# Efficient Algorithms for Computing Random Walk Centrality

Changan Liu, Zixuan Xie, Ahad N. Zehmakan, and Zhongzhi Zhang, *Member, IEEE*

***Abstract*—Random walk centrality is a fundamental metric in graph mining for quantifying node importance and influence, defined as the weighted average of hitting times to a node from all other nodes. Despite its ability to capture rich graph structural information and its wide range of applications, computing this measure for large networks remains impractical due to the computational demands of existing methods. In this paper, we present a novel formulation of random walk centrality, underpinning two scalable algorithms: one leveraging approximate Cholesky factorization and sparse inverse estimation, while the other sampling rooted spanning trees. Both algorithms operate in near-linear time and provide strong approximation guarantees. Extensive experiments on large real-world networks, including one with over 10 million nodes, demonstrate the efficiency and approximation quality of the proposed algorithms.**

***Index Terms*—Graph algorithm, Laplacian matrix, random walk centrality, hitting time, Cholesky factorization, spanning tree.**

## I. Introduction

CENTRALITY measures are fundamental concepts in network analysis, designed to quantify the importance and influence of nodes within a network [1]. These measures find widespread applications in diverse fields, ranging from influence maximization [2] to the development of effective vaccination strategies [3]. Among the most popular centrality measures are degree, PageRank [4], [5], closeness centrality [6], and eigenvector centrality [7].

In recent years, random walk centrality (RWC) has emerged as a particularly powerful and versatile measure [8]–[13]. Due to its ability to encode rich graph structural information at a global level, RWC has a better discriminating power [14] compared to many other centrality measures [12], [13]. This unique capability has led to RWC's adoption in a wide array of applications across various domains [11], [15]–[19].

While RWC is crucial and widely applicable, its computation remains a significant challenge, particularly for large graphs. Exact computation relies on the spectrum of the normalized Laplacian matrix, which requires $O(n^3)$ time for a graph with $n$ nodes, making it impractical for large values of $n$. This computational bottleneck has motivated extensive research into more efficient algorithms for computing RWC in large graphs.

In response to this issue, Zhang et al. [12] developed a novel randomized algorithm for approximating RWC. Their method reduces the estimation to approximating a quadratic form involving the pseudo-inverse of the Laplacian matrix, computed using a Laplacian solver [20]–[22] that runs in linear time with respect to the number of edges. However, this algorithm requires $O(\log n/\epsilon^2)$ invocations of the Laplacian solver for a parameter $\epsilon$ and the construction of a dense $n \times O(\log n/\epsilon^2)$ random matrix. These requirements make it computationally intensive, rendering the approach impractical for large-scale networks with millions of nodes.

In this paper, we present more efficient and effective algorithms for calculating RWC, addressing the limitations of existing methods. Our approach is based on a key observation that computing RWC is equivalent to calculating the diagonal elements of the pseudo-inverse of the normalized Laplacian matrix [9], [23]. While exact computation of these elements traditionally requires $O(n^3)$ time, we introduce a novel formulation to accelerate this process.

Our formulation expresses the diagonal elements of the normalized Laplacian pseudo-inverse in terms of two components, namely the diagonal elements of the inverse of a submatrix of the normalized Laplacian, obtained by deleting a specific row and column, and the corresponding column elements of the normalized Laplacian pseudo-inverse. This decomposition forms the foundation for our more efficient algorithmic approach.

Building upon the new formulation, we develop two fast approximation methods. The first algorithm leverages the positive definiteness of the normalized Laplacian submatrix, enabling the use of Cholesky factorization [24] for efficient computation of a solution. However, since calculating the exact Cholesky factorization and inversion of the factorization factors can be time-consuming, we employ incomplete Cholesky factorization and sparse Cholesky factor inversion. This approach exploits the fact that many elements in the inverse matrix are negligibly small and can be treated as zero, yielding a near-linear time algorithm with theoretical guarantees.

Our second algorithm capitalizes on the relationship between the diagonal elements of the inverse of normalized Laplacian submatrices and random walks with traps [25]. We propose a Monte Carlo simulation method based on ran-

This work was supported by the National Natural Science Foundation of China (No. 62372112 and No. 61872093). *(Corresponding author: Zhongzhi Zhang.)*

Changan Liu, Zixuan Xie, and Zhongzhi Zhang are with Shanghai Key Laboratory of Intelligent Information Processing, College of Computer Science and Artificial Intelligence, Fudan University, Shanghai 200433, China.
E-mail: 19110240031@fudan.edu.cn, 20302010061@fudan.edu.cn, zhangzz@fudan.edu.cn
Ahad N. Zehmakan is with the School of Computing, the Australian National University, Canberra, Australia.
E-mail: ahadn.zehmakan@anu.edu.au

Manuscript received xxxx; revised xxxx.

dom spanning tree generation [26]. Through novel theoretical analysis, we establish the required number of spanning tree samples and the corresponding approximation guarantee that can be achieved.

To validate our algorithms, extensive experiments on real-world networks ranging from several thousand to over 10 million nodes are conducted. The results demonstrate that our first algorithm achieves a significant efficiency improvement over the baseline, with only a slight compromise on approximation accuracy. Our second algorithm substantially outperforms the baseline in both efficiency and approximation quality. Specifically, both of our algorithms can effectively handle networks with over 3 million nodes, whereas the baseline fails to run due to memory overflow. Furthermore, for networks with over 10 million nodes, our first algorithm delivers approximate results with a mean relative error of only 3% within approximately 4000 seconds. Meanwhile, our second algorithm provides an even more accurate approximation with a mean relative error of just 0.5%, requiring slightly more computation time than the first algorithm but still significantly less than the baseline.

Our main contributions can be summarized as follows:

- We introduce a novel formulation for calculating RWC that constitutes the central theoretical contribution of this work.
- This new formulation, for the first time, shifts the main effort of estimating RWC to lightweight routines that estimate the diagonal of a normalized-Laplacian inverse, inspiring two algorithmic frames: one leveraging the sparse inverse of the normalized Laplacian submatrix's incomplete Cholesky factor, and another exploiting its connection to random walks with traps on graphs.
- Through extensive experiments on various real-world and synthetic networks, we demonstrate that our proposed algorithms outperform existing methods in speed by several orders of magnitude, without compromising much on approximation quality.

**Roadmap.** In Section II, we form the ground for our study by providing the necessary definitions. In Section III, a new formulation of RWC is established. Our two algorithms based on matrix factorization and sampling rooted spanning trees are given in Sections IV and V. Section 6 presents our experimental results and their analysis.

## II. Preliminaries

This section sets up the stage for our study by introducing basic notations, concepts about graphs, and the existing algorithms.

### A. Notations

Throughout this paper, we use the following notation unless otherwise specified. Lowercase letters (e.g., $a$) represent scalars, bold lowercase letters (e.g., $\boldsymbol{x}$) denote vectors, and uppercase letters (e.g., $\boldsymbol{M}$) indicate matrices. For indexing elements, we use subscripts. Specifically, $\boldsymbol{x}_i$ represents the $i$-th element of vector $\boldsymbol{x}$, while $\boldsymbol{M}_{ij}$ denotes the element at the $i$-th row and $j$-th column of matrix $\boldsymbol{M}$. Additionally, $\boldsymbol{M}_{i,:}$ and $\boldsymbol{M}_{:,j}$ represent the $i$-th row and $j$-th column of $\boldsymbol{M}$, respectively. We use $\boldsymbol{e}_i$ for the $i$-th standard basis vector and $\boldsymbol{x}^\top$ for the transpose of vector $\boldsymbol{x}$. The symbol $\mathbf{0}$ denotes the all-zeros vector, whereas $\boldsymbol{J}$ represents the all-ones matrix. Finally, $\boldsymbol{M}_i$ stands for the submatrix of $\boldsymbol{M}$ obtained by removing its $i$-th row and column. For any vector $\boldsymbol{x}$, $\boldsymbol{x}_{-i}$ denotes its subvector obtained by removing the $i$-th element.

### B. Graph and Corresponding Matrices

Consider a connected undirected graph (network) $\mathcal{G} = (V, E)$ with nodes $V$ and edges $E \subseteq V \times V$. Let $n := |V|$ and $m := |E|$ denote the number of nodes and the number of edges, respectively. A rooted spanning tree is a connected subgraph of $\mathcal{G}$ which has $n$ nodes and $n-1$ edges with one node designated as the root. We use $\mathcal{N}(i)$ to denote the set of neighbors of $i$, and the degree of node $i$ is $\boldsymbol{d}_i = |\mathcal{N}(i)|$. The Laplacian matrix of $\mathcal{G}$ is the symmetric matrix $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$, where $\boldsymbol{A} \in \{0,1\}^{n\times n}$ is the adjacency matrix whose entry $\boldsymbol{A}_{ij} = 1$ if node $i$ and node $j$ are adjacent, and $\boldsymbol{A}_{ij} = 0$ otherwise. The matrix $\boldsymbol{D} = \mathrm{diag}(\boldsymbol{d}_1, \cdots, \boldsymbol{d}_n)$ represents the diagonal degree matrix and $d_{\max}$ denotes the maximum degree of nodes.

For any pair of distinct nodes $u, v \in V$, we define $\boldsymbol{b}_{uv} = \boldsymbol{e}_u - \boldsymbol{e}_v$. We fix an arbitrary orientation for all edges in $\mathcal{G}$, then we can define the signed edge-node incidence matrix $\boldsymbol{B}^{m\times n}$ of graph $\mathcal{G}$, whose entries are defined as follows: $\boldsymbol{B}_{eu} = 1$ if node $u$ is the head of edge $e$, $\boldsymbol{B}_{eu} = -1$ if $u$ is the tail of $e$, and $\boldsymbol{B}_{eu} = 0$ otherwise. $\boldsymbol{L}$ can be rewritten as $\boldsymbol{L} = \boldsymbol{B}^\top \boldsymbol{B}$ and it is positive semi-definite with its Moore-Penrose pseudo-inverse being $\boldsymbol{L}^\dagger = \left(\boldsymbol{L} + \frac{1}{n}\boldsymbol{J}\right)^{-1} - \frac{1}{n}\boldsymbol{J}$. Finally, we use $\mathcal{L} = \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2}$ to denote the normalized Laplacian matrix.

### C. Random Walk Centrality

In a random walk on a graph $\mathcal{G}$, at any discrete-time step, the walker moves from its current node $i$ to node $j$ independently with probability $\boldsymbol{A}_{ij}/\boldsymbol{d}_i$ (i.e., it moves to one of the neighboring nodes with a uniform probability). This process can be characterized by the transition matrix $\boldsymbol{P}$, where the element $\boldsymbol{P}_{ij}$ is equal to $\boldsymbol{A}_{ij}/\boldsymbol{d}_i$. If $\mathcal{G}$ is finite and non-bipartite, the random walk has the following unique stationary distribution (cf. [27]):

$$\boldsymbol{\pi} = (\pi_1, \pi_2, \cdots, \pi_n)^\top = \left(\frac{\boldsymbol{d}_1}{2m}, \frac{\boldsymbol{d}_2}{2m}, \cdots, \frac{\boldsymbol{d}_n}{2m}\right)^\top. \quad (1)$$

A fundamental quantity for random walks is *hitting time* [9], [28]. The hitting time $\mathcal{H}_{ij}$ represents the expected number of steps for a walker starting at node $i$ to reach node $j$ for the first time.

The *random walk centrality* [10] $\mathcal{H}_u$ of a node $u$, defined as $\mathcal{H}_u = \sum_i \rho(i)\mathcal{H}_{iu}$, where $\rho(\cdot)$ is the starting probability distribution over all nodes in $V$. By definition, $\mathcal{H}_u$ is a weighted average of hitting times to node $u$. Nodes with smaller values of $\mathcal{H}_u$ are considered more central. Unlike shortest-path based centrality measures, RWC accounts for contributions from all paths [29], offering a richer description of a node's importance.

In our study, we consider $\rho(\cdot)$ as the stationary distribution $\boldsymbol{\pi}$, simplifying $\mathcal{H}_u$ to $\sum_i \pi_i \mathcal{H}_{iu}$. This formulation of $\mathcal{H}_u$,

which has been extensively studied [23], [30], [31], measures the expected number of steps to reach node $u$ when starting from a random node following the stationary distribution. According to [9], [23], we have

$$\mathcal{H}_u = \sum_{i=1}^{n} \pi_i \mathcal{H}_{iu} = \frac{(\mathcal{L}^\dagger)_{uu}}{\pi_u}, \tag{2}$$

where $\mathcal{L}^\dagger$ is the pseudo-inverse of $\mathcal{L}$. Thus, $\mathcal{H}_u$ can be computed using only the diagonal of $\mathcal{L}^\dagger$.

