Scaling up Graph Laplacian Pseudo-inverse Computation

Meihao Liao*, Rong-Hua Li*, Qiangqiang Dai*, Hongyang Chen†, Zhenjun Li‡, Guoren Wang*

*Beijing Institute of Technology, China ‡Zhejiang Lab, China ‡Shenzhen Institute of Information Technology, China

{mhliao@bit.edu.cn,lironghuabit@126.com,qiangd66@gmail.com,dr.h.chen@ieee.org,15323940@qq.com,wanggrbit@gmail.com}

ABSTRACT

The pseudo-inverse of a graph Laplacian matrix, denoted as $L^\dagger$, finds extensive application in various graph analysis tasks. Notable examples include the calculation of electrical closeness centrality, determination of Kemeny constant, and evaluation of resistance distance. However, existing algorithms for computing $L^\dagger$ are often computationally expensive when dealing with large graphs. To overcome this challenge, we propose novel solutions for approximating $L^\dagger$ by establishing a connection with the inverse of a Laplacian submatrix L_v. This submatrix is obtained by removing the v-th row and column from the original Laplacian matrix L. The key advantage of this connection is that L_v^{-1} exhibits various interesting combinatorial interpretations. We present two innovative interpretations of L_v^{-1} based on spanning trees and loop-erased random walks, which allow us to develop efficient sampling algorithms. Building upon these new theoretical insights, we propose two novel algorithms for efficiently approximating both electrical closeness centrality and Kemeny constant. We extensively evaluate the performance of our algorithms on five real-life datasets. The results demonstrate that our novel approaches significantly outperform the state-of-the-art methods by several orders of magnitude in terms of both running time and estimation errors for these two graph analysis tasks.

1 INTRODUCTION

Given an n node graph $G = (V, E)$ with an adjacency matrix A and a diagonal degree matrix D (each diagonal element is a degree of a node), two types of widely-used Laplacians are defined: the combinatorial Laplacian $L = D - A$ and the normalized Laplacian $\mathcal{L} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. Since the inverses of L and $\mathcal{L}$ do not exist, previous studies often employ the Moore-Penrose pseudo-inverses $L^\dagger$ and $\mathcal{L}^\dagger$ [4, 16, 57] as alternatives. These Laplacian pseudo-inverses have proven to be crucial in various graph analysis applications, enabling the computation of important quantities such as electrical closeness centrality [16], Kemeny constant [10], Kirchhoff index [68], resistance distance [39, 65], hitting time [20, 21, 51], and generalized graph spectral distance [58]. These random walk-based quantities are also widely used in designing graph embedding system [52, 67], clustering in geo-social networks [55], long tail recommendation [66] and anomalous detection in time varying graphs [56].

Among the various applications of Laplacian pseudo-inverses, we mainly focus on two typical applications: graph centrality and graph invariants. Graph centrality is used to quantify the importance of nodes within a network. Noteworthy centrality measures include PageRank [29, 34], eigenvector centrality [14], closeness centrality [46] and electrical closeness centrality [17, 27]. Among these centrality metrics, computing electrical closeness centrality is equivalent to computing the diagonal elements of the combinatorial Laplacian pseudo-inverse $L^\dagger$ [17, 26]. Unlike centrality measures based on shortest path distance (e.g., closeness centrality), electrical closeness centrality incorporates resistance distance between nodes [17, 26]. Since it considers all paths between nodes, the resulting centrality metric is often more robust. Its effectiveness has been demonstrated in diverse applications such as power grid vulnerability [61] and recommendation systems [28]. It has also been employed to measure the graph variance and graph curvature from a geometric perspective [23, 24]. Graph invariant, on the other hand, characterizes global properties of the entire graph. A notable graph invariant is Kemeny constant [10], which corresponds to the trace of the pseudo-inverse of the normalized Laplacian matrix $\mathcal{L}^\dagger$. Kemeny constant captures the extent of inter-connectivity among nodes in a graph. It finds broad applications in domains including robot surveillance [25] and web search [36].

However, the computation of Laplacian pseudo-inverse poses significant challenges when dealing with large graphs. The direct method involves eigen-decomposition, which has a time complexity of $O(n^3)$. Previous studies have formulated $L^\dagger$ as $L^\dagger = (L + \frac{\vec{1}\vec{1}^T}{n})^{-1} - \frac{\vec{1}\vec{1}^T}{n}$ [16, 57]. However, $L + \frac{\vec{1}\vec{1}^T}{n}$ becomes dense, making the computation of its inverse challenging. Additionally, for each column of $L^\dagger$, the state-of-the-art (SOTA) method [57] proved that the s-th column of $L^\dagger$ is the solution of the Laplacian linear system $Lx = e_s - \frac{\vec{1}}{n}$, where e_s is the unit vector with a value of 1 in the s-th position and zeros elsewhere, and $\vec{1}$ denotes an all-one vector. However, solving this Laplacian system remains computationally expensive for large networks. Moreover, computing electrical closeness centrality and Kemeny constant requires solving n Laplacian systems, which is extremely expensive for large graphs.

In this paper, we address the challenge of computing Laplacian pseudo-inverse by establishing a connection with L_v^{-1}, the inverse of a submatrix obtained by removing the v-th row and column from the Laplacian matrix L. The formulation of L_v offers three advantages: (i) L_v preserves the sparsity of L, enabling efficient computation of its inverse; (ii) While there is a lack of intuitive combinatorial interpretations for the elements of $L^\dagger$ or $\mathcal{L}^\dagger$ in the existing literature, several intriguing combinatorial interpretations exist for the elements of L_v^{-1} [8, 19, 39]; (iii) As both $L^\dagger$ and $\mathcal{L}^\dagger$ remain invariants when varying the choice of v, we can heuristically select an optimal landmark node v to further enhance the efficiency of the computation. Specifically, we present new formulas to express the elements of the Laplacian pseudo-inverse, $L^\dagger$ and $\mathcal{L}^\dagger$, in terms of L_v^{-1}. Additionally, we propose two novel combinatorial interpretations for the elements of L_v^{-1} based on spanning trees and loop-erased walks. Since spanning trees can be sampled using

loop-erased walks, these new interpretations of L_v^{-1} enable the estimation of the elements of $L^\dagger$ and $\mathcal{L}^\dagger$ within the time required to sample a set of loop-erased walks.

We apply our newly developed techniques to tackle two graph analysis tasks: electrical closeness centrality computation, specifically targeting the approximation of the diagonal elements of the pseudo-inverse of the combinatorial Laplacian matrix $L^\dagger$, and Kemeny constant computation, specifically aiming to approximate the trace of the pseudo-inverse of the normalized Laplacian matrix, denoted as $Tr(\mathcal{L}^\dagger)$. For both computational tasks, we propose two novel algorithms based on spanning tree sampling and loop-erased walk sampling. These algorithms demonstrate superior efficiency when compared to the SOTA approaches for approximating both electrical closeness centrality and Kemeny constant.

We conduct extensive experiments on five real-world networks to evaluate the performance of the proposed algorithms for the two computational tasks. The experimental results demonstrate that our algorithms outperform the SOTA approaches in terms of both running time and estimation error. Regarding the approximation of electrical closeness centrality, our spanning tree sampling algorithm exhibits competitive and slightly better performance compared to the SOTA methods, while our loop-erased walk sampling algorithm achieves much lower execution times with similar estimation errors. For the computation of Kemeny constant, our loop-erased walk-based algorithms demonstrate significant improvements. They are around two orders of magnitude faster than the SOTA algorithms while maintaining a comparable level of estimation error. Our contributions are summarized as follows:

New theoretical results. We present new formulas that express $L^\dagger$ and $\mathcal{L}^\dagger$ in terms of L_v^{-1}. Additionally, we introduce two novel combinatorial interpretations of L_v^{-1} based on spanning trees and loop-erased walks. These theoretical results provide valuable insights and can be of independent interests to the research community.

Novel algorithms. We propose two innovative algorithms for electrical closeness centrality approximation, employing spanning tree sampling and loop-erased walk sampling techniques. Similarly, we develop two novel algorithms for approximating Kemeny constant, utilizing spanning tree and loop-erased walk sampling approaches. Notably, the time complexity of drawing a sample in all the proposed algorithms is $Tr((I - P_v)^{-1})$, where P_v is the matrix obtained by removing the v-th row and column of the probability transition matrix $P = D^{-1}A$. As verified in our experiments, $Tr((I - P_v)^{-1})$ is $O(n)$ for real-world graphs, thus our sampling approaches are very efficient. Furthermore, we provide rigorous analysis of these algorithms to demonstrate their efficiency and effectiveness.

Extensive experiments. We conduct extensive experiments using five real-life datasets to evaluate the performance of our algorithms. The results show that our algorithms significantly outperform the SOTA approaches in terms of both running time and estimation error. For example, for computing Kemeny constant on a million-node graph Youtube, to achieve a relative error 10^{-4}, the SOTA method requires 34, 637 seconds, while our algorithm LEWalk takes only 1, 615 seconds, resulting in a 21.45× speed improvement. For reproducibility purpose, we make the source code of our work available at https://anonymous.4open.science/r/KC-KDD-0FC7.

2 PRELIMINARIES

Consider an unweighted, undirected connected graph denoted as $G = (V, E)$, where $|V| = n$ denotes the number of nodes and $|E| = m$ represents the number of edges. Let A be the adjacency matrix of G, where $(A)_{ij} = 1$ if there exists an edge between nodes i and j. Additionally, we define the degree matrix D as a diagonal matrix with $(D)_{ii} = d_i$, representing the degree of node i. To define a simple random walk on G, we use the probability transition matrix

$P = D^{-1}A$. In each step, the random surfer jumps to a neighbor of the current node u with a probability of $\frac{1}{d_u}$, where d_u is the degree of node u. It is well-known that the stationary distribution of such a random walk can be represented by the vector $\boldsymbol{\pi}$, where $\boldsymbol{\pi}(i) = \frac{d_i}{\sum_{u \in V} d_u} = \frac{d_i}{2m}$. This stationary distribution satisfies the properties of a probability distribution, namely, $\sum_{u \in V} \boldsymbol{\pi}(u) = 1$. Furthermore, it satisfies the equation $\boldsymbol{\pi}^T P = \boldsymbol{\pi}^T$.

In our analysis, we consider two types of Laplacian matrices: the combinatorial Laplacian $L = D - A$ and the normalized Laplacian $\mathcal{L} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. Both Laplacian matrices are singular matrices of rank $n - 1$. By employing eigen-decomposition, we can express L as $L = \sum_{i=1}^n \lambda_i u_i u_i^T$, where $\lambda_1 \leq \cdots \leq \lambda_n$, and $\mathcal{L}$ as $\mathcal{L} = \sum_{i=1}^n \sigma_i \tilde{u}_i \tilde{u}_i^T$, where $\sigma_1 \leq \cdots \leq \sigma_n$. It is important to note that both matrices have $\lambda_1 = \sigma_1 = 0$. Consequently, the inverses of the Laplacian matrices do not exist, and instead, we can alternatively use Laplacian pseudo-inverses. Among various Laplacian pseudo-inverses, the Moore-Penrose pseudo-inverse is of significant importance, as it is unique for a given Laplacian matrix. Specifically, it is defined as $L^\dagger = \sum_{i=2}^n \frac{1}{\lambda_i} u_i u_i^T$ for L, and $\mathcal{L}^\dagger = \sum_{i=2}^n \frac{1}{\sigma_i} \tilde{u}_i \tilde{u}_i^T$ for $\mathcal{L}$. For convenience, in the rest of this paper, we refer to the Moore-Penrose pseudo-inverse of Laplacian as Laplacian pseudo-inverse. The Laplacian pseudo-inverses find wide applications. In this paper, we mainly focus on two applications: electrical closeness centrality (ECC) computation and Kemeny constant (KC) computation, while other Laplacian pseudo-inverse related quantities (e.g., Kirchhoff index [68], normalized random walk betweenness centrality [44]) can also be directly approximated by the proposed techniques (See discussions at the end of Section 5).

