of aesthetics-based graph layout. *Empir. Softw. Eng.* 7, 3 (2002), 233–255.
[59] 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.
[60] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* 290, 5500 (2000), 2323–2326.
[61] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M Tamer Özsu. 2017. The ubiquity of large graphs and surprising challenges of graph processing. *PVLDB* 11, 4 (2017), 420–431.
[62] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. 2003. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research* 13, 11 (2003), 2498–2504.
[63] Jieming Shi, Renchi Yang, Tianyuan Jin, Xiaokui Xiao, and Yin Yang. 2019. Realtime top-k personalized pagerank over large graphs on gpus. *PVLDB* 13, 1 (2019), 15–28.

[64] 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.
[65] 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.
[66] Robert R Sokal. 1958. A statistical method for evaluating systematic relationships. *Univ. Kansas, Sci. Bull.* 38 (1958), 1409–1438.
[67] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.
[68] Martyn Taylor and Peter Rodgers. 2005. Applying graphical design techniques to graph visualisation. In *Inf. Vis.* 651–656.
[69] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. In *ICDM*. 613–622.
[70] Warren S Torgerson. 1952. Multidimensional scaling: I. Theory and method. *Psychometrika* 17, 4 (1952), 401–419.
[71] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *JMLR* 9 (2008), 2579–2605.
[72] 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*, Vol. 30. 1719–1749.
[73] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *SIGKDD*. 1225–1234.
[74] Hanzhi Wang, Zhewei Wei, Junhao Gan, Sibo Wang, and Zengfeng Huang. 2020. Personalized pagerank to a Target Node, Revisited. In *SIGKDD*. 657–667.
[75] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. HubPPR: effective indexing for approximate personalized pagerank. *PVLDB* 10, 3 (2016), 205–216.
[76] Sibo Wang, Renchi Yang, Runhui Wang, Xiaokui Xiao, Zhewei Wei, Wenqing Lin, Yin Yang, and Nan Tang. 2019. Efficient algorithms for approximate single-source personalized pagerank queries. *TODS* 44, 4 (2019), 1–37.
[77] 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.
[78] Yunhai Wang, Yanyan Wang, Yinqi Sun, Lifeng Zhu, Kecheng Lu, Chi-Wing Fu, Michael Sedlmair, Oliver Deussen, and Baoquan Chen. 2017. Revisiting stress majorization as a unified framework for interactive constrained graph visualization. *TVCG* (2017).
[79] 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*. 1996–2008.
[80] Yanhong Wu, Nan Cao, Daniel Archambault, Qiaomu Shen, Huamin Qu, and Weiwei Cui. 2016. Evaluation of graph sampling: A visualization perspective. *TVCG* (2016).
[81] Kai Xu, Rohan Williams, Seok-Hee Hong, Qing Liu, and Ji Zhang. 2009. Semi-bipartite graph visualization for gene ontology networks. In *GD*.
[82] 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.
[83] 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*. 1132–1143.
[84] Shiqi Zhang, Renchi Yang, Xiaokui Xiao, Xiao Yan, and Bo Tang. 2023. Effective and efficient pagerank-based positioning for graph visualization. *arXiv preprint arXiv:2112.14944* (2023).
[85] Di Zhuang, J Morris Chang, and Mingchen Li. 2019. DynaMo: Dynamic community detection by incrementally maximizing modularity. *TKDE* (2019).

Received July 2022; revised October 2022; accepted November 2022