### D. Existing Methods

Direct computation of RWC involves inverting the normalized Laplacian matrix (see Eq. (2)) which runs in $O(n^3)$ time. Thus, prior work has resorted to approximate solutions. The state-of-the-art algorithm is proposed by [12], where the authors established a connection between RWC and the quadratic form of the Moore-Penrose pseudo-inverse of the Laplacian matrix $\boldsymbol{L}^\dagger$. They effectively leveraged the Johnson-Lindenstrauss lemma and Laplacian solvers to compute this centrality measure. Their algorithm runs in $\tilde{O}(m/\epsilon^2)$ time where $\tilde{O}(\cdot)$ hides poly$(\log n)$ factors, marking a significant advancement over the previous algorithms.

However, this approach faces several critical challenges when applied to large-scale networks. Primarily, the memory requirements of the Laplacian solver are substantially high. Furthermore, the construction of an $n \times O(\log n/\epsilon^2)$ random matrix, a prerequisite for their method, imposes additional memory constraints that can be prohibitive in large-scale networks. More importantly, the algorithm requires $O(\log n/\epsilon^2)$ executions of the Laplacian solver. These repeated calls to the solver introduce significant computational overhead, especially as the network size grows. These limitations highlight the need for more scalable methods that preserve accuracy while reducing computational complexity, especially for the massive networks common in modern applications.

## III. New formulation of RWC

The primary computational challenge in the algorithm of Zhang et al. [12] stems from its repeated calls to the Laplacian solver. This prompts a natural question: can we reduce the number of solver calls? To this end, we next present a novel formulation for RWC.

**Theorem III.1.** *Let $v \in V$ be a designated pivot node, $\mathcal{L}_v$ be the submatrix of the normalized Laplacian matrix $\mathcal{L}$ obtained by deleting the $v$-th row and the $v$-th column of $\mathcal{L}$, then we have the following reformulation of RWC for all nodes $u \in V$:*

$$\mathcal{H}_u = \frac{1}{\pi_u}\Big((\mathcal{L}_v^{-1})_{uu} - \frac{\boldsymbol{d}_u}{\boldsymbol{d}_v}(\mathcal{L}^\dagger)_{vv} + 2\frac{\sqrt{\boldsymbol{d}_u}}{\sqrt{\boldsymbol{d}_v}}(\mathcal{L}^\dagger)_{uv}\Big). \tag{3}$$

**Proof.** For the pivot node $v$, setting $(\mathcal{L}_v^{-1})_{vv} = 0$ reduces Eq. (3) to Eq. (2), which holds trivially.

For other nodes, we envision the network as an electrical circuit where every edge in $E$ acts as a unit resistor, and the nodes in $V$ function as junctions connecting these resistors. When a unit current is injected into node $u$ and extracted from node $v$, the resulting current $\boldsymbol{i}$ on the edges adheres to Kirchhoff's current law [32], which asserts that the total current flowing into node $v$ must equal the current injected into it: $\boldsymbol{b}_{uv}^\top = \boldsymbol{i}^\top \boldsymbol{B}$.

For any potential vector $\boldsymbol{f}$, according to Ohm's law [32], the induced current on the edge from $u$ to $v$ is given by the product of $\boldsymbol{f}_u - \boldsymbol{f}_v$ and the edge conductance: $\boldsymbol{i}^\top = \boldsymbol{f}^\top \boldsymbol{B}^\top$. Therefore, we have

$$\boldsymbol{b}_{uv}^\top \boldsymbol{D}^{-1/2} = \boldsymbol{f}^\top \boldsymbol{B}^\top \boldsymbol{B}\boldsymbol{D}^{-1/2} = \boldsymbol{f}\boldsymbol{D}^{1/2}\mathcal{L}.$$

Assume $\boldsymbol{f}$ satisfies $\sum_v \boldsymbol{f}_v = 0$. By Green's function [33], [34],

$$\boldsymbol{b}_{uv}^\top \boldsymbol{D}^{-1/2}\mathcal{L}^\dagger \boldsymbol{D}^{-1/2} = \boldsymbol{f}^\top.$$

The effective resistance [35] between $u$ and $v$ is thus

$$R_{uv} = \boldsymbol{f}^\top \boldsymbol{b}_{uv} = \boldsymbol{b}_{uv}^\top \boldsymbol{D}^{-1/2}\mathcal{L}^\dagger \boldsymbol{D}^{-1/2}\boldsymbol{b}_{uv}.$$

Therefore, we have

$$\begin{aligned}&(\boldsymbol{D}^{-1/2}\mathcal{L}^\dagger \boldsymbol{D}^{-1/2})_{uu}\\ =&R_{uv} - (\boldsymbol{D}^{-1/2}\mathcal{L}^\dagger \boldsymbol{D}^{-1/2})_{vv} + 2(\boldsymbol{D}^{-1/2}\mathcal{L}^\dagger \boldsymbol{D}^{-1/2})_{uv}.\end{aligned}$$

By multiplying both sides by $\boldsymbol{d}_u$, we get

$$(\mathcal{L}^\dagger)_{uu} = \boldsymbol{d}_u R_{uv} - \frac{\boldsymbol{d}_u}{\boldsymbol{d}_v}(\mathcal{L}^\dagger)_{vv} + 2\frac{\sqrt{\boldsymbol{d}_u}}{\sqrt{\boldsymbol{d}_v}}(\mathcal{L}^\dagger)_{uv}. \tag{4}$$

According to [36], the resistance distance between nodes $u$ and $v$ is $(\boldsymbol{L}_v^{-1})_{uu}$. Considering that

$$(\boldsymbol{L}_v^{-1})_{uu} = ((\boldsymbol{I} - \boldsymbol{P}_v)^{-1}\boldsymbol{D}_v^{-1})_{uu} = \frac{(\boldsymbol{I} - \boldsymbol{P}_v)^{-1}_{uu}}{\boldsymbol{d}_u}, \quad \text{and}$$

$$(\mathcal{L}_v^{-1})_{uu} = (\boldsymbol{D}_v^{-1/2}(\boldsymbol{I} - \boldsymbol{P}_v)^{-1}\boldsymbol{D}_v^{1/2})_{uu} = (\boldsymbol{I} - \boldsymbol{P}_v)^{-1}_{uu},$$

thus $\boldsymbol{d}_u R_{uv} = (\mathcal{L}_v^{-1})_{uu}$. Plugging it into Eq. (4) ends the proof. □

**Remark III.2** (RWC versus effective resistance [35])**.** *The quantity $\mathcal{H}_u$ in Eq.* (3) *is not equal to the classical effective resistance. The difference is two-fold. First, RWC is built on the normalised Laplacian $\mathcal{L}$, whereas effective resistance is defined with the Laplacian $\boldsymbol{L}$. Second, effective resistance is inherently a pairwise notion, depending on a chosen node pair while $\mathcal{H}_u$ serves as a global centrality that describes the position of node $u$ within the whole network, irrespective of which pivot node $v$ is chosen. Thus, the two quantities serve complementary analytic roles.*

Building on Theorem III.1, we propose a novel paradigm for computing RWC.

**New computation paradigm for RWC**

1. **Pivot solve.** Choose a pivot $v \in V$ and solve the linear system $\mathcal{L}\boldsymbol{y} = \boldsymbol{e}_v$ once. The solution $\boldsymbol{y} = \mathcal{L}^\dagger \boldsymbol{e}_v$ is exactly the $v$-th column of $\mathcal{L}^\dagger$.
2. **Diagonal estimation.** Approximate the remaining $n-1$ diagonal entries of $\mathcal{L}_v^{-1}$ without further global solves.
3. **Centrality recovery.** Compute the random-walk centrality $\mathcal{H}_u$ for every $u \in V$ via Eq. (3).

For the first step, which involves solving the linear system to obtain $\mathcal{L}^{\dagger}_{:,v}$, an exact solution is time-consuming as it involves matrix inversion. To accelerate the computation, we turn to the following linear Laplacian solver.

**Lemma III.3.** *(Fast SDDM Solver [20]–[22]) There is a nearly linear-time solver* $\mathbf{g} = \texttt{LapSolve}(\boldsymbol{L}, \boldsymbol{b}, \theta)$ *which takes a symmetric positive semi-definite matrix* $\boldsymbol{L}$ *with* $m$ *nonzero entries, a vector* $\boldsymbol{b} \in \mathbb{R}^n$*, and an error parameter* $\theta > 0$*, and returns a vector* $\mathbf{g} \in \mathbb{R}^n$ *satisfying* $\left\|\mathbf{g} - \boldsymbol{L}^{-1}\boldsymbol{b}\right\|_{\boldsymbol{L}} \leq \theta \left\|\boldsymbol{L}^{-1}\boldsymbol{b}\right\|_{\boldsymbol{L}}$ *with probability at least* $1 - 1/n$*, where* $\|\mathbf{g}\|_{\boldsymbol{L}} = \sqrt{\mathbf{g}^{\top}\boldsymbol{L}\mathbf{g}}$*. This solver runs in expected time* $\tilde{O}(m)$*, where* $\tilde{O}(\cdot)$ *notation suppresses the* $\mathrm{poly}(\log n)$ *factors.*

However, the off-the-shelf solver accepts only the unnormalised Laplacian $\boldsymbol{L}$, so it cannot be invoked on the normalised form $\mathcal{L}$ directly. Adopting the strategy of [12], we bridge this gap by exploiting the identity below, which relates the Moore–Penrose pseudoinverses of $\boldsymbol{L}$ and $\mathcal{L}$ [37]:

$$\mathcal{L}^{\dagger} = \left(I - \frac{1}{2m}\boldsymbol{D}^{\frac{1}{2}}\mathbf{1}\mathbf{1}^{\top}\boldsymbol{D}^{\frac{1}{2}}\right)\boldsymbol{D}^{\frac{1}{2}}\boldsymbol{L}^{\dagger}\boldsymbol{D}^{\frac{1}{2}}\left(I - \frac{1}{2m}\boldsymbol{D}^{\frac{1}{2}}\mathbf{1}\mathbf{1}^{\top}\boldsymbol{D}^{\frac{1}{2}}\right). \tag{5}$$

Therefore, we can obtain an approximation $\tilde{\boldsymbol{y}}$ for $\mathcal{L}^{\dagger}_{:,v}$ by calling $\texttt{LapSolve}(\boldsymbol{L}, \boldsymbol{b}, \theta)$ where $\boldsymbol{b} = \boldsymbol{D}^{1/2}\left(I - \frac{1}{2m}\boldsymbol{D}^{\frac{1}{2}}\mathbf{1}\mathbf{1}^{\top}\boldsymbol{D}^{\frac{1}{2}}\right)\boldsymbol{e}_v$ and then performing a simple matrix-vector multiplication. Assuming we have the (approximate) diagonal elements of $\mathcal{L}_v^{-1}$, we can efficiently calculate the RWCs according to Eq. (3). The complete procedure for estimating RWC $\tilde{\mathcal{H}}_u$ is presented in Algorithm 1.