Electrical closeness centrality (ECC). Resistance distance measures the distance between nodes in a network by treating it as an electrical system. The resistance distance between node s and t is defined as $r(s, t) = (L^\dagger)_{ss} + (L^\dagger)_{tt} - 2(L^\dagger)_{st}$. It is a distance metric on graphs. Compared to the shortest path distance, resistance distance takes into account all paths between nodes, making it more robust. The electrical closeness centrality $\mathbf{c}(u)$, then, is defined as $\frac{n-1}{\sum_{v \in V} r(u,v)} = \frac{n-1}{Tr(L^\dagger) + n(L^\dagger)_{uu}}$ [16]. The term $Tr(L^\dagger)$ is the same for all nodes $u \in V$. Consequently, the larger the value of $(L^\dagger)uu$, the smaller the value of $c(u)$. Therefore, computing the diagonal elements $(L^\dagger)_{uu}$ is a key problem in determining ECC.

Kemeny constant (KC). The hitting time $h(s, u)$ between node s and u is the expected length for a random walk starts from s to hit u. Kemeny constant, denoted as $\kappa(G)$, is then defined as $\kappa(G) = \sum_{u \in V} h(s, u)\boldsymbol{\pi}(u)$. In other words, Kemeny constant represents the expected length of a random walk from a source node to a randomly-chosen target node in a graph [10]. Thus, a smaller Kemeny constant implies closer connectivity between nodes in the graph. It is also well-known that Kemeny constant is a constant for every source node s and can be computed as $Tr(\mathcal{L}^\dagger) = \sum_{i=2}^n \frac{1}{\sigma_i} = \sum_{i=2}^n \frac{1}{1-\lambda_i}$ [37]. In spectral graph theory, it is well-known that the connectivity of a graph can be assessed by examining σ_2 [8]. A lower value of $\frac{1}{\sigma_2}$ typically indicates stronger connectivity. Importantly, while σ_2 provides initial insights, the higher-order eigenvalues are also indicative of connectivity aspects [13]. The Kemeny constant, by incorporating these higher-order eigenvalues, offers a more comprehensive understanding of a graph's connectivity than σ_2 alone.
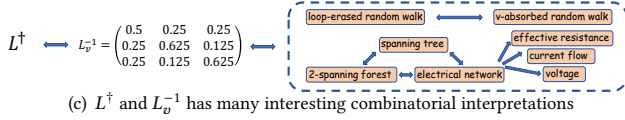
3 NEW THEORETICAL RESULTS

In this section, we propose a series of new theoretical results. An illustration of our computation framework is shown in Fig. 1. First, we give new formulas of $L^\dagger$ and $\mathcal{L}^\dagger$ by relating them to L_v^{-1}. The
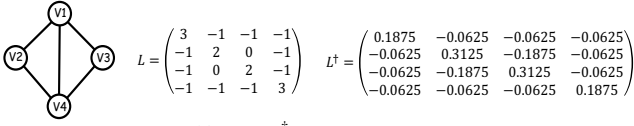
(a) L and $L^\dagger$ of an example graph

$$L^\dagger = \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T \\ -\frac{1}{n}\vec{1}^T \end{pmatrix} L_v^{-1} \begin{pmatrix} I - \frac{1}{n}\vec{1} & I - \frac{1}{n}\vec{1} \end{pmatrix} = \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T \\ -\frac{1}{n}\vec{1}^T \end{pmatrix} \begin{pmatrix} 0.5 & 0.25 & 0.25 \\ 0.25 & 0.625 & 0.125 \\ 0.25 & 0.125 & 0.625 \end{pmatrix} \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T & I - \frac{1}{n}\vec{1} \end{pmatrix}$$

(b) The new formula of $L^\dagger$ in terms of L_v^{-1}

$$L^\dagger \quad\longleftrightarrow\quad L_v^{-1} = \begin{pmatrix} 0.5 & 0.25 & 0.25 \\ 0.25 & 0.625 & 0.125 \\ 0.25 & 0.125 & 0.625 \end{pmatrix} \longleftrightarrow$$

(c) $L^\dagger$ and L_v^{-1} has many interesting combinatorial interpretations

Figure 1: An illustration of our new theoretical results

results show that in order to compute $L^\dagger$ and $\mathcal{L}^\dagger$, it suffices to compute L_v^{-1}. Thus, we focus on studying L_v^{-1}. We review three existing theoretical interpretations of L_v^{-1} in terms of v-absorbed walk and electrical network. However, it is very costly to sample v-absorbed walks and compute an electrical system. To overcome this challenge, we propose two novel combinatorial interpretations of L_v^{-1} in terms of spanning trees and loop-erased walks, which are easier to sample. The new point view of L_v^{-1} is utilized to design efficient algorithms for approximating ECC and KC in Section 4 and Section 5, respectively.

3.1 Relating Laplacian pseudo-inverses to L_v^{-1}

Since the inverse of L does not exist, the analysis of a Moore-Penrose pseudo-inverse is much harder than a normal matrix inverse. Previous studies are often based on the formula $L^\dagger = (L + \frac{\vec{1}\vec{1}^T}{n})^{-1} - \frac{\vec{1}\vec{1}^T}{n}$ [16]. However, $L + \frac{\vec{1}\vec{1}^T}{n}$ becomes a dense matrix that it is hard to obtain its inverse, and there is no obvious physical meaning of $(L + \frac{\vec{1}\vec{1}^T}{n})^{-1}$. To address this issue, we consider another important matrix L_v to study the properties of the Laplacian pseudo-inverse. Specifically, L_v is a submatrix of L obtained by removing the v-th row and column from L, which preserves the sparsity of L. It is important to note that unlike L, the inverse of L_v exists. Moreover, we find that L_v^{-1} has several interesting combinatorial interpretations, and $L^\dagger$ can also be formulated by L_v^{-1}. Since $L^\dagger$ is independent of the choice of v, we can heuristically choose v as an easy-to-hit landmark node (e.g., the highest-degree node) to accelerate the computation of $L^\dagger$. Due to the space limits, all the missing proofs can be found in the appendix [5].

THEOREM 3.1. *Let $(L^\dagger)_v$ denote the submatrix of $L^\dagger$ obtained by deleting the v-th row and column of $L^\dagger$. Let $\vec{1}$ be an $(n-1) \times 1$ all-one matrix, we have:* $L^\dagger = \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T \\ -\frac{1}{n}\vec{1}^T \end{pmatrix} L_v^{-1} \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T & -\frac{1}{n}\vec{1} \end{pmatrix}$.

Based on Theorem 3.1, we can further obtain the element-wise representation of $L^\dagger$. There are four situations:

$$(L^\dagger)_{st} = \mathbf{e}_s^T L_v^{-1}\mathbf{e}_t - \frac{1}{n}\mathbf{e}_s^T L_v^{-1}\vec{1} - \frac{1}{n}\mathbf{e}_t^T L_v^{-1}\vec{1} + \frac{1}{n^2}\vec{1}^T L_v^{-1}\vec{1}, \quad s, t \neq v \tag{1}$$

$$(L^\dagger)_{ss} = \mathbf{e}_s^T L_v^{-1}\mathbf{e}_s - 2\frac{1}{n}\mathbf{e}_s^T L_v^{-1}\vec{1} + \frac{1}{n^2}\vec{1}^T L_v^{-1}\vec{1}, \quad s, t \neq v \tag{2}$$

$$(L^\dagger)_{sv} = -\frac{1}{n}\mathbf{e}_s^T L_v^{-1}\vec{1} + \frac{1}{n^2}\vec{1}^T L_v^{-1}\vec{1}, \quad s \neq v \tag{3}$$

$$(L^\dagger)_{vv} = \frac{1}{n^2}\vec{1}^T L_v^{-1}\vec{1}. \tag{4}$$

These formulas show that we can represent all elements of $L^\dagger$ by L_v^{-1}. It is important to note that the node v can be chosen arbitrarily, while the elements of $L^\dagger$ maintain as invariants. Thus, in practice, we can select a "good" node to speed up computation.

Next, we can also represent $\mathcal{L}^\dagger$ in terms of L_v^{-1}. The normalized Laplacian $\mathcal{L}$ is only different from L in terms of degree normalized terms. Thus, a similar theorem can also be obtained for $\mathcal{L}^\dagger$.

THEOREM 3.2. *Let $\boldsymbol{\pi}_v$ be an $(n-1) \times 1$ vector obtained by deleting the v-th element of the stationary distribution vector $\boldsymbol{\pi}$, $\sqrt{\boldsymbol{\pi}_v}$ is the vector obtained by applying square root on each elements of $\boldsymbol{\pi}_v$, $\sqrt{\boldsymbol{\pi}_v}(u) = \sqrt{\boldsymbol{\pi}_v(u)}$. $\mathcal{L}_v$ is the submatrix of $\mathcal{L}$ obtained by deleting the v-th row and column. We have:*

$$\mathcal{L}^\dagger = \begin{pmatrix} I - \sqrt{\boldsymbol{\pi}_v}\sqrt{\boldsymbol{\pi}_v}^T \\ -\sqrt{\boldsymbol{\pi}(v)}\sqrt{\boldsymbol{\pi}_v}^T \end{pmatrix} \mathcal{L}_v^{-1} \begin{pmatrix} I - \sqrt{\boldsymbol{\pi}_v}\sqrt{\boldsymbol{\pi}_v}^T & -\sqrt{\boldsymbol{\pi}(v)}\sqrt{\boldsymbol{\pi}_v} \end{pmatrix}.$$

Suppose $\vec{\mathbf{d}}_v$ is the degree vector that deleting the v-th element and $\vec{\mathbf{d}}_v(u) = d_u$, since $\mathcal{L}_v^{-1} = D_v^{\frac{1}{2}} L_v^{-1} D_v^{\frac{1}{2}}$, we can further give the element-wise formula for $\mathcal{L}^\dagger$ in terms of L_v^{-1}:

$$(\mathcal{L}^\dagger)_{st} = \sqrt{\boldsymbol{\pi}(s)\boldsymbol{\pi}(t)}(2m\mathbf{e}_s^T L_v^{-1}\mathbf{e}_t - \mathbf{e}_s^T L_v^{-1}\vec{\mathbf{d}}_v - \mathbf{e}_t^T L_v^{-1}\vec{\mathbf{d}}_v + \frac{1}{2m}\vec{\mathbf{d}}_v^T L_v^{-1}\vec{\mathbf{d}}_v), \quad s, t \neq v \tag{5}$$

$$(\mathcal{L}^\dagger)_{ss} = \boldsymbol{\pi}(s)(2m\mathbf{e}_s^T L_v^{-1}\mathbf{e}_s - 2\mathbf{e}_s^T L_v^{-1}\vec{\mathbf{d}}_v + \frac{1}{2m}\vec{\mathbf{d}}_v^T L_v^{-1}\vec{\mathbf{d}}_v), \quad s, t \neq v \tag{6}$$

$$(\mathcal{L}^\dagger)_{sv} = \sqrt{\boldsymbol{\pi}(s)\boldsymbol{\pi}(v)}(-\mathbf{e}_s^T L_v^{-1}\vec{\mathbf{d}}_v + \frac{1}{2m}\vec{\mathbf{d}}_v^T L_v^{-1}\vec{\mathbf{d}}_v), \quad s \neq v \tag{7}$$

$$(\mathcal{L}^\dagger)_{vv} = \boldsymbol{\pi}(v)(\frac{1}{2m}\vec{\mathbf{d}}_v^T L_v^{-1}\vec{\mathbf{d}}_v). \tag{8}$$

Discussion. Theorem 3.1 and Theorem 3.2 show that any functions of Laplacian pseudo-inverse, such as ECC and KC can be expressed by elements of L_v^{-1}. Furthermore, they remain as invariant no matter how we choose the landmark node v. To our knowledge, such results have never been proposed in previous studies. Compared to Theorem 3.1 in [39], their result only represent effective resistance in terms of L_v^{-1}, which is a combination of four elements of $L^\dagger$, while we express both $L^\dagger$ and $\mathcal{L}^\dagger$ in terms of L_v^{-1}. Thus, the results in [39] are just special cases of our results. Compared to [11, 12, 15], our specific formula offers a clearer characterization of Laplacian pseudo-inverses: elements of Laplacian pseudo-inverse, in terms of elements of Laplacian sub-matrix. Laplacian pseudo-inverses remain invariants no matter how we choose v. Although our results can also be derived from the more high-level pseudo-inverse formulas proposed in [11, 12, 15], these studies primarily focus on other topics rather than fast sampling algorithms. Moreover, none of them provide element-wise formulas, let alone combinatorial interpretations of the Laplacian pseudo-inverses. Next, we discuss the theoretical interpretations of L_v^{-1}, which implies that both $L^\dagger$ and $\mathcal{L}^\dagger$ can also have similar interesting explanations.

3.2 Existing interpretations of L_v^{-1}

We review three existing interpretations of L_v^{-1} in terms of v-absorbed walk, 2-spanning forest and electrical network. Since $L^\dagger$ and $\mathcal{L}^\dagger$ can be formulated by L_v^{-1}, they can also be interpreted by these objects. First, v-absorbed walk is a type of random walk that stops when hitting the node v. L_v^{-1} can be intuitively interpreted by v-absorbed walks.

LEMMA 3.3. ("v-absorbed walk" interpretation of L_v^{-1} [39]) Let $\tau_v[s,u]$ denote the expected number of walk steps that passes u in a v-absorbed walk starts from s, $\tilde{\tau}_v[s,u] = \frac{\tau_v[s,u]}{d_u}$. We have: $(L_v^{-1})_{su} = \tilde{\tau}_v[s,u]$.

The v-absorbed random walk interpretation of L_v^{-1} is very useful for computing resistance distance [39]. For resistance distance $r(s,t)$, sampling two v-absorbed walks separately from s and t consumes $(h(s,v) + h(t,v))$ time, where $h(s,v)$ is the hitting time from s to v. It is fast for real-life large graphs since there often exists a node v (e.g. the highest-degree node) that is easy to hit so that $h(s,v)$ and $h(t,v)$ are small [39]. However, for $L^\dagger$ and $\mathcal{L}^\dagger$, we need to compute $\vec{1}^T L_v^{-1} \vec{1}$ (or $\vec{d}_v^T L_v^{-1} \vec{d}_v$), which requires the value of all elements of L_v^{-1}. In this case, we need to sample v-absorbed walks from all nodes, which takes $\sum_{u \in V} h(u,v)$ for each sample. Clearly, the time cost is expensive for large graphs even when we choose v as an easy-to-hit node.

A spanning tree is a connected subgraph of G with no cycle. Similarly, a 2-spanning forest is a subgraph of G with no cycle that has 2 tree components. Let $|\cdot|$ denote the cardinality of a set. According to the all-minors matrix-forest theorem [19], we can express the elements of L_v^{-1} by the number of 2-spanning forests.

LEMMA 3.4. ("2-spanning forest" interpretation of L_v^{-1} [19]) Let $\mathcal{T}$ denote the set of spanning trees. Let $\mathcal{F}_{v|s,u}$ denote the set of 2-spanning forests that v is in one tree component, s and u are both in the other tree component. We have: $(L_v^{-1})_{su} = \frac{|\mathcal{F}_{v|s,u}|}{|\mathcal{T}|}$.

According to the 2-spanning forest interpretation, a direct method for approximating Laplacian pseudo-inverses is to count the number of 2-spanning forests by sampling. However, sampling a 2-spanning forest is a hard problem and no efficient algorithm for sampling 2-spanning forests is known [6, 48].

Additionally, L_v^{-1} is also related to electrical networks. When considering the graph as an electrical network, where each node is a junction and each edge is a unit resistor, based on the classical physical laws, when sending flows into the electrical network, the voltages, currents, effective resistances as well as the energy of the electrical system can all be represented by L_v^{-1} [13, 57].

LEMMA 3.5. ("electrical network" interpretation of L_v^{-1} [13]) Consider an electrical network that unit flow comes in through node s, and unit flow comes out through node v. Then $(L_v^{-1})_{su}$ is the voltage at node $u \in V$. Specifically, $(L_v^{-1})_{ss}$ is the effective resistance between node s and v. According to the Ohm's law, $(L_v^{-1})_{se_1} - (L_v^{-1})_{se_2}$ is the current flow thourgh edge (e_1, e_2). As a by-product, the energy of the electrical network is also $(L_v^{-1})_{ss}$.

Computing the corresponding quantities in an electrical system requires solving a linear system [57], which is costly when applied for large graphs. In practice, all diagonal elements of L_v^{-1} are often required. This usually involves solving n linear systems, which is much more expensive.

3.3 Spanning tree interpretation of L_v^{-1}

There exist many studies that relate L_v^{-1} to spanning trees. However, it is still unknown whether we can represent arbitrary elements of L_v^{-1} in terms of spanning trees. We overcome this problem by considering paths in spanning trees.

LEMMA 3.6. ("spanning tree" interpretation of L_v^{-1}) Let $\mathcal{T}$ denote the set of spanning trees. Let $\mathcal{T}_{e_1,e_2}^{s,v}$ denote the set of spanning trees where the unique path from s to v passes edge $e = (e_1, e_2)$. Let $\mathcal{P}^{u,v}$

be an arbitrary path from u to v in graph G. Then we have: $(L_v^{-1})_{su} = \sum_{(e_1,e_2) \in \mathcal{P}^{u,v}} \frac{|\mathcal{T}_{e_1,e_2}^{s,v}| - |\mathcal{T}_{e_2,e_1}^{s,v}|}{|\mathcal{T}|}$.

Lemma 3.6 represents L_v^{-1} in terms of the number of spanning trees. In [39], similar results were proposed for effective resistance, which is the diagonal elements of L_v^{-1}. Our result connect L_v^{-1} to voltages and send current flows on spanning trees, thus can interpret all elements of L_v^{-1}, which is negligible in previous studies. To our knowledge, this is the first time that matrix tree theorems for an arbitrary element of L_v^{-1} is given. Based on Lemma 3.6, we can sample a number of spanning trees, examine whether the path from s to v passes through the edges $(e_1, e_2) \in \mathcal{P}^{u,v}$, then we can build an unbiased estimator of $(L_v^{-1})_{sv}$. For an estimator of $L^\dagger$, we can further combine the number of spanning trees to estimate the elements of L_v^{-1} to build an unbiased estimator of $L^\dagger$. In such cases, we are not only interesting in a single element of L_v^{-1}. By combining the results of Lemma 3.5 and Lemma 3.6, we can obtain the following corollary:

COROLLARY 1. Suppose that $\vec{f} \in \mathbf{R}^m$ is the flow on a network that satisfying Kirchhoff's current law and Kirchhoff's voltage law, $\vec{b}$ is the demand of the flow ($\vec{b}(u)$ represents the flow comes out through node u) that satisfying $\vec{1}^T \vec{b} = 0$. Let $\vec{f}_T$ denote a flow on a spanning tree T with the same demand $\vec{b}$, we have: $\vec{f} = \sum_{T \in \mathcal{T}} \frac{1}{|\mathcal{T}|} \vec{f}_T$.

The corollary indicates that if we can represent the desired quantities (e.g. $L^\dagger$) as voltages or flows in an electrical system, then we can sample a number of spanning trees and sending flows on them. By performing some operations on the spanning trees, we can obtain an estimation of the electrical system, as well as the desired quantities. These ideas will be utilized in our algorithms in Section 5.

For real-world graphs, the Wilson algorithm [62] is the best algorithm for sampling a spanning tree. The time complexity of the Wilson algorithm is $Tr((I - P_v)^{-1})$ [43]. For real-life graphs, we can derive an upper bound for the time complexity of the Wilson algorithm, since $Tr((I - P_v)^{-1}) = \sum_{u \in V} \tau_v[u,u] = \sum_{u \in V} r(u,v)d_u \leq m\Delta_G$, where Δ_G is the diameter of the graph. The last inequality holds because effective resistance between u and v is always smaller than the shortest path distance between u and v [8], which can further be bounded by Δ_G. Real-life graphs often have the "small world" property, charaterized by a small diameter. Thus, the algorithm is very efficient in real-life graphs.

3.4 Loop-erased walk interpretation of L_v^{-1}

Loop-erased walk is a type of random walk where we obtain a loop-erased trajectory by erasing all the loops in the random walk trajectory. The Wilson algorithm [62] is a famous algorithm for sampling a spanning tree that utilizing the loop-erased walk. Given a graph G and a root node v, the Wilson algorithm maintains its loop-erased trajectory to construct a spanning tree $\mathcal{T}$. It starts with only the root node v in $\mathcal{T}$, $\mathcal{T} = \{v\}$. By fixing an arbitrary ordering of $V \setminus \{v\}$, random walks are simulated from a node $u \notin \mathcal{T}$ (following the ordering) until it hits $\mathcal{T}$. Each time a random walk stops, the loop-erased trajectory of the random walk is added into $\mathcal{T}$. This process terminates when all nodes of G are added into $\mathcal{T}$. In this paper, we denote such a process by LE_v, which is a complete execution of a loop-erased walk with root node v. It is well-known that $\mathcal{T}$ is a uniformly sampled spanning tree independent of the node ordering we fixed [62].

The basic proof technique in [62] to show why the Wilson algorithm produces a uniform spanning tree is a stack representation of LE_v. Wilson et al. [62] proved that the following process is identical
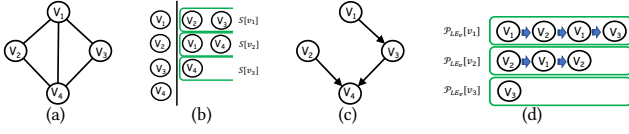
Figure 2: A loop-erased walk LE_v **with** $v = v_4$ **and its stack representation. (a) an example graph; (b) the stack representation of** LE_v. **When operating the stacks by the ordering** v_1, v_2, v_3, **the loop-erased walk trajectory is** $v_1, v_2, v_1, v_3, v_4, v_2, v_4$. **When the ordering is** v_2, v_3, v_1, **the trajectory is** $v_2, v_1, v_2, v_4, v_3, v_4, v_1, v_3$; **(c) The resulting spanning tree is independent of the node ordering; (d) the loop-included path** $\mathcal{P}_{\mathsf{LE}_v}[u]$ **for** $u = v_1, v_2, v_3$.

to LE_v: Given a graph G and a node v, in the stack representation of LE_v, each node $u \in V$ maintains a stack $S[u]$. All the stacks are initialized empty and marked white. Then, we operate on the stacks following an ordering of $V \setminus \{v\}$ which is fixed before. Specifically, $S[v]$ is marked gray first. When a stack $S[u]$ is met, we (i) randomly select a node w from the neighbors of u, and (ii) push w on the top of $S[u]$. Finally, we (iii) switch to the stack $S[w]$. The operation continues until we switch to a stack that is marked gray. Each time there is a loop on top of the stacks, the loop will be popped. When a stack marked gray is met, we mark all stacks that are not empty as gray and continue the operations from a stack $S[u]$ that has not been marked gray. This process terminates until all nodes are marked gray. Wilson proved that the resulting stacks are independent of the node ordering we fixed. Denote $\mathrm{Top}(S[u])$ the top element of $S[u]$, then the edges $(u, \mathrm{Top}(S[u]))$ for $u \in V$ and $u \neq v$ form a uniformly sampled spanning tree with root v.

In this paper, we go one step further. When there is a loop on top of the stacks, instead of popping the nodes of the loop out of the stacks, we keep them in the stack. We discover that such a stack representation of LE_v captures all information of L_v^{-1}. Fig. 2 illustrates a running example. Fig. 2(a) is an example graph and Fig. 2(b) is the stack representation of a sample of LE_v. It can be seen that the stack representation is independent of the fixed node ordering. When the fixed node ordering of $V \setminus \{v_4\}$ is v_1, v_2, v_3, the loop-erased walk trajectory (the stack process ordering) is $v_1, v_2, v_1, v_3, v_4, v_2, v_4$. When the ordering is v_2, v_3, v_1, the trajectory is $v_2, v_1, v_2, v_4, v_3, v_4, v_1, v_3$. Although the ordering is different, we can obtain a same stack representation of these loop-erased walks. The resulting spanning tree is also the same, as illustrated in Fig. 2(c). This implies that when we have sampled a loop-erased walk LE_v, we can obtain $n!$ samples from the stack representation of LE_v by using different node orderings to traverse the stack representation of LE_v.