**Algorithm 1:** APPRWC$(\mathcal{G}, v, \theta, \{(\mathcal{L}_v^{-1})_{uu} | u \in V \setminus \{v\}\})$

**Input :** A connected graph $\mathcal{G} = (V, E)$ with $n$ nodes;
pivot node $v$; error parameter $\theta$;
$(\mathcal{L}_v^{-1})_{uu}$ for all $u \in V \setminus \{v\}$

**Output :** Approximated RWC $\tilde{\mathcal{H}}_u$ for all $u \in V$

$\boldsymbol{b} = \boldsymbol{D}^{1/2}\left(I - \frac{1}{2m}\boldsymbol{D}^{\frac{1}{2}}\mathbf{1}\mathbf{1}^{\top}\boldsymbol{D}^{\frac{1}{2}}\right)\boldsymbol{e}_v$

$\boldsymbol{z} = \texttt{LapSolve}(\boldsymbol{L}, \boldsymbol{b}, \theta)$

$\tilde{\boldsymbol{y}} = \left(I - \frac{1}{2m}\boldsymbol{D}^{\frac{1}{2}}\mathbf{1}\mathbf{1}^{\top}\boldsymbol{D}^{\frac{1}{2}}\right)\boldsymbol{D}^{\frac{1}{2}}\boldsymbol{z}$

**for** $u \in V \setminus \{v\}$ **do**

  $\tilde{\mathcal{H}}_u = \frac{1}{\pi_u}\left((\mathcal{L}_v^{-1})_{uu} - \frac{\boldsymbol{d}_u}{\boldsymbol{d}_v}\tilde{\boldsymbol{y}}_v + 2\frac{\sqrt{\boldsymbol{d}_u}}{\sqrt{\boldsymbol{d}_v}}\tilde{\boldsymbol{y}}_u\right)$

$\tilde{\mathcal{H}}_v = \frac{1}{\pi_v}\tilde{\boldsymbol{y}}_v$

**return** $\tilde{\mathcal{H}}_u$ for all $u \in V$

To summarize this section, Theorem III.1 inspires a novel computation paradigm of the computation of RWC including three lightweight steps: a *single* Laplacian solve for a pivot column of $\mathcal{L}^{\dagger}$, followed by the estimation of $n-1$ diagonal entries of $\mathcal{L}_v^{-1}$ and a final inexpensive aggregation. In contrast, the best existing method [12] invokes a Laplacian solver $O\big(\log n/\epsilon^2\big)$ times, so both running time and memory scale with that factor. By collapsing the sequence of linear solves to a *single* solve for the pivot column, the new formulation eliminates the dominant cost term and *shifts* all remaining effort to lightweight routines that estimate the diagonal of $\mathcal{L}_v^{-1}$. Two complementary instantiations of this idea follow: Section IV exploits the symmetric positive-definiteness of $\mathcal{L}_v$ to build a sparse incomplete-Cholesky surrogate, while Section V achieves the same goal through a rooted spanning-tree sampler. Together, they compute the paradigm in near-linear time and significantly cut the overall memory and runtime.

## IV. FastChol Algorithm: Matrix Factorization

Building on the reformulation in Section III, which reduces RWC to accessing diagonal entries of $\mathcal{L}_v^{-1}$, and taking advantage of the symmetric positive definiteness of $\mathcal{L}_v$ established in [38], we next approximate those entries by replacing the exact inverse with a sparse substitute obtained from an incomplete-Cholesky factor.

### A. Theoretical Foundation

To overcome the computational complexity of direct matrix inversion, we propose a two-step approach that significantly reduces time complexity, consisting of incomplete Cholesky factorization and sparse approximate inverse.

The Cholesky factorization, widely used for its efficiency in handling symmetric positive definite matrices [24], [39]–[41], produces a lower triangular matrix $\boldsymbol{Z}$ such that $\boldsymbol{A} = \boldsymbol{Z}\boldsymbol{Z}^{\top}$. This factorization is advantageous as obtaining $\boldsymbol{Z}^{-1}$ can significantly reduce the computational cost compared to directly inverting $\boldsymbol{A}$.

Given the computational challenges associated with full matrix factorization, particularly for large-scale systems, incomplete Cholesky factorization is widely used instead [42], [43]. This approach approximates the full factorization while maintaining sparsity, which proceeds similarly to the full Cholesky factorization but drops fill-in elements below a certain threshold, often guided by the original matrix's sparsity pattern [44]. As $\mathcal{L}_v$ is symmetric positive definite, we can apply this technique to efficiently approximate $\mathcal{L}_v^{-1}$. The following lemma formalizes the properties of this incomplete factorization.

**Lemma IV.1.** *Let* $\mathcal{G} = (V, E)$ *be a connected graph and let* $\mathcal{L}_v$ *be its corresponding normalized Laplacian submatrix. Given parameter* $\delta > 0$ *(drop threshold), there exists an algorithm* ICHOL$(\mathcal{L}_v, \delta)$ *that runs in* $O(m + nnz(\boldsymbol{R}))$ *time* ($nnz(\boldsymbol{R})$ *is the number of non-zero elements in* $\boldsymbol{R}$*) and returns a lower triangular matrix* $\boldsymbol{R} \in \mathbb{R}^{(n-1)\times(n-1)}$*. The relative error in Frobenius norm is bounded by*

$$\frac{\|\mathcal{L}_v - \boldsymbol{R}\boldsymbol{R}^{\top}\|_F}{\|\mathcal{L}_v\|_F} = O(\sqrt{k}\cdot\delta), \tag{6}$$

*where* $k$ *is the number of elements dropped during factorization, and for a square matrix* $\boldsymbol{M}$*,* $\|\boldsymbol{M}\|_F = \sum_i \sum_j \boldsymbol{M}_{ij}^2$*.*

**Proof.** Let $\boldsymbol{E} = \mathcal{L}_v - \boldsymbol{R}\boldsymbol{R}^\top$ be the error matrix. For the incomplete Cholesky factorization, off-diagonal elements $(\mathcal{L}_v)_{ij}$ are dropped if

$$|(\mathcal{L}_v)_{ij}| < \delta \cdot \sqrt{|(\mathcal{L}_v)_{ii}| \cdot |(\mathcal{L}_v)_{jj}|}.$$

Therefore, each dropped element contributes at most $\delta^2 \cdot |(\mathcal{L}_v)_{ii}| \cdot |(\mathcal{L}_v)_{jj}|$ to $\|E\|_F^2$.

Since $\mathcal{L}_v$ is derived from a normalized Laplacian matrix, we know that its diagonal elements are equal to 1. The total number of dropped elements is $k$, so we have

$$\|\boldsymbol{E}\|_F^2 \leq k \cdot \delta^2 \cdot \max_{ij}(|(\mathcal{L}_v)_{ii}| \cdot |(\mathcal{L}_v)_{jj}|) \leq k \cdot \delta^2.$$

Taking the square root and dividing by $\|\mathcal{L}_v\|_F$, we have

$$\frac{\|\boldsymbol{E}\|_F}{\|\mathcal{L}_v\|_F} \leq \frac{\sqrt{k} \cdot \delta}{\|\mathcal{L}_v\|_F} = O(\sqrt{k} \cdot \delta). \tag{7}$$

The last equality holds because $\|\mathcal{L}_v\|_F \geq 1$ for a normalized Laplacian matrix with at least one non-zero off-diagonal element.

The time complexity $O(m+nnz(\boldsymbol{R}))$ follows from the fact that the algorithm needs to process all $m$ edges of the original graph and perform operations on the non-zero elements of $\boldsymbol{R}$. □

With the incomplete Cholesky factorization, we can now approximate $\mathcal{L}_v$ as $\boldsymbol{R}\boldsymbol{R}^\top$. This allows us to reformulate computing the diagonal entries of $\mathcal{L}_v^{-1}$ in terms of $\boldsymbol{R}$:

$$(\mathcal{L}_v^{-1})_{uu} \approx \boldsymbol{e}_u^\top (\boldsymbol{R}^{-1})^\top \boldsymbol{R}^{-1} \boldsymbol{e}_u = \|\boldsymbol{R}^{-1}\boldsymbol{e}_u\|_2^2. \tag{8}$$

However, computing $\boldsymbol{R}^{-1}$ is costly. This brings us to the second step of our strategy: devising an efficient method to approximate $\boldsymbol{R}^{-1}\boldsymbol{e}_u$ for all $u \in V \setminus \{v\}$ without explicitly computing $\boldsymbol{R}^{-1}$.

### *B. Algorithm Design*

The core idea is to construct a sparse matrix $\widetilde{\boldsymbol{S}}$ that approximates $\boldsymbol{R}^{-1}$. We now turn our attention to the core of our sparse approximation strategy. The efficiency of our approach hinges on a crucial property of the inverse of the incomplete Cholesky factor, which is based on the key observation given in the following lemma.

**Lemma IV.2.** *Let $\boldsymbol{R}$ be the incomplete Cholesky factor of $\mathcal{L}_v$, and let $\boldsymbol{S} = \boldsymbol{R}^{-1}$, $\boldsymbol{S}$ is then non-negative. For any $u$-th column of $\boldsymbol{S}$,*

$$\boldsymbol{S}_{:,u} = \frac{1}{\boldsymbol{R}_{uu}}\boldsymbol{e}_u - \frac{1}{\boldsymbol{R}_{uu}} \sum_{i=u+1}^{n} \boldsymbol{R}_{iu}\boldsymbol{S}_{:,i}. \tag{9}$$

**Proof. Verification of Eq.** (9)**:** Multiply both sides of the equation by $\boldsymbol{R}$ and for the right-hand side,

$$\frac{1}{\boldsymbol{R}_{uu}}\boldsymbol{R}\boldsymbol{e}_u - \frac{1}{\boldsymbol{R}_{uu}} \sum_{i=u+1}^{n} \boldsymbol{R}_{iu}\boldsymbol{R}\boldsymbol{S}_{:,i}$$

$$= \frac{1}{\boldsymbol{R}_{uu}}((0,\ldots,0,\boldsymbol{R}_{uu},\boldsymbol{R}_{u+1,u},\ldots,\boldsymbol{R}_{nu})^\top - \sum_{i=u+1}^{n} \boldsymbol{R}_{iu}\boldsymbol{e}_i)$$

$$= \boldsymbol{e}_u + \boldsymbol{v} - \frac{1}{\boldsymbol{R}_{uu}} \sum_{i=u+1}^{n} \boldsymbol{R}_{iu}\boldsymbol{e}_i = \boldsymbol{e}_u + \boldsymbol{v} - \boldsymbol{v} = \boldsymbol{e}_u,$$

where $\boldsymbol{v} = (0,\ldots,0,0,\frac{\boldsymbol{R}_{u+1,u}}{\boldsymbol{R}_{uu}},\ldots,\frac{\boldsymbol{R}_{nu}}{\boldsymbol{R}_{uu}})^\top$.

For the left-hand side, with the definition of $\boldsymbol{S}$, we know $\boldsymbol{R}\boldsymbol{S}_{:,u} = \boldsymbol{e}_u$, verifying Eq. (9).

**Non-negativity of $\boldsymbol{S}$:** We use induction on $u$, starting from $n$ and moving backwards.

Base case ($u = n$): $\boldsymbol{S}_{:,n} = \frac{1}{\boldsymbol{R}_{nn}}\boldsymbol{e}_n \geq 0$, as $\boldsymbol{R}_{nn} > 0$.

Inductive step: Assume $\boldsymbol{S}_{:,i} \geq 0$ for all $i > u$. From Eq. (9), we have

$$\boldsymbol{S}_{:,u} = \frac{1}{\boldsymbol{R}_{uu}}\boldsymbol{e}_u - \frac{1}{\boldsymbol{R}_{uu}} \sum_{i=u+1}^{n} \boldsymbol{R}_{iu}\boldsymbol{S}_{:,i}.$$

Since $\boldsymbol{R}_{uu} > 0$, $\boldsymbol{R}_{iu} \leq 0$ for $i > u$, and $\boldsymbol{S}_{:,i}$ is non-negative for $i > u$ by induction hypothesis, we have that $\boldsymbol{S}_{:,u}$ is non-negative for all $u$ and thus $\boldsymbol{S}$ is non-negative. □

Based on Lemma 4.3, a right-to-left recursive approach is suggested for computing the columns of $\boldsymbol{S}$. That is, to compute the $u$-th column of $\boldsymbol{S}$, we require information from all subsequent columns $i$ where $\boldsymbol{R}_{iu} \neq 0$. Nevertheless, this approach incurs significant computational cost, with a time complexity of $O(n^2)$ owing to the dense nature of $\boldsymbol{S}$. In our context, we propose two strategies to reduce this complexity.