We have seen above that the spanning tree maintained at the top of the stacks can interpret L_v^{-1}. Here, we can exploit more information from the stacks. Specifically, we define a loop-included path $\mathcal{P}_{\mathsf{LE}_v}[s]$ from the stack representation of LE_v, which is constructed as follows: The start node of $\mathcal{P}_{\mathsf{LE}_v}[s]$ is $u = s$, following the stack representation of LE_v, u jumps to the node at the bottom of $S[u]$, until the root node v is reached. Each time the path leaves a node u, the node at the bottom of $S[u]$ is removed from $S[u]$. As illustrated in Fig. 2(d), there is a sample of $\mathcal{P}_{\mathsf{LE}_v}[u]$ from all nodes $u = v_1, v_2, v_3$. We state that the defined loop-included path $\mathcal{P}_{\mathsf{LE}_v}$ can interpret arbitrary element of L_v^{-1}.

LEMMA 3.7. *("Loop-erased walk" interpretation of L_v^{-1}) Given a loop-erased walk* LE_v, *let* $\tau_{\mathsf{LE}_v}[s, u]$ *be the expected number of existences of node u in* $\mathcal{P}_{\mathsf{LE}_v}[u]$, $\tilde{\tau}_{\mathsf{LE}_v}[s, u] = \frac{\tau_{\mathsf{LE}_v}[s,u]}{d_u}$. *We have:* $(L_v^{-1})_{su} = \tilde{\tau}_{\mathsf{LE}_v}[s, u]$. *Specifically, let* $\tilde{\tau}_{\mathsf{LE}_v}[s, s]$ *be the normalized*

expected number of existences of node s in LE_v, we have: $(L_v^{-1})_{ss} = \tilde{\tau}_{\mathsf{LE}_v}[s, s]$.

Note that previous study [39] only considers the expected number of passes to a node in the loop-erased walk sampling process. Compared to Lemma 5.1 in [39], our result is based on the novel idea of designing a way of traversing the stack representation, thus extending the interpretations of L_v^{-1} in terms of loop-erased walks to non-diagonal elements. Notice that after storing the stack representation as the index, we can directly obtain random walk samples from each node to v by traversing the stacks. Since reading the elements from the cache is faster than sampling a neighbor (especially in complex networks such as weighted graphs), this approach holds potential for significant computational savings and increased efficiency in handling complex network structures.

Lemma 3.7 suggests that the distribution of $\mathcal{P}_{\mathsf{LE}_v}[u]$ is the same as an v-absorbed walk trajectory from u. Thus, the expected length of $\mathcal{P}_{\mathsf{LE}_v}[u]$ is the hitting time $h(u, v)$, the expected number of nodes in the stacks is $Tr((I - P_v)^{-1})$. The advantage of the "loop-erased walk" interpretation for sampling is apparently. In order to approximate all elements of L_v^{-1}, we need to sample v-absorbed walks from all nodes $u \in V$, which consumes $\sum_{u \in V} h(u, v) = \vec{1}^T (I - P_v)^{-1} \vec{1}$ time. For loop-erased walk sampling, we only need to sample one loop-erased walk with node v in expected running time $Tr((I - P_v)^{-1})$, which is much lower than $\vec{1}^T (I - P_v)^{-1} \vec{1}$.

Similar to spanning tree sampling, the loop-erased walk sampling also requires a landmark node v as input. When v is an easy-to-hit node, $Tr((I - P_v)^{-1})$ is small, which makes the sampling techniques more efficient. For convenience, we set v as the highest-degree node in graph. We observe that such a choice is near optimal for real-life graphs, as evidenced in our experiments (see Section 6.2).

4 ALGORITHMS FOR ECC COMPUTATION

In this section, we study the problem of approximating electrical closeness centrality (ECC). Recall that in order to compute ECC, it suffices to compute $(L^\dagger)_{uu}$ for $u \in V$. Then, based on the formula $\mathbf{c}(u) = \frac{n-1}{n(L^\dagger)_{uu} + Tr(L^\dagger)}$, the ECC $\mathbf{c}(u)$ for all $u \in V$ can be computed in linear time if we have known $(L^\dagger)_{uu}$ for all $u \in V$. According to Equation (1) and Equation (4), $(L^\dagger)_{uu} = (L_v^{-1})_{uu} - 2\mathbf{e}_u^T L_v^{-1} \vec{1} + \frac{1}{n^2} \vec{1}^T L_v^{-1} \vec{1}$ for $u \in V$ and $u \neq v$; and $(L^\dagger)_{vv} = \frac{1}{n^2} \vec{1}^T L_v^{-1} \vec{1}$. The computation of $(L^\dagger)_{uu}$ for $u \in V$ can be decomposed into two stages: (i) computation of $(L_v^{-1})_{uu}$ for all $u \in V$ and $u \neq v$; (ii) computation of $\mathbf{e}_u^T L_v^{-1} \vec{1}$ for $u \in V$ and $u \neq v$. Then, $\vec{1}^T L_v^{-1} \vec{1} = \sum_{u \in V, u \neq v} \mathbf{e}_u^T L_v^{-1} \vec{1}$, the whole diagonal of $L^\dagger$ can be obtained.

4.1 Existing solutions of ECC computation

As we discussed before, the problem of computing ECC is identical to computing $(L^\dagger)_{uu}$. A straightforward approach is to apply eigendecomposition on L which costs $O(n^3)$ time. Bozzo and Franceschet [16] computes $L^\dagger$ by the formula $L^\dagger = (L + \frac{\vec{1}\vec{1}^T}{n})^{-1} - \frac{\vec{1}\vec{1}^T}{n}$. However, the all-one matrix makes the first part a dense matrix, and it is hard to compute the inverse of a dense matrix, thus the method is not scalable to large real-life graphs. Recently, Angriman et al. approximate $(L^\dagger)_{uu}$ for $u \in V$ by spanning tree sampling [4]. They first solve a linear system $L\mathbf{x} = \mathbf{e}_s - \frac{\vec{1}}{n}$ to obtain the s-th column of $L^\dagger$. Then they use spanning tree sampling to approximate single-source resistance distance $r(s, u)$ for all $u \in V$. Finally, $(L^\dagger)_{uu}$ for $u \in V$ is obtained by $(L^\dagger)_{uu} = r(s, u) + 2(L^\dagger)_{su} - (L^\dagger)_{ss}$. This algorithm is the state-of-the-art algorithm to compute ECC. However, the cost of accurately solving a Laplacian system is still hard for large graphs. Compared to their work, we show that there is no need to solve

(a) G, network flow   (b) flows on spanning trees
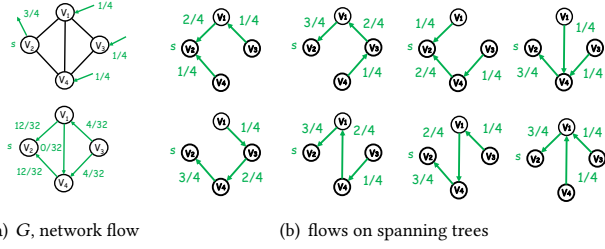
Figure 3: Illustration of approximating $L^\dagger$ by spanning trees. (a) A graph G and an electrical network flow where $\frac{1}{n}$ current flow comes in through every node $u \in V$ and a unit current flow comes out through v_2; (b) There are 8 spanning trees. The current flows along an edge (u,v) in G equals the average current flows through (u,v) on all spanning trees.

the linear system as it can also be approximated via spanning tree sampling. Our algorithm directly approximates the elements of $L^\dagger$, thus gives a "pure" spanning tree sampling algorithm. Moreover, our loop-erased walk sampling algorithm further improves the efficiency of ECC approximation.

4.2 A spanning tree sampling algorithm

In this subsection, we propose a novel Monte Carlo algorithm SpanTree for approximating ECC based on spanning tree sampling. First, according to the "electrical system" interpretation of L_v^{-1}, $(L_v^{-1})_{uu}$ for $u \in V$ and $u \neq v$ is the single source resistance distance $r(v,u)$ from node v to all nodes u in the graph. We can use the spanning tree sampling algorithms proposed in [4, 39] to approximate $(L_v^{-1})_{uu}$ for $u \in V$. However, it is not enough to approximate the diagonal elements of L_v^{-1} ($e_u^T L_v^{-1} \vec{1}$ for $u \in V$ and $u \neq v$ should also be approximated). To tackle this problem, we first interpret the v-th column of $L^\dagger$, the $(L^\dagger)_{vu}$ for $u \in V$ in terms of electrical voltages of a certain electrical system.

Interpreting $L^\dagger$ by electrical voltages. In Section 3.2, $(L_v^{-1})_{su}$ for $u \in V$ and $u \neq v$ is the voltage at node u in an electrical system where a unit current flow comes in through node s and a unit current flow comes out through node v. Based on Equation (3) and Equation (4), due to the fact that electrical systems are linear objects, we can combine n such electrical systems to interpret $(L^\dagger)_{vu}$ for $u \in V$:

Lemma 4.1. *Consider an electrical system that $\frac{1}{n}$ current flow comes in through every node $u \in V$, and $\frac{1}{n}$ current flow comes out through node v in graph G. Then, in such an electrical system, $-(L^\dagger)_{vu}$ is the voltage at node $u \in V$. According to the Ohm's law, $(L^\dagger)_{ve_2} - (L^\dagger)_{ve_1}$ is the current flow through the edge (e_1, e_2). As a by-product, the energy of the electrical network is also $(L^\dagger)_{vv}$.*

To better understand the result, we give an illustrative example. In Fig. 3(a), there is an electrical system where $\frac{1}{n}$ ($\frac{1}{4}$) current flow comes in through every node in G and 1 current flow comes out through v_2. The current flow on each edge is depicted in Fig. 3(a). As we can see, the energy of the system is $(\frac{12}{32})^2 + (\frac{12}{32})^2 + (\frac{4}{32})^2 + (\frac{4}{32})^2 = \frac{5}{16} = (L^\dagger)_{22}$; $-(L^\dagger)_{23}$ is the voltage at node v_3 which is $\frac{3}{16}$. Thus, $(L^\dagger)_{23} = -\frac{3}{16}$. We can obtain the voltages by solving a Laplacian linear system $L\mathbf{x} = e_v - \frac{\vec{1}}{n}$ [57]. However, deterministically solving such a system, even equipped with the fastest Laplacian solver [33], is still very costly. To overcome this problem, we design

an approximate algorithm for a column of $L^\dagger$ based on sampling spanning trees.

Approximate the electrical flow via sampling spanning trees. According to the "spanning tree" interpretation of L_v^{-1}, the current flows on graph G are the average of the current flows on all spanning trees of G. Based on Lemma 4.1, in order to compute the v-th column of $L^\dagger$, we need to send $\frac{1}{n}$ current flow in through every node $u \in V$, and 1 current flow out through node v in the graph. Here, we can also send such flows on all spanning trees. Then, the current flows on the original graph is expected to be the average of the flows on spanning trees. Fig. 3 illustrates an example. There are 8 spanning trees of G. In each spanning tree, we send $\frac{1}{4}$ current flow in through each node in G, and send 1 current flow out through v_2. The current flow on each spanning tree is depicted in Fig. 3(b). It is easy to verify that the current flows on G is exactly the average of the current flows on all spanning trees. This gives us an interesting Monte Carlo algorithm: we can sample a small number of spanning trees, send flows on these spanning trees and take the average. Then, we get an approximate current flow of the desired electrical system. As we have discussed, we can utilize the Wilson algorithm [62] to efficiently sample a spanning tree. Notice that computing the current flow on a spanning tree is relatively easy. We only need to traverse the paths from all nodes u to a node root v in the spanning tree, and add the current that injects node u to each edge in the path. After that, we can approximate $e_u^T L_v^{-1} \vec{1}$ for $u \in V$ and $u \neq v$ from $(L^\dagger)_{vu}$ for $u \in V$ based on Equation (3) and Equation (4). Thus, we give a "pure" spanning tree sampling algorithm for approximating $(L^\dagger)_{uu}$ for $u \in V$ by combining the two stages.

Algorithm details. Algorithm 1 illustrates the pseudo-code of our spanning tree sampling algorithm for approximating ECC. The algorithm takes a landmark node v as input, which is the "0" voltage node. We need to compute the electrical flows in the electrical system in Lemma 3.5 and Lemma 4.1. According to the Kirchhoff's law, in order to compute the voltage of node u, we should compute the current flow along a path from u to the "0" voltage node v. Thus, we fix a BFS tree with root v at first, which stores a path $\mathcal{P}_{\text{BFS}}[u]$ from each node u to the root v (Line 1). Then we sample a number of spanning trees using the Wilson algorithm (Line 4). The results of the spanning tree will be stored in a vector "next" where next$[u]$ records the out-edge of u in the spanning tree [62]. For each spanning tree, we first compute the current flow on that spanning tree. Specifically, we traverse all the paths from each node $u \in V$ to the root node v, add the current flows injects each node u on every edge in that path. Since every node is only the endpoint of one edge in a spanning tree, we denote ps$[u]$ the path support of edge $(u, \cdot)$, which is the total current through the edge out-edge of u in the spanning tree. This can be processed in at most $n\Delta_G$ time, where Δ_G is the diameter of graph G. This process is often very efficient in practice, as real-world graphs typically exhibit a small-world property [45]. The total path support ps$[e_1]$ on each edge $(e_1, \cdot)$ is the current flow on that edge (Line 5-7). Note that the current flow for estimating $(L_v^{-1})_{uu}$ for $u \in V$ (unit current flows in through u, unit current flows out through v) is 1 on edge edge, so we do not need to compute the path support. Then, we compute the voltage of all nodes by traversing the paths stored in $\mathcal{T}_{\text{BFS}}$. A question here is: how can we update all the voltages in linear time? To solve this problem, we perform a DFS traversal on the sampled spanning tree $\mathcal{T}$ with root v (Line 8). By computing the visit time "vis" and finish time "fin" in the DFS procedure, we can answer whether a node is the ancestor of another node in a tree in constant time [22]. Thus, we can also determine whether an edge is sampled in $\mathcal{T}$ in constant time. Next, for each node $u \in V$, we traverse the path $\mathcal{P}_{\mathcal{T}_{\text{BFS}}}[u]$ from u to v in the pre-computed BFS tree. For

Algorithm 1: SpanTree (ECC)

Input: Graph G, a landmark node v, sample size ω
Output: $(\widetilde{L^\dagger})_{uu}$ as an unbiased estimation of $(L^\dagger)_{uu}$ for $u \in V$

1   Pre-compute a BFS tree $\mathcal{T}_{\text{BFS}}$ with root v, $\mathcal{P}_{\mathcal{T}_{\text{BFS}}}[u]$ is the path from u to v in $\mathcal{T}_{\text{BFS}}$;
2   **for** $i = 1 : \omega$ **do**
3     $\mathbf{x}_1(u) \leftarrow 0, \mathbf{x}_2(u) \leftarrow 0$ for $u \in V$;
4   Uniformly sample a spanning tree $\mathcal{T}$ using the Wilson algorithm, $\mathcal{T}$ stores in a vector next where next$[u]$ records the endpoint of the edge $(u, \cdot)$ in $\mathcal{T}$, $\mathcal{P}_{\mathcal{T}}[u]$ is the path from u to v in $\mathcal{T}$;
5   **for** *each* $u \in V$ **do**
6   **for** *each edge* $e = (e_1, e_2) \in \mathcal{P}_{\mathcal{T}}[u]$ **do**
7         $\text{ps}[e_1] \leftarrow \text{ps}[e_1] + \frac{1}{n}$;   // Compute path support ps in $\mathcal{T}$;
8   Perform DFS on $\mathcal{T}$ from node v, compute the visit time $\text{vis}[u]$ and the finish time $\text{fin}[u]$ for all nodes $u \in V$;
9   **for** *each* $u \in V$ **do**
10   **for** *each edge* $e = (e_1, e_2) \in \mathcal{P}_{\mathcal{T}_{\text{BFS}}}[u]$ **do**
11   **if** $\text{next}[e_1] = e_2$ *and* $[\text{vis}[e_1], \text{fin}[e_1]] \subseteq [\text{vis}[u], \text{fin}[u]]$ **then**
12         $\mathbf{x}_1(u) \leftarrow \mathbf{x}_1(u) + \frac{1}{\omega}, \mathbf{x}_2(u) \leftarrow \mathbf{x}_2(u) + \frac{\text{ps}[e_1]}{\omega}$;
13   **if** $\text{next}[e_2] = e_1$ *and* $[\text{vis}[e_2], \text{fin}[e_2]] \subseteq [\text{vis}[u], \text{fin}[u]]$ **then**
14         $\mathbf{x}_1(u) \leftarrow \mathbf{x}_1(u) - \frac{1}{\omega}, \mathbf{x}_2(u) \leftarrow \mathbf{x}_2(u) - \frac{\text{ps}[e_2]}{\omega}$;

15   **for** *each* $u \in V$ **do**
16     $(\widetilde{L^\dagger})_{uu} \leftarrow \mathbf{x}_1(u) - 2\mathbf{x}_2(u) + \frac{1}{n} \sum_{u \in V} \mathbf{x}_2(u)$;
17   **return** $(\widetilde{L^\dagger})_{uu}$ for $u \in V$;

Algorithm 2: LEWalk (ECC)

Input: Graph G, a landmark node v, sample size ω
Output: $(\widetilde{L^\dagger})_{uu}$ as an unbiased estimation of $(L^\dagger)_{uu}$ for $u \in V$

1   Fix an arbitrary ordering $(v_1, \cdots, v_{n-1})$ of $V \setminus \{v\}$;
2   $\mathbf{y}_1(u) \leftarrow 0, \mathbf{y}_2(u) \leftarrow 0$ for $u \in V$;
3   **for** $i = 1 : \omega$ **do**
4     $\text{InTree}[u] \leftarrow \text{false}, \text{next}[u] \leftarrow -1$ for $u \in V$;
5     $\text{InTree}[v] \leftarrow \text{true}$;
6   **for** $j = 1 : n$ **do**
7       $u \leftarrow v_j$;
8   **while** $!\text{InTree}[u]$ **do**
9         $\mathbf{y}_1(u) \leftarrow \mathbf{y}_1(u) + \frac{1}{\omega d_u}$;
10        $\text{next}[u] \leftarrow \text{RandomNeighbor}(u)$;
11        $u \leftarrow \text{next}[u]$;
12       $u \leftarrow v_i$;
13   **while** $!\text{InTree}[u]$ **do**
14         $\text{InTree}[u] \leftarrow \text{true}, u \leftarrow \text{next}[u]$;

15   **for** $i = 1 : \omega$ **do**
16   Uniformly sample a node $s \in V$;
17     $u \leftarrow s$;
18   **while** $u \mathrel{!=} v$ **do**
19       $\mathbf{y}_2(u) \leftarrow \mathbf{y}_2(u) + \frac{1}{\omega d_u}, u \leftarrow \text{RandomNeighbor}(u)$;

20   **for** *each* $u \in V$ **do**
21     $(\widetilde{L^\dagger})_{uu} \leftarrow \mathbf{y}_1(u) - 2\mathbf{y}_2(u) + \frac{1}{n} \|\mathbf{y}_2\|_1$;
22   **return** $(\widetilde{L^\dagger})_{uu}$ for $u \in V$;

---

each edge (e_1, e_2) in the path, we examine whether the spanning tree path from u to v by comparing the visit time and finish time. If $[\text{vis}[e_1], \text{fin}[e_1]] \subseteq [\text{vis}[u], \text{fin}[u]]$, then e_1 is the ancestor of u, thus the spanning tree path from u to v contains the edge (e_1, e_2). We update the two voltages $\mathbf{x}_1(u)$ and $\mathbf{x}_2(u)$ accordingly (Line 9-14), where $\mathbf{x}_1(u)$ is the voltage of node u in an electrical network that unit current comes in thorough u and comes out through v, we have $E[\mathbf{x}_1(u)] = (L_v^{-1})_{uu}$; $\mathbf{x}_2(u)$ is the voltage on node u in an electrical network where $\frac{1}{n}$ current comes in through each node $u \in V$ and comes out through node v. We have $E[\mathbf{x}_2(u)] = \frac{1}{n}\mathbf{e}_u^T L_v^{-1}\vec{1}$. Finally, we approximate $(L^\dagger)_{uu}$ for $u \in v$ by the formula $(\widetilde{L^\dagger})_{uu} = \mathbf{x}_1(u) - 2\mathbf{x}_2(u) + \frac{1}{n}\sum_{u \in V}\mathbf{x}_2(u)$ (Line 15-16). According to Equation (3) and Equation (4), $(\widetilde{L^\dagger})_{uu}$ is an unbiased estimator of $(L^\dagger)_{uu}$.

Algorithm analysis. We first analyze the time complexity of SpanTree.

Lemma 4.2. *Let Δ_G denote the diameter of graph G. The time complexity of SpanTree for computing ECC is $O(Tr((I - P_v)^{-1}) + n\Delta_G)$ per sample, with an $O(m + n)$ pre-processing cost.*

Next, we show that SpanTree achieves an absolute error with a probability at least $1 - p_f$, where p_f is a small failure probability.

Theorem 4.3. *If the sample size $\omega \geq \frac{32\Delta_G^2 \log(\frac{2n}{p_f})}{\epsilon^2}$, SpanTree can output $(\widetilde{L^\dagger})_{uu}$ that satisfies $|(\widetilde{L^\dagger})_{uu} - (L^\dagger)_{uu}| \leq \epsilon$ for all $u \in V$ with a probability at least $1 - p_f$.*

Notice that the time complexity of any spanning tree sampling algorithms in [4, 39] cannot avoid such a Δ_G^2 term. However, in real-life graphs, Δ_G is often very small (e.g., $O(\log n)$ due to the small-world property [35]). Thus, the algorithm is very efficient on real-life graphs.

4.3 A loop-erased walk sampling algorithm

In this subsection, we propose a novel algorithm LEWalk for approximating ECC based on loop-erased walk sampling. According to Lemma 3.7, $(L_v^{-1})_{uu}$ for $u \in V$ and $u \neq v$ is the expected normalized number of passes to u in a loop-erased walk LE_v. We also need

to approximate $\mathbf{e}_u^T L_v^{-1}\vec{1}$ for $u \in V$ and $u \neq v$. We first state that $\mathbf{e}_u^T L_v^{-1}\vec{1} = \sum_{i \in V, i \neq v}(L_v^{-1})_{ui}$ is the expected normalized length of the loop-included path $\mathcal{P}_{\text{LE}_v}[u]$:

Lemma 4.4. *Let $|\mathcal{P}_{\text{LE}_v}[u]|_n$ denote the normalized length of the loop-included path $\mathcal{P}_{\text{LE}_v}[u]$, defined as the sum of $\frac{1}{d_u}$ for each node u the path has passed (a node may exist for many times), then we have: $E[|\mathcal{P}_{\text{LE}_v}[u]|_n] = \mathbf{e}_u^T L_v^{-1}\vec{1}$.*

As a result, we can obtain an estimation of $\mathbf{e}_u^T L_v^{-1}\vec{1}$, $u \in V$ by traversing all the loop-included paths $\mathcal{P}_{\text{LE}_v}[u]$ and compute the normalized length $|\mathcal{P}_{\text{LE}_v}[u]|_n$ for an estimation of $\mathbf{e}_u^T L_v^{-1}\vec{1}$ for all $u \in V$. Recall that after we have sampled a loop-erased walk LE_v, we can traverse a loop-included path $\mathcal{P}_{\text{LE}_v}[u]$ in time $h(s, v) = \mathbf{e}_s^T L_v^{-1}\vec{\mathbf{d}}_v$ if we store the stack representation of LE_v. However, traversing all the paths still requires time $\vec{1}^T(I - P_v)^{-1}\vec{1}$ which is much larger than the spanning tree sampling cost $Tr((I - P_v)^{-1})$. In practice, we can sample from the whole set of the loop-included paths.