**Algorithm 2:** FASTCHOL($\mathcal{G}, v, \epsilon_p, \delta, w_s, \zeta, \theta$)

**Input:** A connected graph $\mathcal{G} = (V, E)$; threshold $\epsilon_p$; pivot node $v$; parameters $\delta$ for ICHOL; base window size $w_s$; error parameter $\theta$

**Output:** $\tilde{\mathcal{H}}_u \ \forall u \in V$

$\boldsymbol{R} \leftarrow$ ICHOL$(\mathcal{L}_v, \delta)$
Set $\widetilde{\boldsymbol{S}}$ as an $(n-1) \times (n-1)$ zero matrix and $\tilde{\boldsymbol{\tau}} = \boldsymbol{0}^{(n-1)\times 1}$
$end \leftarrow n-1$
**for** $u = n-1$ **to** $1$ **do**
  $w_s \leftarrow \min(w_s \cdot (1 + \boldsymbol{d}_u/d_{\max}), n)$
  **if** $end - (u+1) > w_s$ **then** $end \leftarrow u + w_s$
  Initialize $\boldsymbol{S}^*_{:,u} \leftarrow \boldsymbol{0}^{(n-1)\times 1}$
  $\boldsymbol{S}^*_{uu} \leftarrow 1/\boldsymbol{R}_{uu}$
  **for** $i = u+1$ **to** $u + w_s$ **do**
    **if** $\boldsymbol{R}_{iu} \neq 0$ **then**
      $\boldsymbol{S}^*_{:,u} \leftarrow \boldsymbol{S}^*_{:,u} - \boldsymbol{R}_{iu}/\boldsymbol{R}_{uu} \cdot \widetilde{\boldsymbol{S}}_{:,i-1}$
  **if** $nnz(\boldsymbol{S}^*_{:,u}) \leq \zeta$ **then return** $\boldsymbol{S}^*_{:,u}$
  $\boldsymbol{S}^{**}_{:,u} \leftarrow$ Sort$(|\boldsymbol{S}^*_{:,u}|)$// elementwise absolution
  $k^* \leftarrow \arg\max_k \frac{\|\boldsymbol{S}^{**}_{k+1:end,u}\|_1}{\|\boldsymbol{S}^*_{:,u}\|_1} \leq \epsilon_p$
  Discard elements in $\boldsymbol{S}^*_{:,u}$ below $(\boldsymbol{S}^{**}_{:,u})_{k^*}$ to obtain $\widetilde{\boldsymbol{S}}_{:,u}$
  $\tilde{\tau}_u \leftarrow \|\widetilde{\boldsymbol{S}}_{:,u}\|_2^2$
$\tilde{\mathcal{H}}_u \leftarrow$ APPRWC$(\mathcal{G}, v, \theta, \tilde{\tau})$
**return** $\tilde{\mathcal{H}}_u \ \forall u \in V$

First, we observe that the elements in $\boldsymbol{S}$ are generally small. Thus, we can discard the insignificant elements in each column while focusing our attention on more significant ones, a process we refer to as sparsification. The sparsification process uses $\zeta$ to determine if discarding is necessary. If the number of non-zero elements in the initial estimate $\boldsymbol{S}^*_{:,u}$ exceeds $\zeta$, we

then use a threshold $\epsilon_p$ to decide which elements to discard. Specifically, only the $k$ largest elements with an approximation error not exceeding $\epsilon_p$ are retained. This approach ensures the sparsity upper bound $\zeta$ is respected while maintaining the most accurate sparse approximation within the error tolerance $\epsilon_p$.

Next, to further reduce complexity, we introduce a *sliding window* technique that limits the calculation of each column to only a few nearby columns on the right. This approach leverages the fact that these nearby columns already incorporate information from further right columns. Recognizing the scale-free nature of real-world networks, we adjust the window size $w_s$ based on node degree, scaling it with the ratio of the current node's degree to the maximum degree. This adaptive window allows higher-degree nodes to benefit from more information. Using this technique, we construct a final sparse approximation $\widetilde{\boldsymbol{S}}_{:,u}$ for each column.

Incorporating these ideas, we introduce FASTCHOL for estimating RWC, which is outlined in Algorithm 2. It begins with incomplete Cholesky factorization of $\mathcal{L}_v$ to obtain $\boldsymbol{R}$. The core of the algorithm is a loop that computes each column of $\widetilde{\boldsymbol{S}}$ from right to left, employing two key techniques for each column. First, the sliding window technique is implemented through adaptive window sizing (Lines 5-6), which limits calculations to nearby columns based on node degree. This is followed by the sparsification process (Lines 11-14), where the elements in $\boldsymbol{S}^*$ are sorted in descending order to facilitate efficient sparsification. After computing the sparse approximations of $\boldsymbol{S}_{:,u}$, we obtain the diagonal elements $\tilde{\tau}_u$ according to Eq. (8). Finally, the algorithm uses APPRWC to estimate RWCs.

### C. Performance Analysis

We now present a performance analysis of our algorithm FASTCHOL, which is summarized by the following theorem.

**Theorem IV.3.** *Given a connected graph $\mathcal{G} = (V, E)$, FASTCHOL runs in $O(n(w_s \log n + \log^2 n)) + \tilde{O}(m)$ time. For any node $u \in V$, let $\widetilde{\boldsymbol{S}}_{:,u}$ be the approximation of $\boldsymbol{S}_{:,u} = \boldsymbol{R}^{-1}\boldsymbol{e}_u$ (Line 14 in Algorithm 2), the relative error between them can be bounded by*

$$\frac{|\|\boldsymbol{S}_{:,u}\|_2^2 - \|\widetilde{\boldsymbol{S}}_{:,u}\|_2^2|}{\|\boldsymbol{S}_{:,u}\|_2^2} \leq 2\epsilon_{wd} - \epsilon_{wd}^2 + \epsilon_p^2 n(1+\epsilon_{wd})^2 = \epsilon, \quad (10)$$

*where $w_s$ is the base window size, $\epsilon_p$ is the sparsification parameter, and $\epsilon_{wd}$ is the error introduced by the sliding window.*

**Proof.** According to Lemma IV.1, incomplete Cholesky factorization takes $O(m + nnz(\boldsymbol{R}))$ time. For sparse graphs, both $m$ and $nnz(\boldsymbol{R})$ are typically $O(n)$. The main loop iterates $n$ times, with each iteration involving $O(w_s \log n + \log^2 n)$ for sparsifying the corresponding columns and $O(\log n)$ for maintenance of the sliding window. The final computation of $\tilde{\mathcal{H}}_u$ for all $u \in V$ takes $\tilde{O}(m)$. Summing these components yields the stated time complexity.

Let $\boldsymbol{S}_{:,u}$ be the exact $u$-th column of $\boldsymbol{R}^{-1}$, $\boldsymbol{S}^*_{:,u}$ be the approximation after utilizing the sliding window technique, and $\widetilde{\boldsymbol{S}}_{:,u}$ be the final sparsified result of $\boldsymbol{S}^*_{:,u}$.

For the sparsification error, our algorithm ensures

$$\frac{\|\boldsymbol{S}^*_{:,u} - \widetilde{\boldsymbol{S}}_{:,u}\|_1}{\|\boldsymbol{S}^*_{:,u}\|_1} \leq \epsilon_p.$$

Converting this 1-norm bound to 2-norm, we have

$$\|\boldsymbol{S}^*_{:,u} - \widetilde{\boldsymbol{S}}_{:,u}\|_2^2 \leq \|\boldsymbol{S}^*_{:,u} - \widetilde{\boldsymbol{S}}_{:,u}\|_1^2 \leq (\epsilon_p \|\boldsymbol{S}^*_{:,u}\|_1)^2 \leq \epsilon_p^2 n \|\boldsymbol{S}^*_{:,u}\|_2^2.$$

Therefore,

$$\frac{|\|\boldsymbol{S}^*_{:,u}\|_2^2 - \|\widetilde{\boldsymbol{S}}_{:,u}\|_2^2|}{\|\boldsymbol{S}^*_{:,u}\|_2^2} \leq \epsilon_p^2 n.$$

Let $\epsilon_{wd}$ be the relative error introduced by the sliding window, then we have

$$\frac{\|\boldsymbol{S}_{:,u} - \boldsymbol{S}^*_{:,u}\|_2}{\|\boldsymbol{S}_{:,u}\|_2} \leq \epsilon_{wd}.$$

The sparsification error is bounded as before. We can relate $\|\boldsymbol{S}^*_{:,u}\|_2$ to $\|\boldsymbol{S}_{:,u}\|_2$, then we obtain

$$\|\boldsymbol{S}^*_{:,u}\|_2 \leq \|\boldsymbol{S}_{:,u}\|_2 + \|\boldsymbol{S}_{:,u} - \boldsymbol{S}^*_{:,u}\|_2 \leq \|\boldsymbol{S}_{:,u}\|_2(1 + \epsilon_{wd}).$$

By combining these inequalities, we get

$$\begin{aligned}\|\boldsymbol{S}_{:,u}\|_2^2 - \|\widetilde{\boldsymbol{S}}_{:,u}\|_2^2 &\leq \|\boldsymbol{S}_{:,u}\|_2^2 - \|\boldsymbol{S}^*_{:,u}\|_2^2 + \|\boldsymbol{S}^*_{:,u} - \widetilde{\boldsymbol{S}}_{:,u}\|_2^2 \\ &\leq \|\boldsymbol{S}_{:,u}\|_2^2(2\epsilon_{wd} - \epsilon_{wd}^2 + \epsilon_p^2 n(1+\epsilon_{wd})^2).\end{aligned}$$

Applying the triangle inequality to the error terms allows us to derive the upper bound stated in the theorem.

The $\epsilon_{wd}$ error arises from using only the most recent vectors stored in $\widetilde{\boldsymbol{S}}$ to compute $\boldsymbol{S}^*_{:,u}$. For the $j$-th column, we can obtain

$$\boldsymbol{S}_{:,j} = \frac{\mathbf{e}_j}{\boldsymbol{R}_{jj}} - \sum_{i>j} \frac{\boldsymbol{R}_{ij}}{\boldsymbol{R}_{jj}} \boldsymbol{S}_{:,i}, \quad \boldsymbol{S}^*_{:,j} = \frac{\mathbf{e}_j}{\boldsymbol{R}_{jj}} - \sum_{i\in\Gamma} \frac{\boldsymbol{R}_{ij}}{\boldsymbol{R}_{jj}} \boldsymbol{S}_{:,i},$$

where $\Gamma$ denotes the set of indices of the most recently computed columns that are used in the approximation.

The $\epsilon_{wd}$ error can be expressed as

$$\epsilon_{wd} = \frac{\|\boldsymbol{S}_{:,j} - \boldsymbol{S}^*_{:,j}\|_2}{\|\boldsymbol{S}_{:,j}\|_2} = \frac{\|\sum_{i\notin\Gamma} \frac{\boldsymbol{R}_{ij}}{\boldsymbol{R}_{jj}} \boldsymbol{S}_{:,i}\|_2}{\|\boldsymbol{S}_{:,j}\|_2}.$$

While exact computation of $\epsilon_{wd}$ is challenging due to its dependence on the graph structure and algorithm's progression, our experiments demonstrate that the adaptive window size approach effectively balances error and efficiency. □

The error bound in Eq. (10) captures the impact of sparse inverse estimation and the sliding window technique in FASTCHOL. Notably, $\|\boldsymbol{S}_{:,u}\|_2^2$ approximates $(\mathcal{L}_v^{-1})_{uu}$, with accuracy depending on the incomplete Cholesky factorization quality. The overall precision of our method in approximating $(\mathcal{L}_v^{-1})_{uu}$ is thus influenced by both the error bound from Theorem IV.3 and the incomplete Cholesky factorization's effectiveness. Experimental results in Section 6 demonstrate that FASTCHOL maintains high accuracy in large-scale, real-world networks despite these cumulative approximations.

In terms of time complexity, FASTCHOL demonstrates competitive performance compared to the algorithm in [12], which has a complexity of $\tilde{O}(m/\epsilon^2)$, with $\epsilon$ typically set small for accuracy. Our algorithm generally exhibits lower time complexity, particularly in large networks where $m$ approaches

$n$. Additionally, the parameter $w_s$ allows for flexible management of the trade-off between computational complexity and accuracy.

## V. FastWalk Algorithm: Sampling Spanning Trees

In the previous section, we proposed an algorithm based on incomplete Cholesky factorization and sparse inverse estimation but could not provide the final cumulative approximation error. Moreover, as will be discussed in Section VI, FastChol has a non-negligible approximation error. Seeking a more accurate method, we note that the diagonal elements of $\mathcal{L}_v^{-1}$ are related to *random walk with a trap* on graphs [25], [45], where a random walk stops upon reaching the trapping node. Specifically, the $u$-th diagonal element of $\mathcal{L}_v^{-1}$ represents the expected number of visits to $u$ by a random walker starting from $u$ before meeting the trapping node $v$.

**Lemma V.1.** *[25] Given a network with node $v$ being the trap, let $\boldsymbol{t} \in \mathbb{R}^{(n-1)\times 1}$ denote the vector whose $u$-th element $\boldsymbol{t}_u$ represents the expected number of visits to node $u$ for a random walk starting from $u$ before meeting the trapping node $v$. Then $\boldsymbol{t}_u$ and $\mathcal{L}_v^{-1}$ can be connected as*

$$\boldsymbol{t}_u = (\mathcal{L}_v^{-1})_{uu}, \text{ for all } u \in V, u \neq v. \tag{11}$$

Based on the above lemma, a straightforward approach to approximate $\boldsymbol{t}$ would be simulating multiple random walks from each node in the network with the trap node $v$ and counting the number of returns to that node. However, this method is essentially a repetitive process performed $n-1$ times to approximate individual diagonal elements, flagging potential rooms for improvement. Furthermore, in this method, we only count and use the number of random walk's returns to the initial node during the simulation. This approach discards information from other steps of the random walk, which could potentially be useful.