Sample from the paths. First, we give a new result about $\mathbf{e}_u^T L_v^{-1}\vec{1}$ for $u \in V$:

Lemma 4.5. *For a loop-erased walk LE_v, let $\tilde{\tau}_{\text{LE}_v}^n[u]$ denote the expected normalized number of passes of a node u in a loop-included path $\mathcal{P}_{\text{LE}_v}[s]$ from a node s sampled uniformly from V. We have: $\tilde{\tau}_{\text{LE}_v}^n[u] = \frac{1}{n}\mathbf{e}_u^T L_v^{-1}\vec{1}$.*

According to Lemma 4.5, we can sample a small number of nodes uniformly from V, and traverse a small fraction of loop-included paths from those nodes. By Lemma 4.5, it is easy to check that the average normalized passes to a node u is an unbiased estimator of $\mathbf{e}_u^T L_v^{-1}\vec{1}$. Thus, we sample ω spanning trees using the Wilson algorithm and using the loop-erased walk estimators to update the approximate ECC vector according to Eq. (1) and Eq. (4).

Algorithm details. The pseudo-code of the algorithm LEWalk is illustrated in Algorithm 2. It takes a landmark node v and sample size ω as inputs. Following the Wilson's implementation of sampling loop-erased walks, Algorithm 2 first fixes an ordering of

$V \setminus \{v\}$ (Line 1), then it samples ω loop-erased walks with root v (Line 3-14). Specifically, for each loop-erased walk, it initializes two vectors InTree, which records the nodes that have been added into the loop-erased trajectory, and next, which records the next node in the loop-erased trajectory (Line 4). First, node v is added into the trajectory (Line 5). Then, a random walk starts from nodes that have not been added into the trajectory until it hits nodes that are already in the trajectory (Line 6-11). After that, we retrace the random walk trajectory to erase loops, and add it into the loop-erased trajectory (Line 12-14). For each node u passed, it adds $\mathbf{y}_1(u)$ by $\frac{1}{d_u}$ (Line 9). We have $E[\mathbf{y}_1(u)] = (L_v^{-1})_{uu}$. On real-life graphs, since we only sample a small fraction of nodes, we do not need to store the whole stack representation of loop-erased walks. Instead, we can sample v-absorbed walks online, since the loop-included path $\mathcal{P}_{\mathsf{LE}_v}[u]$ has the same distribution as v-absorbed path $\mathcal{P}_v[u]$. When choosing the node v as an easy-to-hit node, the cost of sampling v-absorbed walks is very small compared to sampling loop-erased walks, as verified in our experiments. Thus, after sampling the loop-erased walk, it samples ω v-absorbed walks from a uniformly sampled source node s (Line 15-19). For each time it passes a node u, we add $\mathbf{y}_1(u)$ by $\frac{1}{\omega d_u}$ (Line 19). Finally, we obtain an estimation of $(L^\dagger)_{uu}$ based on the formula $(L^\dagger)_{uu} = (L_v^{-1})_{uu} - 2\mathbf{e}_u^T L_v^{-1} \vec{1} + \frac{1}{n^2} \vec{1}^T L_v^{-1} \vec{1}$ (Line 21).

Algorithm analysis. The complexity of LEWalk is similar to that of SpanTree, while it avoids the complicated operations on trees. Thus, it is more efficient in practice.

LEMMA 4.6. *The time complexity of* LEWalk *for computing* ECC *is* $O(Tr((I - P_v)^{-1}) + \sum_{u \in V} \frac{1}{n} h(u,v))$ *for each sample.*

In real-life datasets, when properly selecting the landmark node v, the second term in Lemma 4.6 can be much smaller than the first term. For example, on the Astro-ph dataset desribed in Tab 1, $Tr((I - P_v)^{-1})$ is 2.38×10^5, $\sum_{u \in V} \frac{1}{n} h(u,v)$ is only 835, when selecting v as the highest degree node. This suggests that the time complexity of LEWalk is near $O(n)$ in real-life graphs, thus is very efficient. Since the normalized number of passes of a loop-erased walk cannot be bounded, concentration inequalities cannot be applied to analyze LEWalk. However, we find that LEWalk performs better than SpanTree in practice (see Section 6.2). This is perhaps because when the landmark node v is chosen as an easy-to-hit node, the probability that a node is passed twice in a loop-included path is very small, which makes the variance of the algorithm also small. As it is very hard to characterize the distribution of the number of passes to a node in a loop-erased walk, we leave the variance analysis of the loop-erased walk sampling algorithm as an open question.

5 ALGORITHMS FOR KC COMPUTATION

In this section, we focus on the problem of Kemeny constant computation. Recall that Kemeny constant is the trace of the pseudo-inverse of the normalized Laplacian $Tr(\mathcal{L}^\dagger)$. Based on Equation (6) and Equation (8), we have the following formula:

LEMMA 5.1. $Tr(\mathcal{L}^\dagger) = Tr((I - P_v)^{-1}) - \frac{(\mathcal{L}^\dagger)_{vv}}{\pi(v)}$.

Lemma 5.1 can be directly derived from Theorem 3.2. Compared to [18, 60], where similar formulas representing $Tr(\mathcal{L}^\dagger)$ in terms of Laplacian pseudo-inverses were proposed, our result shows that Kemeny constant can be expressed using L_v^{-1} and the combinatorial objects. According to this formula, we can also divide the task of computing Kemeny constant into two stages: (i) computing the sum of $(L_v^{-1})_{uu} d_u$ (also $(I - P_v)_{uu}^{-1}$) for all $u \in V$ and $u \neq v$; (ii)

computing $\frac{(\mathcal{L}^\dagger)_{vv}}{\pi(v)}$ (also $\frac{1}{2m} \vec{\mathbf{d}}_v^T L_v^{-1} \vec{\mathbf{d}}_v$). Then, the Kemeny constant can be easily obtained.

5.1 Existing solutions of KC computation.

The direct method to compute Kemeny constant is to apply eigendecomposition on $\mathcal{L}$. This costs $O(n^3)$ time which is prohibitive. Zhang et. al. proposes a method ApproxKemeny [64] based on solving Laplacian linear systems. Specifically, they utilize the Hutchinson's estimator that $Tr(\mathcal{L}^\dagger) = \frac{1}{\omega} \sum_{i=1}^\omega \mathbf{x}_i^T \mathcal{L}^\dagger \mathbf{x}_i$, where $\mathbf{x}_i$ is a random vector with the probability distribution $Pr[\mathbf{x}(i) \pm 1] = \frac{1}{2}$, it is known that $E[\mathbf{x}_i^T \mathcal{L}^\dagger \mathbf{x}_i] = Tr(\mathcal{L}^\dagger)$ [32]. Then, by arithmetic operation, they prove that $\frac{1}{\omega} \sum_{i=1}^\omega \mathbf{x}_i^T \mathcal{L}^\dagger \mathbf{x}_i = \frac{1}{\omega} \sum_{i=1}^\omega \|BL^\dagger \mathbf{y}_i\|_2^2$, where $\mathbf{y}_i = D^{\frac{1}{2}} (I - \frac{1}{2m} D^{\frac{1}{2}} \vec{1} \vec{1}^T D^{\frac{1}{2}}) \mathbf{x}_i$. Here, instead of directly computing $L^\dagger$, they compute $L^\dagger \mathbf{y}_i$ by solving ω Laplacian linear system $L\mathbf{z}_i = \mathbf{y}_i$. For solving the Laplacian systems, they choose the state-of-the-art fastest Laplacian solver proposed in [33]. However, although the Laplacian solver is highly optimized, solving a number of Laplacian systems is still costly in large graphs. Recently, Li et. al. proposes a Monte Carlo algorithm RW [37] for approximating Kemeny constant. Their algorithm is based on the observation that $Tr(\mathcal{L}^\dagger) = n - 1 + \sum_{k=0}^\infty (Tr(P^k) - 1)$. Then, they simulate random walks from each node u with length l for an estimation of the diagonal elements of P^k. If $k < l$, then $(P^k)_{uu}$ is the probability that a random walk from u returns back to u at the k-th step. After that, the Kemeny constant can be approximated by adding up all the $(P^k)_{uu}$ terms. However, to achieve a small estimation error, l is supposed to be very large (In [37], $l = 25791$ in their experiments). The time complexity of RW is $O(ln)$ per sample, which is costly with such a large l. In the following, we show that Kemeny constant can also be approximated by sampling loop-erased walks. The cost of sampling a spanning tree and sampling loop-erased walk with a root chosen as the landmark node v is much lower than that of sampling simple random walks.

5.2 A spanning tree sampling algorithm

In this subsection, we propose a novel Monte Carlo algorithm SpanTree to approximate Kemeny constant based on spanning tree sampling. $(L_v^{-1})_{uu}$ for $u \in V$ and $u \neq v$ can be estimated using the spanning tree sampling algorithms proposed in [4, 39]. We can estimate $Tr((I - P_v)^{-1}) = \sum_{u \in V, u \neq v} (L_v^{-1})_{uu} d_u$ by simply adding these elements. The problem remained is to approximate $\frac{1}{2m} \vec{\mathbf{d}}_v^T L_v^{-1} \vec{\mathbf{d}}_v$. For $\frac{1}{2m} \vec{\mathbf{d}}_v^T L_v^{-1} \vec{\mathbf{d}}_v$, as it is a weighted sum of L_v^{-1}, we can estimate it in a similar way as estimating $\vec{1}^T L_v^{-1} \vec{1}$ based on the "spanning tree" interpretation of L_v^{-1}. First, we find that we can represent $\frac{1}{2m} \vec{\mathbf{d}}_v^T L_v^{-1} \vec{\mathbf{d}}_v$ in terms of electrical systems:

COROLLARY 2. *Consider an electrical network that $\frac{d_u}{2m}$ current flow comes in through every node $u \in V$, and unit current flow comes out through node v. Let $\mathbf{x}(u)$ be the voltage at node u. Then $\frac{(\mathcal{L}^\dagger)_{vv}}{\pi(v)}$ is the weighted sum of voltage $\sum_{u \in V} \mathbf{x}(u) d_u$.*

Based on the electrical system interpretation of $\frac{1}{2m} \vec{\mathbf{d}}_v^T L_v^{-1} \vec{\mathbf{d}}_v$, we can also approximate $\frac{1}{2m} \vec{\mathbf{d}}_v^T L_v^{-1} \vec{\mathbf{d}}_v$ by sampling spanning trees as directly solving these linear systems is hard. For example, as illustrated in Fig. 4(a), $\frac{(\mathcal{L})_{22}}{\pi(2)} = 1.48$ is exactly the weighted sum of all voltages in the electrical network, which is $\frac{32*3}{80} + \frac{32*3}{80} + \frac{40*2}{80}$. And the electrical network flow is the average of the network flow on each spanning trees, as illustrated in Fig. 4(b). It suggests that we can sample spanning trees, send flows, and take the average (weighted

(a) G, network flow   (b) flows on spanning trees
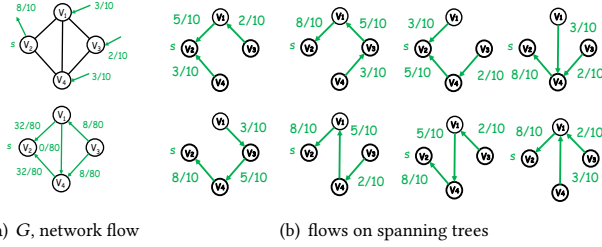
Figure 4: Illustration of approximating $\mathcal{L}^{\dagger}$ by spanning trees. (a) A graph G and an electrical network flow where $\frac{d_u}{2m}$ current flow comes in through every node $u \in V$ and a unit current flow comes out through v_2; (b) There are 8 spanning trees. The current flows along an edge (u,v) in G equals the average current flows through (u,v) on all spanning trees.

sum of) voltages as an approximation of Kemeny constant. We refer to such an algorithm as SpanTree, the pseudo-code is deferred to the appendix [5]. Similar to the spanning tree sampling algorithm for computing ECC, the time complexity is also $O(Tr((I-P_v)^{-1}) + n\Delta_G)$ per sample. We can also give an absolute error guarantee by properly setting the sample size ω.