To address this issue, we observe an interesting connection between the random walk with a trap and the loop-erased random walk [46], [47]. The loop-erasure operation on a random walk involves removing loops in the order they occur. In a graph $\mathcal{G}$ and a random walk trajectory $\gamma = (v_1, \cdots, v_l)$ on $\mathcal{G}$, we define the loop-erased trajectory $\mathrm{LE}(\gamma) = (v_{i_1}, \cdots, v_{i_j})$ by eliminating loops. Specifically, $i_j$ is determined iteratively: $i_1 = 1$, and $i_{j+1} = \max\{i \mid v_i = v_{i_j}\} + 1$. Loop-erased random walks avoid self-intersections and halt when revisiting the previous trajectory.

Wilson's algorithm, which applies loop-erasure to a random walk, was introduced to generate a uniform spanning tree rooted at a specified node [26]. The algorithm involves four key steps: (i) initializing a tree with the root node, (ii) fixing an arbitrary node ordering $u_1, \cdots, u_{n-1}$, (iii) following this order to perform an unbiased random walk until it reaches a node within the tree, and (iv) applying the loop-erasure operation to add nodes and edges to the tree. This process continues until all nodes are included in the tree, ensuring uniform sampling of all possible spanning trees [26]. Interestingly, when setting node $v$ as the root, the expected number of visits on each node $u$ equals $\boldsymbol{t}_u$ [26], which is the expected number of visits on $u$ for a random walk with a trap starting from $u$. This connection can be formally expressed as follows.

**Lemma V.2.** *[26] Suppose that we generate a spanning tree using Wilson's method with $v$ being the root. Let $\tilde{\boldsymbol{t}} \in \mathbb{R}^{(n-1)\times 1}$ be a vector of random variables with its $u$-th element being the random variable of the number of visits to a node $u$. Then, we have*

$$\mathbb{E}\left[\tilde{\boldsymbol{t}}_u\right] = \boldsymbol{t}_u = (\mathcal{L}_v^{-1})_{uu}, \text{ for all } u \in V, u \neq v. \tag{12}$$

Using this lemma, we can independently sample $l$ spanning trees with Wilson's algorithm, treating the trapping node $v$ as the root. In the $i$-th sample, we track the number of visits, $\tilde{t}_{ui}$, to each node $u \in V \setminus \{v\}$. Consequently, we can unbiasedly estimate $\boldsymbol{t}_u$ as $\tilde{\boldsymbol{t}}_u = \sum_{i=1}^{l} \tilde{t}_{ui}/l$, the average visits to node $u$ across the $l$ samples. Importantly, each step of the random walk yields valuable information. The main challenge is determining the number of samples needed to bound the approximate error, usually achieved using Hoeffding's Inequality.

**Theorem V.3.** *(Hoeffding's Inequality [48]). Let $Z_1, Z_2, \ldots, Z_{n_z}$ be i.i.d. random variables with $Z_j$ $(\forall 1 \leq j \leq n_z)$ being strictly bounded by the interval $[a, b]$. We define the average of these variables by $Z = \frac{1}{n_z}\sum_{j=1}^{n_z} Z_j$. Then, for any $\epsilon > 0$, we have*

$$\mathbb{P}[|Z - \mathbb{E}[Z]| \geq \epsilon] \leq 2\exp\left(\frac{-2\epsilon^2 n_z}{(b-a)^2}\right).$$

However, Theorem V.3 requires a finite upper bound of the estimator, which does not exist for $\tilde{t}_{ui}$. Instead, we bound $\tilde{t}_{ui}$ with a high probability, as shown in the following lemma.

**Lemma V.4.** *Given a graph $\mathcal{G}$ and a trap node $v$, if $t$ is selected satisfying $t \geq \log(m/(n\sqrt{n-1})\,\|\boldsymbol{d}_{-v}\|_2)/\log(\lambda)$, then*

$$\Pr\left(\tilde{t}_{ui} > t\right) \leq 1/n,$$

*where $\lambda$ denotes the largest absolute eigenvalue of matrix $\boldsymbol{P}_v$.*

**Proof.** Since $\tilde{t}_{ui}$ denotes the number of visits of node $u$ for the $i$-th sample, it is easy to verify that $\tilde{t}_{ui} \leq T_{uv}/2$, where $T_{uv}$ denotes the hitting time from $u$ to $v$ for this sample. According to Lemma 5.4 in [49], the proportion of samples of which node $u$ could not reach $v$ within $t$ steps is less than $1/n$, completing the proof. □

**Lemma V.5** (Lemma 5.4 from [49])**.** *Given a graph $\mathcal{G} = (V, E)$, a node $v \in V$, if maximum length $t$ of random walks satisfies $t = \log(m\beta/\sqrt{n-s}\,\|\boldsymbol{d}_{-v}\|_2)/\log(\lambda)$, where $\lambda$ is the spectral radius of $\boldsymbol{P}_{-v}$, then the expected fraction of random walks that do not reach $v$ within $t$ steps is less than $\beta$.*

Next, we introduce a theoretical bound for the sample size $l$.

**Lemma V.6.** *Given a connected graph $\mathcal{G} = (V, E)$, a trap node $v$, and an error parameter $\epsilon$, if we sample $l$ independent random spanning trees using the Wilson's method with $v$ being the root node, satisfying $l \geq 2\epsilon^{-2}\log 2n \log^2(m/(n\sqrt{n-1})\|\boldsymbol{d}_{-v}\|_2)/\log^2(\lambda)$,*

$$\Pr\left(\left|(\mathcal{L}_v^{-1})_{uu} - \tilde{\boldsymbol{t}}_u\right| \geq \frac{\epsilon}{2}\right) \leq \frac{2}{n}. \tag{13}$$

**Proof.** Lemma V.4 reveals that $\tilde{t}_{ui}$ exhibits an explicit upper bound $t = \log(m/(n\sqrt{n-1})\,\|\boldsymbol{d}_{-v}\|_2)/\log(\lambda)$ with a high probability. Plugging this into Theorem V.3, we obtain

$$\Pr\Big(|\tilde{\boldsymbol{t}}_u - \boldsymbol{t}_u| \geq \frac{\epsilon}{2}\Big) \leq 1 - \Big(1 - 2\exp\Big(-\frac{2l\epsilon^2}{4t^2}\Big)\Big)\Big(1 - \frac{1}{n}\Big) \leq 1 - \Big(1 - \frac{1}{n}\Big)^2 \leq 1 - \Big(1 - \frac{2}{n}\Big) = \frac{2}{n},$$

where the second inequality follows from $l \geq (2t^2 \log 2n)/\epsilon^2$. This concludes the proof. □

**Algorithm 3:** FASTWALK$(\mathcal{G}, v, \epsilon)$

**Input :** A connected graph $\mathcal{G} = (V, E)$; the pivot node $v$; the error parameter $\epsilon$
**Output :** $\tilde{\mathcal{H}}_u\ \forall u \in V$

$\theta \leftarrow \frac{\epsilon}{2\Delta d_{\max}\left(1+\frac{d_{\max}(1+n)}{2m}\right)^2\sqrt{mn\Delta}}$
$l \leftarrow 2\epsilon^{-2}\log 2n \log^2(m/(n\sqrt{n-1}))\|\boldsymbol{d}_{-v}\|_2)/\log^2(\lambda)$
$\tilde{\boldsymbol{t}} \leftarrow \mathbf{0}^{(n-1)\times 1}$
Fix an arbitrary ordering $(1, \cdots, n-1)$ of $V \setminus \{v\}$
**for** $i = 1 : l$ **do**
  $InTree[u] \leftarrow false, Next[u] \leftarrow -1\ \forall u \in V$
  $InTree[v] \leftarrow true$
  **for** $j = 1 : n-1$ **do**
    $u \leftarrow j, \tilde{\boldsymbol{t}}_u \leftarrow \tilde{\boldsymbol{t}}_u + \frac{1}{l}$
    **while** *!Intree[u]* **do**
      $Next[u] \leftarrow RandomNeighbor(u)$
      $u \leftarrow Next[u], \tilde{\boldsymbol{t}}_u \leftarrow \tilde{\boldsymbol{t}}_u + \frac{1}{l}$
    $\tilde{\boldsymbol{t}}_u \leftarrow \tilde{\boldsymbol{t}}_u - \frac{1}{l}$
    $u \leftarrow j$
    **while** *!InTree[u]* **do**
      $InTree[u] \leftarrow true, u \leftarrow Next[u]$
$\tilde{\mathcal{H}}_u \leftarrow$ APPRWC$(\mathcal{G}, v, \theta, \tilde{\boldsymbol{t}})$
**return** $\tilde{\mathcal{H}}_u\ \forall u \in V$

### A. The Sampling Algorithm

Building on our analysis from above, we introduce FASTWALK for approximating RWC, outlined in Algorithm 3. It takes a graph $\mathcal{G}$, a pivot node $v$, and an error parameter $\epsilon$ as input. The algorithm computes the error parameter $\theta$ for the Laplacian solver and determines the required sample size $l$ based on Lemma V.6. After initializing necessary data structures and establishing a fixed node order, FASTWALK samples $l$ rooted spanning trees using loop-erased random walks, efficiently tracking the next node to ensure loop erasure. During each sample, it records the visits to each node. Finally, it computes the approximate diagonal elements $\tilde{\boldsymbol{t}}$ of $\mathcal{L}_v^{-1}$ and uses the APPRWC algorithm to determine the RWC for each node as per Eq. (3). The following theorem summarizes the performance of FASTWALK.

**Theorem V.7.** *Algorithm 3 runs in* $O(l \times \mathrm{Tr}\left((\boldsymbol{I} - \boldsymbol{P}_v)^{-1}\right) + \tilde{O}(m))$*, where* $\mathrm{Tr}\left((\boldsymbol{I} - \boldsymbol{P}_v)^{-1}\right)$ *represents the trace of the matrix* $(\boldsymbol{I} - \boldsymbol{P}_v)^{-1}$*,* $l$ *is the sample size, and* $\tilde{O}(\cdot)$ *hides* $poly(\log n)$ *factors. The output ensures that the estimated RWC* $\tilde{\mathcal{H}}_u\ \forall u \in V$ *satisfies* $\pi_u|\tilde{\mathcal{H}}_u - \mathcal{H}_u| \leq \epsilon$.

**Proof.** The time complexity of Algorithm 3 primarily consists of two components. The first component involves executing the Laplacian solver once, which has a time complexity of $\tilde{O}(m)$. The second component is sampling the rooted spanning trees using Wilson's method. This is dominated by the number of visits to all nodes during the $l$ samplings, which corresponds to $l$ times the summation of the diagonal elements of the matrix $(\boldsymbol{I} - \boldsymbol{P}_v)^{-1}$. This summation is indeed equal to $l \times \mathrm{Tr}\left((\boldsymbol{I} - \boldsymbol{P}_v)^{-1}\right)$.

To align the relative error bound from the Laplacian solver with the absolute error in Eq. (13), we first observe that for $\boldsymbol{x} = \boldsymbol{L}^\dagger \boldsymbol{e}_v$ and $\tilde{\boldsymbol{x}} = \texttt{LapSolve}(\boldsymbol{L}, \boldsymbol{e}_v, \theta)$, the following inequalities hold for both vectors: $\sqrt{\lambda_2} \cdot \|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{x}\|_{\boldsymbol{L}} \leq \sqrt{4m} \cdot \|\boldsymbol{x}\|_\infty$ , where $\lambda_2$ is the second smallest eigenvalue of $\boldsymbol{L}$ [50]. Additionally, it is known that $\lambda_2 \geq 4/(n \cdot \Delta)$ [51], where $\Delta$ represents the graph diameter. Defining $\boldsymbol{M}_l = \big(\boldsymbol{I} - \frac{1}{2m}\boldsymbol{D}^{\frac{1}{2}}\boldsymbol{1}\boldsymbol{1}^\top\boldsymbol{D}^{\frac{1}{2}}\big)\boldsymbol{D}^{\frac{1}{2}}$ and $\boldsymbol{M}_r = \boldsymbol{D}^{\frac{1}{2}}\big(\boldsymbol{I} - \frac{1}{2m}\boldsymbol{D}^{\frac{1}{2}}\boldsymbol{1}\boldsymbol{1}^\top\boldsymbol{D}^{\frac{1}{2}}\big)$, we proceed as follows to calculate the infinity norm of the matrices $\boldsymbol{M}_l$ and $\boldsymbol{M}_r$. The diagonal elements of $\boldsymbol{M}_r$ are given by $(\boldsymbol{M}_r)_{ii} = \boldsymbol{d}_i^{1/2} - \frac{1}{2m}\boldsymbol{d}_i\boldsymbol{d}_i^{1/2}$, and the off-diagonal elements are $(\boldsymbol{M}_r)_{ij} = -\frac{1}{2m}\boldsymbol{d}_i\boldsymbol{d}_j^{1/2}$ for $i \neq j$. The infinity norm is the maximum row sum of the absolute values of the elements in $\boldsymbol{M}_r$. Thus, for each row $i$,

$$\sum_{j=1}^{n} |(\boldsymbol{M}_r)_{ij}| \leq |(\boldsymbol{M}_r)_{ii}| + \sum_{j=1, j\neq i}^{n} |(\boldsymbol{M}_r)_{ij}|.$$

By simplifying, we get

$$\sum_{j=1}^{n} |(\boldsymbol{M}_r)_{ij}| \leq \boldsymbol{d}_i^{1/2}\Big(1 + \frac{\boldsymbol{d}_i}{2m}\Big) + \frac{\boldsymbol{d}_i}{2m}\sum_{j=1,\, j\neq i}^{n} \boldsymbol{d}_j^{1/2}.$$

To obtain an upper bound, we note that $\boldsymbol{d}_i \leq d_{\max}$ and $\boldsymbol{d}_i^{1/2} \leq d_{\max}^{1/2}$, leading to

$$\sum_{j=1}^{n} |(\boldsymbol{M}_r)_{ij}| \leq d_{\max}^{1/2}\Big(1 + \frac{d_{\max}}{2m}\Big) + \frac{d_{\max}}{2m}\sum_{j=1,\, j\neq i}^{n} d_{\max}^{1/2}.$$

Recognizing that $\sum_{j=1,\, j\neq i}^{n} d_j^{1/2} \leq n d_{\max}^{1/2}$, we further simplify to

$$\sum_{j=1}^{n} |(\boldsymbol{M}_r)_{ij}| \leq d_{\max}^{1/2}\Big(1 + \frac{d_{\max}}{2m}\Big) + \frac{n d_{\max} d_{\max}^{1/2}}{2m}.$$

Therefore, the infinity norm is bounded by

$$\|\boldsymbol{M}_r\|_\infty \leq d_{\max}^{1/2}\Big(1 + \frac{d_{\max}(1+n)}{2m}\Big).$$

Using similar methods, we can derive the same infinity norm for matrix $\boldsymbol{M}_l$.

Building on the above results, we have

$$\begin{aligned}
&\|\mathcal{L}[:,v]-\tilde{\boldsymbol{y}}\|_\infty \le \|\boldsymbol{M}_l\|_\infty\|\boldsymbol{M}_r\|_\infty\|\tilde{\boldsymbol{x}}-\boldsymbol{x}\|_\infty \\
&\le \frac{\|\boldsymbol{M}_l\|_\infty^2}{\sqrt{\lambda_2}}\|\tilde{\mathbf{x}}-\mathbf{x}\|_{\boldsymbol{L}} \le \frac{\theta\|\boldsymbol{M}_l\|_\infty^2}{\sqrt{\lambda_2}}\|\mathbf{x}\|_{\boldsymbol{L}} \\
&\le \frac{\theta\|\boldsymbol{M}_l\|_\infty^2}{\sqrt{4/(n\Delta)}}\|\mathbf{x}\|_{\boldsymbol{L}} \le \frac{2\theta\|\boldsymbol{M}_l\|_\infty^2\sqrt{mn\Delta}}{2}\|\mathbf{x}\|_\infty \\
&\le \theta d_{\max}\Big(1+\frac{d_{\max}(1+n)}{2m}\Big)^2\Delta\sqrt{mn\Delta} \le \frac{\epsilon}{2}. \qquad (14)
\end{aligned}$$

The first inequality arises from basic linear algebra. The penultimate inequality comes from the fact that $\boldsymbol{x}$ expresses potentials scaled by $1/n$, arising from $n-1$ effective resistance problems fused together [35]. The maximum norm of $\boldsymbol{x}$ can thus be bounded by $(n-1)\frac{1}{n}R_{uv} \le \Delta$, because the graph diameter limits the effective resistance. The last inequality follows from

$$\theta = \frac{\epsilon}{2\Delta d_{\max}\left(1+\frac{d_{\max}(1+n)}{2m}\right)^2\sqrt{mn\Delta}}.$$

By combining Eq. (13) and Eq. (14), and considering that the pivot node is chosen with a much larger degree than other nodes, we conclude the proof. □

**Discussion: relation of FASTCHOL and FASTWALK.** Both algorithms arise from the pivot-based reformulation but estimate the same diagonal quantity of $\mathcal{L}_v^{-1}$ via *orthogonal* primitives: FASTCHOL leverages the positive definiteness of $\mathcal{L}_v$ to build a sparse triangular factor and read diagonals by cheap solves, whereas FASTWALK relies on the loop-erased walks to form a sample-based estimator. We therefore view them as complementary choices.

## VI. EXPERIMENTAL EVALUATION

### A. Experimental Setting

**Datasets.** To evaluate the efficiency and accuracy of our proposed algorithms, namely FASTCHOL and FASTWALK, we conduct experiments on 6 real world networks from SNAP [52] (Table I), which span several orders of magnitude in size and a wide range of sparsity—measured by average degree.

TABLE I
BASIC STATISTICS OF THE SIX STUDIED REAL-WORLD NETWORKS. DIA. AND AVGDEG. REPRESENT THE GRAPH DIAMETER AND THE AVERAGE DEGREE, RESPECTIVELY.

| Network | #nodes ($n$) | #edges ($m$) | Dia. ($\Delta$) | AvgDeg. |
|---|---|---|---|---|
| Facebook | 4,039 | 88,234 | 8 | 43.7 |
| DBLP | 317,080 | 1,049,866 | 23 | 6.6 |
| Youtube | 1,134,890 | 2,987,624 | 24 | 5.3 |
| Orkut | 3,072,441 | 117,185,083 | 10 | 76.3 |
| soc-LiveJournal | 3,997,962 | 34,681,189 | 16 | 17.3 |
| hetero-LiveJournal | 12,678,882 | 160,995,330 | 17 | 25.4 |

**Implementation Details.** All experiments are run on a Linux machine with an Intel Xeon(R) Gold 6240@2.60GHz 32-core processor and 256GB of RAM. Before timing, the graph data is loaded into memory. While most algorithms are implemented in Julia, the incomplete Cholesky factorization in FASTCHOL is implemented in MATLAB using the `ichol` function. Results are excluded if the algorithm does not return within 24 hours. For FASTCHOL and FASTWALK, the node with the highest degree is selected as the pivot node (see Eq. (3)), which reduces errors for FASTCHOL and improves efficiency for FASTWALK. The largest absolute eigenvalue $\lambda$ of $\boldsymbol{P}_v$ for each network is approximated via ARPACK [53]. In FASTCHOL, the drop tolerance $\delta$ is set to $10^{-4}$, $\zeta$ is set to $O(\log n)$, and the window size is generally $O(\log n)$. For fairness, the same $\theta$ value is used in FASTCHOL as in FASTWALK. Experiments are conducted on the largest connected component of each network using a single thread.

**Competitors and Ground Truth.** To showcase the superiority of our proposed algorithms, we compare them against the APPSOLVER algorithm from [12]. We compute the exact RWC as the ground truth for the Facebook dataset. For larger graphs, it is not computationally possible to obtain the exact value of RWC using the existing approaches. Therefore, considering the remarkable accuracy of FASTWALK as will be revealed in Fig. 2 (a), we treat the output of FASTWALK with $\epsilon = 0.0001$ as the ground truth for RWC.

**Evaluation Measures.** To evaluate the approximation *quality*, we measure the mean relative errors and the Kendall tau distance [54]: a metric used to quantify the degree of disagreement between the rankings produced by different methods. It is defined mathematically as: $K(L_1, L_2) = \frac{2(\alpha-\beta)}{n(n-1)}$. In this formula, $\alpha$ represents the number of concordant pairs, while $\beta$ denotes the number of discordant pairs. The value of the distance ranges from $-1$ to 1, where a value of 1 indicates complete agreement between the two rankings, and a value of $-1$ signifies total disagreement. We use the wall-clock running time to evaluate the *efficiency* of algorithms.

### B. Efficiency

We first investigate the efficiency of our algorithms FASTCHOL and FASTWALK. To this end, in Fig. 1, we report the running time of FASTCHOL, FASTWALK and that of the APPSOLVER algorithm from [12] when $\epsilon$ is varied from 0.01 to 0.3. The results show that for all three algorithms, the execution time decreases when $\epsilon$ increases, implying that there is a trade-off between running time and accuracy. Moreover, we can see that for all approximation parameters $\epsilon$, the computational time for FASTCHOL and FASTWALK is significantly smaller than that of APPSOLVER, especially for large-scale networks, which is consistent with our analysis in Sections IV and V. Thus, FASTCHOL and FASTWALK can significantly improve the efficiency compared to APPSOLVER. For the networks with more than 3 million nodes (Orkut, soc-Livejournal and hetero-Livejournal), the APPSOLVER algorithm runs out of memory. However, for these networks, we can approximately compute the RWC for all nodes using algorithms FASTCHOL and FASTWALK, further showcasing the efficiency and scalability of our proposed methods.

Regarding the running time comparison between FASTCHOL and FASTWALK, Fig. 1 demonstrates that FASTCHOL outperforms FASTWALK in most cases. This advantage is likely attributed to the use of sparsification and

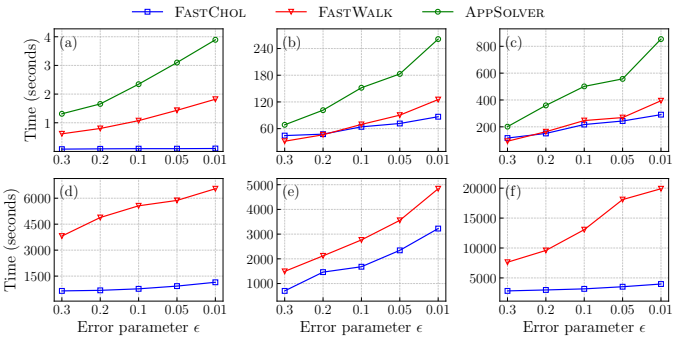


Fig. 1. Running time comparison for different algorithms on several networks: (a) Facebook, (b) DBLP, (c) Youtube, (d) Orkut, (e) soc-Livejournal and (f) hetero-Livejournal.

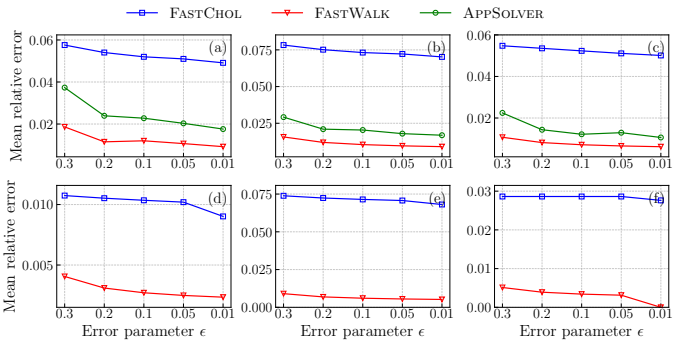


Fig. 2. Mean relative error for different algorithms on several networks: (a) Facebook, (b) DBLP, (c) Youtube, (d) Orkut, (e) soc-Livejournal and (f) hetero-Livejournal.

sliding window strategies in FastChol, whereas in dense graphs, the random walks in FastWalk may take too many steps before getting trapped.

### C. Approximation Quality

In addition to high efficiency, our algorithms also provide a great level of accuracy. To demonstrate this, we compare the results of FastWalk and FastChol with those of AppSolver. In Fig. 2, we report the mean relative errors of different algorithms, which is defined as $\frac{1}{n}\sum_{u\in V}|\mathcal{H}_u - \tilde{\mathcal{H}}_u|/\mathcal{H}_u$. Based on Fig. 2, generally for all three algorithms and choices of $\epsilon$, the error value is negligible. FastWalk provides the best accuracy among the three algorithms. While FastChol's error is slightly larger than FastWalk and AppSolver, it is significantly faster, as we discussed in the previous subsection. Thus, we conclude that the incomplete Cholesky factorization and the sparse inverse estimation techniques enhance the efficiency, with a minor compromise on the accuracy.