THEOREM 5.2. *If the sample size* $\omega \geq \frac{8m^2\Delta_G^2 \log(\frac{2}{p_f})}{\epsilon^2}$, *SpanTree can output $\tilde{\kappa}(G)$ that satisfies $|\tilde{\kappa}(G) - \kappa(G)| \leq \epsilon$ with a probability at least $1 - p_f$.*

Notice that the m^2 term is mainly because that the Kemeny constant also scales with m (to see this, the worst-case lower bound of the Kemeny constant is $1 + \frac{(n-1)^2}{n}$, and it is often larger in general graphs), and we consider the absolute errors. If we consider the relative error, e.g. $|\tilde{\kappa}(G) - \kappa(G)| \leq m\epsilon'$, the m^2 term can be removed. This can be derived from Theorem 5.2 by directly setting $\epsilon = m\epsilon'$.

5.3 A loop-erased walk sampling algorithm

In this subsection, we propose the second novel Monte Carlo algorithm LEWalk for approximating Kemeny constant based on loop-erased walk sampling. By the "loop-erased walk" interpretation of L_v^{-1}, $Tr((I-P_v)^{-1})$ is exactly the expected number of walk lengths of a loop-erased walk LE_v with root v. Thus, $Tr((I-P_v)^{-1})$ can be approximated by sampling a number of loop-erased walks and take the average of the random walk lengths. On the other hand, $\frac{(\mathcal{L}^{\dagger})_{vv}}{\pi(v)} = \frac{1}{2m}\vec{\mathbf{d}}_v^T L_v^{-1} \vec{\mathbf{d}}_v$ can also be interpreted by loop-erased walks. Since $\frac{1}{2m}\vec{\mathbf{d}}_v^T L_v^{-1}\vec{\mathbf{d}}_v = \sum_{u \in V, u \neq v} \pi(u) e_u^T L_v^{-1}\vec{\mathbf{d}}_v = \sum_{u \in V} \pi(u) h(u,v)$, it can be represented as the expected length of a loop-included path $\mathcal{P}_{\text{LE}_v}[u]$ with node u sampled from the stationary distribution $\boldsymbol{\pi}$. To approximate $\frac{1}{2m}\vec{\mathbf{d}}_v^T L_v^{-1}\vec{\mathbf{d}}_v$, we can traverse the stack representation of the loop-erased walks sampled for approximating $Tr((I-P_v)^{-1})$. Such an algorithm requires to store the stack representation of the loop-erased walks, which consumes $Tr((I-P_v)^{-1})$ space per sample. In order to save memory consumption, we can approximate $\frac{1}{2m}\vec{\mathbf{d}}_v^T L_v^{-1}\vec{\mathbf{d}}_v$ by sampling v-absorbed walks online (with little additional cost) since the loop-included path $\mathcal{P}_{\text{LE}_v}[u]$ has the same distribution as the v-absorbed walk path $\mathcal{P}_v[u]$. We refer to such an algorithm as LEWalk, the pseudo-code is deferred to the appendix [5].

The running time of LEWalk consists of two parts: (i) the time of sampling loop-erased walks ($Tr((I-P_v)^{-1})$) and (ii) the time of sampling v-absorbed walks ($\sum_{u \in V} \pi(u) h(u,v)$). For real-world

graphs, there often exists a high-degree node that is easy-to-hit. The value of $\sum_{u \in V} \pi(u) h(u,v)$ is much smaller than $Tr((I-P_v)^{-1})$. Thus, the expected running time of LEWalk is $O(Tr((I-P_v)^{-1}))$ per sample. Similar to the case of ECC computation, concentration inequalities cannot be applied to analyze LEWalk, because the random walk length of a loop-erased walk cannot be bounded. Nevertheless, the empirical performance of LEWalk is very good (even much better than SpanTree), as evidenced in our experiments.

Comparison to [4] and [39]. To our knowledge, our work is the first to offer combinatorial interpretations of $L^{\dagger}$ and KC in terms of random walks, electrical voltages and energies, spanning trees, and loop-erased walks, which extends the work [21] which represents $L^{\dagger}$ in terms of 2-spanning forest. The proposed algorithms can significantly outperform the SOTA approaches for computing ECC [4] and KC [37, 64]. We notice that there are two previous studies [4, 39] which also utilize the techniques of spanning tree and loop-erased walk sampling. However, our study is fundamentally different from these studies. First, our work differs from [4] in two aspects: (i) [4] only interprets effective resistance in terms of spanning trees. This is natural since it is defined as difference of voltages. Unlike [4], we convert $L^{\dagger}$ to L_v^{-1}, and connect L_v^{-1} to the voltage of a node, and then send flows on spanning trees. Here, the idea of sending flows on spanning trees is novel; (ii) Our algorithm is a pure spanning tree sampling algorithm while the algorithm in [4] involves a black-box Laplacian solver. Our experiments show that our pure sampling algorithm demonstrates much higher efficiency than the algorithm proposed in [4]. Second, our work also differs from [39] in two aspects: (i) Theorem 3.1, Lemma 3.6 and Lemma 3.7 are much stronger than Theorem 3.1, Lemma 3.8 and Lemma 5.1 in [39], as we discussed previously. (ii) The algorithms in [39] are tailored for computing effective resistance. To estimate ECC and KC, it needs to first express them in terms of effective resistances (they are weighted sum of effective resistances between all node pairs [10, 68]). This requires invoking the algorithms in [39] for n times, which is very expensive. Our results provide efficient algorithms for ECC and KC, and it is non-trivial to design data structures to ensure that sending currents from multiple nodes without increasing the time complexity too much.

Extentions to other Laplacian based quantities. Although our main focus in this study is on approximating the electrical closeness centrality and Kemeny constant, it is worth noting our approaches can also be applied to approximate other graph metrics, such as the Kirchhoff index [68] and random walk betweenness centrality [44]. Specifically, the Kirchhoff index [68] is defined as $Tr(L^{\dagger})$. We have already estimated the diagonal elements $(L^{\dagger})_{uu}$ for $u \in V$ in Section 4. To obtain an estimation of the Kirchhoff index, we simply need to sum up all these diagonal elements. The random walk betweenness centrality $\mathbf{c}_b$ [44] is defined as $\mathbf{c}_b(u) = \frac{1}{n} + \frac{Tr(L^{\dagger})}{(n-1)(Tr(L^{\dagger}) + n(L^{\dagger})_{uu})}$ for $u \in V$ [4]. Similarly, to approximate this quantity, we can sum up the diagonal elements $(L^{\dagger})_{uu}$ for $u \in V$ using the proposed techniques.

6 EXPERIMENTS
6.1 Experimental settings

Datasets. We use 5 real-life datasets: 2 small graphs and 3 large graphs. All datasets can be obtained from [35] and the detailed statistics is shown in Table 1. Note that we also estimate $\frac{Tr((I-P_v)^{-1})}{n}$ for each dataset by simulating 10^4 loop-erased walks with root v as the highest-degree node and take the average steps. We can observe that real-life datasets often have a small value of $\frac{Tr((I-P_v)^{-1})}{n}$. This
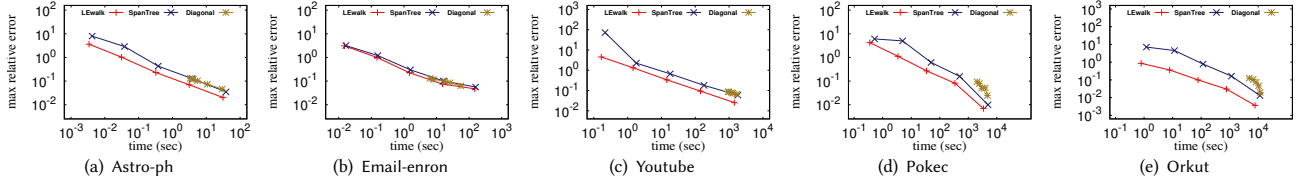
Figure 5: Maximum relative error v.s. runtime of different algorithms for approximating electrical closeness centrality
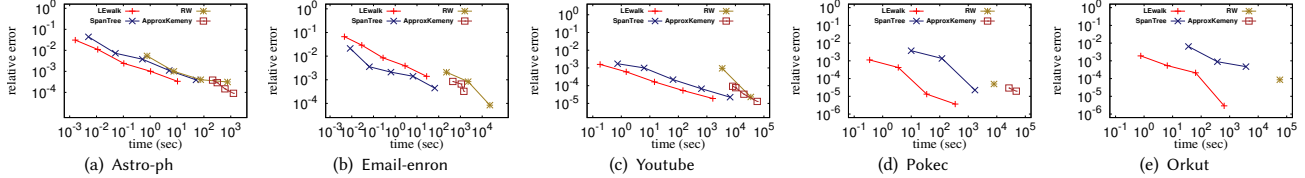

Figure 6: Relative error v.s. runtime of different algorithms for approximating Kemeny constant

confirms our analysis that sampling a loop-erased walk has an $O(n)$ time complexity on real-life datasets. For ECC approximation, the result is a vector $\mathbf{c}$ with $\mathbf{c}(u)$ represents the ECC of node u for all $u \in V$. On two small graphs, we can apply eigen-decomposition to compute the exact value of ECC. On three large graphs, it is hard to derive the exact values. Thus, we use our best algorithm LEWalk with $\omega = 10^6$ for ECC approximation to obtain "ground-truths". For KC approximation, the result is a constant $\kappa(G)$. On two small graphs, we apply eigen-decomposition to compute the exact value of KC. On three large graphs, it is hard to derive the exact values. Thus, we use our best algorithm LEWalk with $\omega = 10^6$ for KC approximation to obtain "ground-truths". As verified in the experiments, the variance of our algorithm is very small with that setting of ω, therefore we can expect that the estimated "ground-truth" values are very close to their true values.

Parameters. For ECC approximation, we implement two algorithms, SpanTree and LEWalk. There is one parameter ω which controls the sample size. We compare the proposed algorithms with the state-of-the-art (SOTA) method Diagonal [4]. There are two parameters ϵ, κ in Diagonal which controls the accuracy of the algorithm. We set $\kappa = 0.3$ as default and vary ϵ following [4]. We do not include other algorithms because they are all outperformed by Diagonal [4]. For KC approximation, we implement two our algorithms SpanTree and LEWalk. There is also one parameter ω that controls the sample size. We compare the two proposed algorithms with two SOTA methods, RW [37] and ApproxKemeny [64]. RW has one parameter ω to control the sample size. ApproxKemeny has one parameter ϵ to control the accuracy. For all the proposed algorithms, there is also a landmark node v that we take as input. We set v as the highest-degree node by default, and we will evaluate the other choices of v in Section 6.2.

Experimental environment. All the algorithms are implemented in C++. All experiments are run on a Linux 20.04 server with Intel 2.0 GHz CPU and 128GB memory. All algorithms used in our experiments are compiled using GCC9.3.0 with -O3 optimization. An exception is the comparison method ApproxKemeny [64]. Despite utilizing the same black-box C++ Laplacian solver as implemented in [4], our C++ implementation of ApproxKemeny under-perform compared to the original Julia counterpart [64]. To ensure a fair and rigorous comparison, and to maintain the integrity of experimental results in line with the original paper, we have opted to report results based on the original Julia implementation in [64]. Similar decision was also made by the authors in [37], who also compared their algorithms using the Julia version of ApproxKemeny originally implemented in [64], although their algorithms were also not implemented in Julia.