In many cases, nodes are ranked based on their centrality values. Therefore, we use the Kendall tau distance to measure the rank correlation between the approximated RWC and the ground truth values. We report in Fig. 3 the ranking correlation of different algorithms with $\epsilon = 0.1$. The results show that algorithm FastWalk always obtains higher Kendall tau distances than AppSolver, while FastChol obtains slightly lower Kendall tau distance than AppSolver.

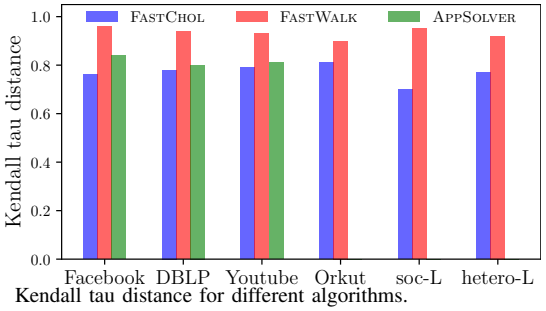


Fig. 3. Kendall tau distance for different algorithms.

TABLE II
PEAK MEMORY USAGE (GB) COMPARISON OVER SIX REAL-WORLD GRAPHS.

| | AppSolver | FastChol | FastWalk |
|---|---|---|---|
| Facebook | 3.781 | 0.575 | 0.729 |
| DBLP | 8.312 | 0.746 | 1.096 |
| Youtube | 53.829 | 1.526 | 2.356 |
| Orkut | – | 26.494 | 47.825 |
| soc-LiveJournal | – | 26.357 | 14.823 |
| hetero-LiveJournal | – | 11.496 | 67.561 |

However, this rank quality sacrifice is acceptable considering the distinguished efficiency performance of FastChol as shown in Fig. 1.

### D. Memory Overhead

To gauge the memory overhead of the three algorithms, AppSolver, FastChol, and FastWalk, we recorded the peak-resident memory each requires on six publicly available graphs (Table II). Missing entries indicate that the run terminated because available RAM was exhausted.

The measurements reveal a clear hierarchy. On the moderate-scale Facebook, DBLP, and Youtube graphs AppSolver consumes 3.8 GB, 8.3 GB, and 53.8 GB, respectively, while FastChol holds the footprint to 0.6–1.5 GB and FastWalk to 0.7–2.4 GB—up to a 35-fold reduction relative to the baseline. As graph size or density increases, the gap widens. AppSolver fails outright on Orkut, soc-LiveJournal, and hetero-LiveJournal, yet the other two methods still complete. On dense Orkut FastChol peaks at 26.5 GB, roughly half of FastWalk's 47.8 GB; on the sparser soc-LiveJournal FastWalk drops to 14.8 GB while FastChol rises to 26.4 GB; on hetero-LiveJournal FastChol again leads with 11.5 GB, whereas FastWalk climbs to 67.6 GB. Overall, FastChol delivers the most consistent, and typically the lowest, memory footprint, extending the solvable range well beyond that of the solver, while FastWalk provides a competitive alternative whose footprint varies with graph structure.

### E. Ablation Study

To better understand the source of FastChol's performance, we now conduct a detailed ablation study. This analysis focuses on FastChol due to its multi-component design, whereas FastWalk's behavior is more directly governed by the sample size $l$. We evaluate the full algorithm against

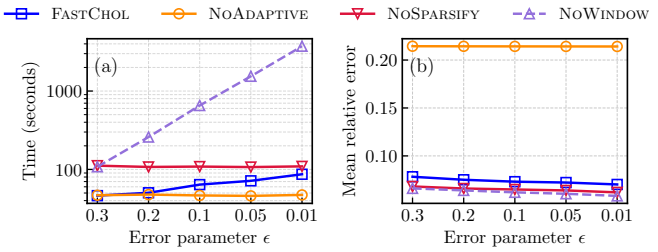


Fig. 4. Ablation study for FastChol on the effects of key components. (a) time comparison and (b) error comparison.

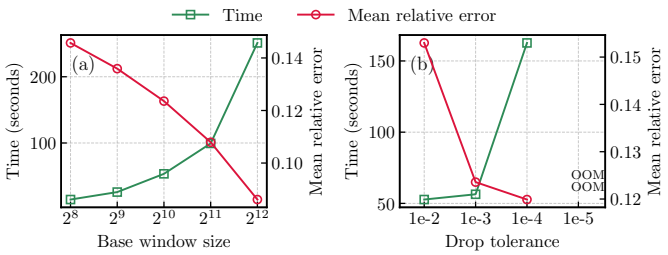


Fig. 5. Sensitivity of hyperparameters: (a) base window size and (b) drop tolerance.

three variants to isolate the contribution of our key optimization on the DBLP dataset, disabling the sparsification step (NoSparsify), removing the sliding window while utilizing the full available columns (NoWindow), and using a fixed window (NoAdaptive). As shown in Fig. 4, the results highlight the distinct role of each optimization. Note that the y-axis for time in subplot (a) is on a logarithmic scale to better visualize the differences.

The sliding window is essential for efficiency: removing it in NoWindow makes runtime rise sharply as $\epsilon$ decreases and yields the slowest curve. With windowing retained, the running time of FastChol and NoSparsify stays nearly flat in $\epsilon$; NoSparsify is consistently slower than FastChol due to denser operators, and NoAdaptive offers no clear runtime advantage over FastChol. On accuracy, NoAdaptive has the largest errors across $\epsilon$, demonstrating the importance of our adaptive strategy for maintaining accuracy. Among the other three, errors are close— FastChol is slightly less accurate than NoSparsify, which is slightly less accurate than NoWindow—in exchange for FastChol's pronounced speedup.

### F. Hyperparameter Sensitivity

To provide guidance for parameter tuning, we investigate the sensitivity of FastChol's performance to its key hyperparameters: the base window size ($w_s$) and the IChol drop tolerance ($\delta$). The analysis is conducted on the DBLP dataset by varying one parameter at a time. The impact on running time and relative error is reported in Fig. 5.

As shown in Fig. 5 (a), we observe a clear trade-off associated with the base window size. Specifically, increasing $w_s$ leads to a significant reduction in mean relative error, but at the cost of a corresponding increase in computation time. Similarly, Fig. 5 (b) illustrates that a smaller drop tolerance $\delta$ enhances accuracy but also increases the running time. Extremely small values for $\delta$ can lead to out-of-memory errors. These results highlight the importance of balancing these parameters to achieve a desired trade-off between computational efficiency and approximation quality.

## VII. Related Work

Identifying "crucial" nodes is a fundamental problem in network science and graph data mining [14], [55], essential for network analysis with applications across various fields [56]–[60]. Over the decades, many centrality measures have been introduced to characterize nodes' roles in networks [1], [61]–[64]. Among various centrality indices, betweenness centrality and closeness centrality are arguably the two most frequently used ones, especially in social network analysis [6], [65]. However, both metrics only consider the shortest paths, neglecting contributions from other paths, which can lead to counterintuitive results [66]. To address this, measures accounting for all paths between nodes have been proposed, such as PageRank [5], [67], Katz centrality [68], and information centrality [29], [69]. This paper focuses on random walk centrality [8]–[10], a powerful centrality measure which is capable of capturing complex global structural information leveraging information from all paths.

Due to the extensive applications of random walk centrality across domains [15]–[19], its computation has garnered significant attention. Exact computation requires cubic time relative to the number of nodes, which is impractical for large networks. The state-of-the-art method proposed by Zhang et al. [12] expresses random walk centrality in terms of quadratic forms of the Laplacian pseudo-inverse and develops a fast algorithm based on the Johnson-Lindenstrauss lemma and Laplacian solver. Nevertheless, the substantial memory requirements of this Laplacian solver limit its practicality for large-scale networks comprising millions of edges. Additionally, their approach requires $O(\log n/\epsilon^2)$ calls to the Laplacian solver for an error parameter $\epsilon$, making it computationally expensive.

In this paper, we introduced a fast algorithm leveraging the positive definiteness of the normalized Laplacian submatrix, utilizing incomplete Cholesky factorization and the sparse inverse of the factorization components. Additionally, we proposed a novel algorithm based on spanning tree sampling, derived from the relation between the inverse of the normalized Laplacian submatrix and random walks. While matrix factorization and spanning tree sampling techniques are widely used in designing various graph algorithms [13], [24], [39], [40], [70]–[75], our approach uniquely addresses the computation of RWC in a non-trivial manner.

## VIII. Conclusion

In this paper, we presented a new pivot-based formulation for computing Random Walk Centrality (RWC). This formulation reduces the core computation to a *single* Laplacian system solve and a subsequent diagonal block estimation, significantly lowering the problem's theoretical complexity. From this formulation, we developed FastChol and FastWalk

by pioneering the use of incomplete Cholesky factorization and loop-erased random walks for large-scale RWC computation. Experiments on real-world networks with up to 160 million edges show our algorithms' superior performance, delivering up to 35× memory and 10× time savings over the state-of-the-art method while maintaining comparable accuracy. Notably, our algorithms succeed even when the baseline method fails due to memory limitations. These achievements make full-graph RWC analysis practical on common commercial hardware, opening new doors for research previously limited by network scale.

Future work could enhance the current algorithms, for instance by improving the accuracy of the sparse Cholesky approximation and devising more efficient sampling techniques. Beyond this, we will also explore a lightweight hybrid that adaptively balances the two methods (see Section V for a brief discussion). Extending the framework is also a promising direction, including adapting the methods for real-time computation on dynamic graphs and generalizing them to calculate RWC for groups of nodes.

## REFERENCES


[1] L. Lü, D. Chen, X.-L. Ren, Q.-M. Zhang, Y.-C. Zhang, and T. Zhou, "Vital nodes identification in complex networks," *Physics Reports*, vol. 650, pp. 1–63, 2016.

[2] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 137–146.

[3] Y. Yang, A. McKhann, S. Chen, G. Harling, and J.-P. Onnela, "Efficient vaccination strategies for epidemic control using network information," *Epidemics*, vol. 27, pp. 115–122, 2019.

[4] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks*, vol. 30, pp. 107–117, 1998.

[5] D. F. Gleich, "Pagerank beyond the web," *SIAM Review*, vol. 57, no. 3, pp. 321–363, 2015.

[6] A. Bavelas, "A mathematical model for group structures," *Human Organization*, vol. 7, no. 3, pp. 16–30, 1948.

[7] P. Bonacich, "Factoring and weighting approaches to status scores and clique identification," *Journal of Mathematical Sociology*, vol. 2, no. 1, pp. 113–120, 1972.

[8] J. D. Noh and H. Rieger, "Random walks on complex networks," *Physical Review Letters*, vol. 92, no. 11, p. 118701, 2004.

[9] L. Lovász, "Random walks on graphs: A survey," *Combinatorics, Paul Erdös is eighty*, vol. 2, no. 1, pp. 1–46, 1993.

[10] C. Mavroforakis, M. Mathioudakis, and A. Gionis, "Absorbing random-walk centrality: Theory and algorithms," in *IEEE International Conference on Data Mining*. IEEE, 2015, pp. 901–906.

[11] H.-M. Chun, S. Hwang, B. Kahng, H. Rieger, and J. D. Noh, "Heterogeneous mean first-passage time scaling in fractal media," *Physical Review Letters*, vol. 131, no. 22, p. 227101, 2023.

[12] Z. Zhang, W. Xu, and Z. Zhang, "Nearly linear time algorithm for mean hitting times of random walks on a graph," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020, pp. 726–734.

[13] H. Xia, W. Xu, Z. Zhang, and Z. Zhang, "Means of hitting times for random walks on graphs: Connections, computation, and optimization," *ACM Transactions on Knowledge Discovery from Data*, vol. 19, no. 2, 2025.

[14] Q. Bao and Z. Zhang, "Discriminating power of centrality measures in complex networks," *IEEE Trans. Cybern.*, vol. 52, no. 11, pp. 12 583–12 593, 2022.

[15] M. Loecher and J. Kadtke, "Enhanced predictability of hierarchical propagation in complex networks," *Physics Letters A*, vol. 366, no. 6, pp. 535–539, 2007.

[16] F. Blöchl, F. J. Theis, F. Vega-Redondo, and E. O. Fisher, "Vertex centralities in input-output networks reveal the structure of modern economies," *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, vol. 83, no. 4, p. 046127, 2011.

[17] B. C. Johnson and S. Kirkland, "Estimating random walk centrality in networks," *Computational Statistics & Data Analysis*, vol. 138, pp. 190–200, 2019.

[18] S. Oldham, B. Fulcher, L. Parkes, A. Arnatkevićiūtė, C. Suo, and A. Fornito, "Consistency and differences between centrality measures across distinct classes of networks," *PloS One*, vol. 14, no. 7, p. e0220061, 2019.

[19] A. P. Riascos, D. Boyer, P. Herringer, and J. L. Mateos, "Random walks on networks with stochastic resetting," *Physical Review E*, vol. 101, no. 6, p. 062147, 2020.

[20] D. A. Spielman and S. Teng, "Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems," *SIAM Journal on Matrix Analysis and Applications*, vol. 35, no. 3, pp. 835–885, 2014.

[21] M. B. Cohen, R. Kyng, G. L. Miller, J. W. Pachocki, R. Peng, A. B. Rao, and S. C. Xu, "Solving sdd linear systems in nearly $m \log^{1/2} n$ time," in *Proceedings of the 46th Annual Symposium on Theory of Computing*. ACM, 2014, pp. 343–352.

[22] Y. Gao, R. Kyng, and D. A. Spielman, "Robust and practical solution of laplacian equations by approximate elimination," *CoRR*, vol. abs/2303.00709, 2023.

[23] A. Beveridge, "A hitting time formula for the discrete Green's function," *Combinatorics, Probability and Computing*, vol. 25, no. 3, pp. 362–379, 2016.

[24] P. Herholz and O. Sorkine-Hornung, "Sparse Cholesky updates for interactive mesh parameterization," *ACM Transactions on Graphics*, vol. 39, no. 6, 2020.

[25] Z. Zhang, Y. Yang, and Y. Lin, "Random walks in modular scale-free networks with multiple traps," *Physical Review E*, vol. 85, p. 011106, 2012.

[26] D. B. Wilson, "Generating random spanning trees more quickly than the cover time," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, 1996, pp. 296–303.

[27] Y. Lin and Z. Zhang, "Random walks in weighted networks with a perfect trap: An application of Laplacian spectra," *Physical Review E*, vol. 87, no. 6, p. 062140, 2013.

[28] S. Condamin, O. Bénichou, V. Tejedor, R. Voituriez, and J. Klafter, "First-passage times in complex scale-invariant media," *Nature*, vol. 450, no. 7166, pp. 77–80, 2007.

[29] M. E. J. Newman, "A measure of betweenness centrality based on random walks," *Social Networks*, vol. 27, no. 1, pp. 39–54, 2005.

[30] V. Tejedor, O. Bénichou, and R. Voituriez, "Global mean first-passage times of random walks on complex networks," *Physical Review E*, vol. 80, no. 6, p. 065104, 2009.

[31] A. Beveridge, "Centers for random walks on trees," *SIAM Journal on Discrete Mathematics*, vol. 23, no. 1, pp. 300–318, 2009.

[32] P. G. Doyle and J. L. Snell, *Random Walks and Electric Networks*. Mathematical Association of America, 1984.

[33] Y. Li and Z.-L. Zhang, "Random walks and Green's function on digraphs: A framework for estimating wireless transmission costs," *IEEE/ACM Transactions on Networking*, vol. 21, no. 1, pp. 135–148, 2013.

[34] W. Thomson and G. E. Green, "An essay on the application of mathematical analysis to the theories of electricity and magnetism." *Journal für die reine und angewandte Mathematik*, pp. 73 – 89, 1850.

[35] P. Tetali, "Random walks and the effective resistance of networks," *Journal of Theoretical Probability*, vol. 4, no. 1, pp. 101–109, 1991.

[36] N. S. Izmailian, R. Kenna, and F. Wu, "The two-point resistance of a resistor network: a new formulation and application to the cobweb network," *Journal of Physics A: Mathematical and Theoretical*, vol. 47, no. 3, p. 035003, 2013.

[37] E. Bozzo, "The moore–penrose inverse of the normalized graph Laplacian," *Linear Algebra and its Applications*, vol. 439, no. 10, pp. 3038–3043, 2013.

[38] J. McDonald, M. Neumann, H. Schneider, and M. Tsatsomeros, "Inverse M-matrix inequalities and generalized ultrametric matrices," *Linear Algebra and its Applications*, vol. 220, pp. 321–341, 1995.

[39] T. A. Davis, S. Rajamanickam, and W. M. Sid-Lakhdar, "A survey of direct methods for sparse linear systems," *Acta Numerica*, vol. 25, pp. 383–566, 2016.

[40] M. Benzi, "Preconditioning techniques for large linear systems: a survey," *Journal of Computational Physics*, vol. 182, no. 2, pp. 418–477, 2002.

[41] Z. Liu and W. Yu, "Computing effective resistances on large graphs based on approximate inverse of Cholesky factor," in *Design, Automation & Test in Europe Conference*. IEEE, 2023.

[42] Y. Saad, *SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations*. NASA Ames Research Center, 1994.

[43] C.-J. Lin and J. J. Moré, "Incomplete Cholesky factorizations with limited memory," *SIAM Journal on Scientific Computing*, vol. 21, no. 1, pp. 24–45, 1999.

[44] J. A. Meijerink and H. A. van der Vorst, "An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix," *Mathematics of Computation*, vol. 31, no. 137, pp. 148–162, 1977.

[45] Z. Z. Zhang, Y. Qi, S. G. Zhou, W. L. Xie, and J. H. Guan, "Exact solution for mean first-passage time on a pseudofractal scale-free web," *Physical Review E*, vol. 79, no. 2, p. 021127, 2009.

[46] Lawler and F. Gregory, "A self-avoiding random walk," *Duke Mathematical Journal*, vol. 47, no. 3, pp. 655–693, 1980.

[47] G. F. Lawler, *Intersections of Random Walks*, ser. Modern Birkhäuser Classics. Springer New York, 2012.

[48] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, 1963.

[49] X. Zhou and Z. Zhang, "Opinion maximization in social networks via leader selection," in *Proceedings of the ACM Web Conference*. ACM, 2023, pp. 133–142.

[50] E. Angriman, M. Predari, A. van der Grinten, and H. Meyerhenke, "Approximation of the diagonal of a Laplacian's pseudoinverse for complex network analysis," in *28th Annual European Symposium on Algorithms*, vol. 173. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020, pp. 6:1–6:24.

[51] N. M. M. De Abreu, "Old and new results on algebraic connectivity of graphs," *Linear Algebra and its Applications*, vol. 423, no. 1, pp. 53–73, 2007.

[52] J. Leskovec and R. Sosič, "SNAP: A general-purpose network analysis and graph-mining library," *ACM Transactions on Intelligent Systems and Technology*, vol. 8, no. 1, p. 1, 2016.

[53] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. Society for Industrial and Applied Mathematics, 1998.

[54] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[55] A. N. Langville and C. D. Meyer, *Who's# 1?: the science of rating and ranking*. Princeton University Press, 2012.

[56] M. E. J. Newman, *Networks: An Introduction*. Oxford University Press, Oxford, UK, 2010.

[57] S. Zhang, R. Yang, J. Tang, X. Xiao, and B. Tang, "Efficient approximation algorithms for spanning centrality," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, 2023, pp. 3386–3395.

[58] L. Pellegrina, "Efficient centrality maximization with rademacher averages," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2023, pp. 1872–1884.

[59] C. Cousins, C. Wohlgemuth, and M. Riondato, "Bavarian: Betweenness centrality approximation with variance-aware rademacher averages," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, 2021, pp. 196–206.

[60] A. M. de Lima, M. V. G. da Silva, and A. L. Vignatti, "Estimating the percolation centrality of large networks through pseudo-dimension theory," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, 2020, pp. 1839–1847.

[61] S. White and P. Smyth, "Algorithms for estimating relative importance in networks," in *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2003, pp. 266–275.

[62] P. Boldi and S. Vigna, "Axioms for centrality," *Internet Mathematics*, vol. 10, no. 3-4, pp. 222–262, 2014.

[63] M. Benzi and C. Klymko, "On the limiting behavior of parameter-dependent network centrality measures," *SIAM Journal on Matrix Analysis and Applications*, vol. 36, no. 2, pp. 686–706, 2015.

[64] F. Bonchi, G. De Francisci Morales, and M. Riondato, "Centrality measures on big graphs: Exact, approximated, and distributed algorithms," in *Proceedings of the International Conference Companion on World Wide Web*, 2016, pp. 1017–1020.

[65] A. Bavelas, "Communication patterns in task-oriented groups," *The Journal of the Acoustical Society of America*, vol. 22, no. 6, pp. 725–730, 1950.

[66] E. Bergamini, M. Wegner, D. Lukarski, and H. Meyerhenke, "Estimating current-flow closeness centrality with a multigrid Laplacian solver," in *Proceedings of the 7th SIAM Workshop on Combinatorial Scientific Computing*, 2016, pp. 1–12.

[67] F. R. K. Chung and W. Zhao, *PageRank and Random Walks on Graphs*. Berlin, Heidelberg: Springer, 2010, pp. 43–62.

[68] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, 1953.

[69] K. Stephenson and M. Zelen, "Rethinking centrality: Methods and examples," *Social Networks*, vol. 11, no. 1, pp. 1–37, 1989.

[70] M. Liao, R. Li, Q. Dai, H. Chen, H. Qin, and G. Wang, "Efficient resistance distance computation: The power of landmark-based approaches," in *Proceedings of the ACM on Management of Data*, vol. 1, 2023, pp. 1–27.

[71] S. Wang, R. Yang, X. Xiao, Z. Wei, and Y. Yang, "FORA: simple and effective approximate single-source personalized PageRank," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 505–514.

[72] C. Li, C. An, Z. Gao, F. Yang, Y. Su, and X. Zeng, "Unleashing the power of graph spectral sparsification for power grid analysis via incomplete Cholesky factorization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 9, pp. 3053–3066, 2023.

[73] R. Inayoshi, H. N. Aung, and H. Ohsaki, "Bloomwalk and cuckoowalk: Fast random walks utilizing probabilistic data structure," in *IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE Computer Society, 07 2024, pp. 342–349.

[74] S. Li, X. Huang, and C.-H. Lee, "An efficient and scalable algorithm for estimating kemeny's constant of a markov chain on large graphs," in *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. ACM, 2021, pp. 964–974.

[75] H. Xia and Z. Zhang, "Efficient approximation of kemeny's constant for large graphs," *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, 2024.

PLACE PHOTO HERE

**Changan Liu** received the B.Eng. degree from the School of Software, Dalian University of Technology, Dalian, China, in 2019. He received the Ph.D. degree from the School of Computer Science, Fudan University, Shanghai, China, in 2024. He is currently a Research Fellow at Nanyang Technological University, Singapore. His research interests include network science, computational social science, graph data mining, social network analysis, and graph foundation models.

PLACE PHOTO HERE

**Zixuan Xie** received the B.Eng. degree in software engineering from Fudan University, Shanghai, China, in 2024. She is currently pursuing the Ph.D. degree in Department of Computer Science, University of Virginia, Charlottesville, United States. Her research interests include reinforcement learning theory, stochastic approximation, and in-context learning.

PLACE PHOTO HERE

**Ahad N. Zehmakan** received his PhD degree in 2020 from ETH Zurich and is currently a faculty member in the School of Computing at the Australian National University. His research interests include graph algorithms, social network analysis, and graph neural networks.

PLACE PHOTO HERE

**Zhongzhi Zhang** (M'19) received the B.Sc. degree in applied mathematics from Anhui University, Hefei, China, in 1997 and the Ph.D. degree in management science and engineering from Dalian University of Technology, Dalian, China, in 2006. From 2006 to 2008, he was a Post-Doctoral Research Fellow with Fudan University, Shanghai, China, where he is currently a Full Professor with the College of Computer Science and Artificial Intelligence. He has published over 200 papers in international journals or conferences. Since 2019, he has been selected as one of the most cited Chinese researchers (Elsevier) every year. His current research interests include network science, graph data mining, social network analysis, computational social science, spectral graph theory, and random walks.

Dr. Zhang was a recipient of the Excellent Doctoral Dissertation Award of Liaoning Province, China, in 2007, the Excellent Post-Doctor Award of Fudan University in 2008, the Shanghai Natural Science Award (third class) in 2013, the Wilkes Award for the best paper published in The Computer Journal in 2019, and the CCF Natural Science Award (second class) in 2022.