Table 1: Datasets ($\bar{d}$: average degree; Δ_G: diameter of the graph)

Type	Dataset	n	m	$\bar{d}$	Δ_G	$\frac{Tr((I-P_v)^{-1})}{n}$
Small graphs	Astro-ph	17,903	196,972	22	14	1.33
	Email-enron	33,696	180,811	10.73	13	1.43
Large graphs	Youtube	1,134,890	2,987,624	5.27	24	1.55
	Pokec	1,632,803	22,301,964	27.32	14	1.07
	Orkut	3,072,441	117,184,899	76.28	10	1.02

6.2 Performance studies

Results of ECC computation. We first evaluate the performance of algorithms for approximating ECC. We plot the error v.s. time results for the two proposed algorithms SpanTree and LEWalk, as well as the SOTA method Diagonal [4]. For SpanTree and LEWalk, we vary the sample size ω from 10^0 to 10^5. For Diagonal, we vary ϵ from 0.9 to 0.3 following the settings in [4]. We evaluate the error of ECC vector by the maximum relative error defined as $\max_{u \in V} \frac{|\mathbf{c}(u) - \tilde{\mathbf{c}}(u)|}{\mathbf{c}(u)}$. We also evaluate the error by the $L1$-error which is defined as $\sum_{u \in V} |\tilde{\mathbf{c}}(u) - \mathbf{c}(u)|$. We run each algorithm for 100 times and report the average results. The relative error results are shown in Fig. 5, while the $L1$-error results are deferred to the appendix [5]. As can be seen, SpanTree can achieve comparable and slightly better relative error and $L1$-error compared to Diagonal within similar time. For example, on Pokec, when taking 4000 seconds to compute the ECC, Diagonal can achieve a max relative error 0.01 while SpanTree can achieve a max relative error as small as 0.006. Note that SpanTree avoids the calling of highly-optimized Laplacian solvers as invoked in Diagonal. It is a "pure" spanning tree sampling algorithm which is much easier to implement. LEWalk can achieve a certain accuracy in terms of both relative error and $L1$-error in much less time compared to SpanTree and Diagonal. For example, on Orkut, LEWalk takes 774 seconds while SpanTree takes $12,540$ seconds to achieve a max relative error 0.01. These results demonstrate the high efficiency of our loop-erased walk sampling approach.

Results of KC computation. In this experiment, we study the performance of algorithms for approximating KC. We plot the error v.s. time results for two proposed algorithms SpanTree and LEWalk, as well as two SOTA methods RW [37] and ApproxKemeny [64]. For SpanTree, LEWalk and RW, we vary the sample size ω from 10^0 to 10^5, following a similar experimental setting in [37]. For ApproxKemeny, we vary the accuracy parameter ϵ from 0.9 to 0.3 following [64]. We measure the quality of results by the relative error defined as $\frac{|\tilde{\kappa}(G) - \kappa(G)|}{\kappa(G)}$ as well as the absolute error defined as $|\tilde{\kappa}(G) - \kappa(G)|$. We run each algorithm for 100 times and report the average results. We omit the results when the running time of

(a) Email-enron   (b) Youtube
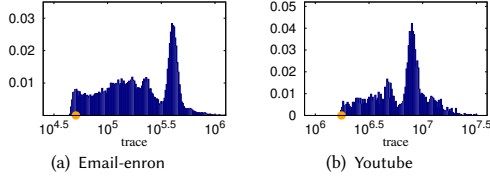
Figure 7: Distribution of $Tr((I - P_v)^{-1})$ on different datasets (the orange circle denotes the value of $Tr((I - P_v)^{-1}$ with v chosen as the highest-degree node)

an experiment exceeds 24 hours. The results of relative error are shown in Fig. 6, while the results of absolute error are deferred to the appendix [5] due to the space limits. As can be seen, the running time of our algorithms are orders of magnitude faster than the SOTA algorithms. For example, on Youtube, to achieve a relative error 10^{-4}, RW takes 34,637 seconds, ApproxKemeny takes 45,609 seconds, while the proposed algorithm LEWalk takes only 1,615 seconds, which is 21.45× faster than the SOTA algorithms. For the proposed algorithms, LEWalk is slightly faster than SpanTree. On large datasets Youtube and Orkut, LEWalk can achieve an absolute error as small as 10^1 in hundreds of seconds, and its relative error is also extremely small (10^{-4}). These results demonstrate that our loop-erased walk sampling based algorithms are very efficient in computing Kemeny constant.

Effect of landmark node v. In former experiments, we select the landmark node v as the highest-degree node by default. In this experiment, we investigate the effect of various landmark selection strategies. We conduct experiments on both small graphs and large graphs. For small graphs, we run LEWalk and SpanTree for KC approximation with all nodes in graph chosen as the landmark node v. For large graphs, we sample 10^4 nodes uniformly, and run LEWalk and SpanTree with those nodes chosen as landmarks. We run all algorithms for 100 times and report the average results. We first plot the distribution of $Tr((I - P_v)^{-1})$, since it is the expected running time of sampling a loop-erased walk LE_v, which is used in all proposed algorithms. The results are shown in Fig. 7. We show the results of a small graph Email-enron and a large graph Youtube, and the results on the other datasets are consistent. It can be seen that $Tr((I - P_v)^{-1})$ varies when v changes, the highest-degree node has (nearly) smallest $Tr((I - P_v)^{-1})$ value. Then, we plot the runtime and error (relative error) distribution with different v for KC computation (Due to space limits, the results for ECC computation is deferred to the appendix [5]). The results can be found in Fig. 8. Again, we can see that the highest-degree node is almost the best choice for achieving short time and low error. The reason is that the highest-degree node is often easy-to-hit, thus the length of the loop-included path in LE_v may be short, leading to both low traversal cost and small variance. These results confirm our theoretical analysis in Section 3.

6.3 Case studies

In this subsection, we conduct case studies to show the applications of ECC as a graph centrality metric, and KC as a graph invariant. **Comparison of different centrality measures on** DBLP **and**

WordNet**.** We first compare different centrality measures on two real-life datasets. DBLP is a collaboration network for researchers in database and data mining with 37,177 nodes and 131,715 edges, where each node represents an author, each edge represents a collaboration relationship. WordNet is a word network with 5,040 nodes and 55,258 edges, where each node is a word, each edge appears when there is a dependency. We compare four commonly used centrality measures, degree centrality, PageRank centrality, betweenness centrality (BC) [42] and closeness centrality (CC) [46]

(a) Email-enron, time   (b) Email-enron, error

(c) Youtube, time   (d) Youtube, error
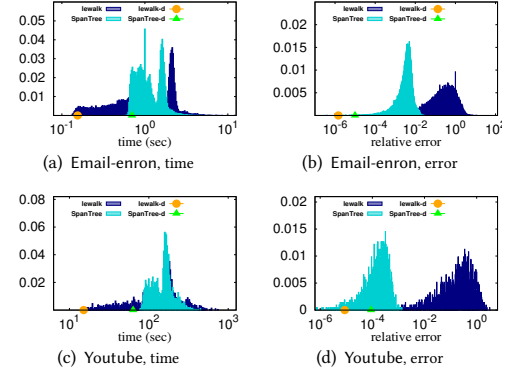
Figure 8: Distribution of performance of algorithms for different choices of landmark node (Kemeny constant)

with the electrical closeness centrality (ECC). For PageRank centrality, we set $\alpha = 0.15$, which is the commonly-used choice [29]. The top-10 centrality nodes in DBLP and WordNet measured by these centralities are shown in Table 2. As can be seen, all the top-10 results are well-known authors and commonly-used words. The results of ECC are close to that of other widely used centrality measures. This result verifies that ECC is indeed a useful centrality metric. Furthermore, we also rank the centrality values after randomly removing 30% edges from the original graphs. The results are shown in Table 2. As discussed before, ECC considers all paths between nodes, while both CC and BC only consider the shortest paths. Thus, ECC is often more robust with respect to edge perturbations. We can observe in Table 2 that after removing a fraction of edges, there are more drastic changes in the rankings of BC and CC compared to ECC. These results suggest that ECC is an effective and robust metric for detecting the important nodes in real-life networks.

Kemeny constant for real-life networks. We compute KC for 7 real-life networks. All the datasets are available at [35]. In summary, they can be divided into 3 categories: social networks, collaboration networks and co-purchasing networks. We plot the results with node number n on the x axis, and $\frac{\kappa(G)}{n}$ on the y axis. The results are shown in Fig. 9(a). We can clearly see that the social network Pokec, LiveJournal, Orkut has $\kappa(G)$ very close to the node number n, while the 3 citation networks DBLP, Hep-th and Astro-ph have relatively larger $\kappa(G)$. The co-purchasing network Amazon has the largest $\frac{\kappa(G)}{n}$ value. It suggests that social network connects more closely compared to collaboration networks and co-purchasing networks. However, all these real-life networks have $O(n)$ Kemeny constants. In [37], the authors find that real-world power-law graphs often has a KC value close to the optimal value $1 + \frac{(n-1)^2}{n}$, which is also the KC value for complete graphs. The result implies that Kemeny constant can reveal the connectivity property of a network, and real-life networks often have a small and near-optimal Kemeny constant.

Kemeny constant for synthetic networks. We also compute KC for synthetic networks. There are many models to generate networks that are similar to real-world graphs. We select three popular models: the Barabasi-Albert graph (BA), the Erdos-Renyi graph (ER) and the hyperbolic random graph (HY). Specifically, BA model generates networks based on a small network with $m_0 \geq m$ edges. For each new coming node, it add m new edges based on the degree of the current nodes. BA model generates graphs with a power-law distribution. ER model generates the famous $G(n, p)$ Erdos-Renyi graphs with n nodes and p is the probability that there exists an edge between any pair of nodes. The HY model generates

Table 2: 10 **centrality results on** DBLP **(the original graph and the graph obtained by removing 30% edges)**

	the original graph					the graph obtained by removing 30% edges				
Rank	Degree	PageRank	BC	CC	ECC	Degree	PageRank	BC	CC	ECC
1	Jiawei Han	Jiawei Han	Philip S. Yu	Jiawei Han	Jiawei Han	Christos Faloutsos	Christos Faloutsos	Christos Faloutsos	Divesh Srivastava	Christos Faloutsos
2	Philip S. Yu	Philip S. Yu	Jiawei Han	Philip S. Yu	Philip S. Yu	Jiawei Han	Philip S. Yu	Philip S. Yu	Beng Chin Ooi	Jiawei Han
3	Christos Faloutsos	Christos Faloutsos	Christos Faloutsos	Beng Chin Ooi	Christos Faloutsos	Philip S. Yu	Jiawei Han	Jiawei Han	Michael J. Franklin	Philip S. Yu
4	Michael J. Franklin	Jian Pei	Gerhard Weikum	Christos Faloutsos	Michael J. Franklin	Michael J. Franklin	Jian Pei	Jian Pei	Christos Faloutsos	Michael J. Franklin
5	Jian Pei	Gerhard Weikum	Beng Chin Ooi	Gerhard Weikum	Raghu Ramakrishnan	Gerhard Weikum	Gerhard Weikum	Gerhard Weikum	Raghu Ramakrishnan	Raghu Ramakrishnan
6	Gerhard Weikum	Michael Stonebraker	Jian Pei	Michael J. Franklin	Beng Chin Ooi	Jian Pei	Michael Stonebraker	Michael J. Franklin	Raghu Ramakrishnan	Jian Pei
7	Michael Stonebraker	Hector Garcia-Molina	Michael J. Franklin	H. V. Jagadish	Jian Pei	Michael Stonebraker	W. Bruce Croft	ChengXiang Zhai	H. V. Jagadish	Gerhard Weikum
8	Raghu Ramakrishnan	Michael J. Franklin	Raghu Ramakrishnan	Divesh Srivastava	Gerhard Weikum	Raghu Ramakrishnan	Michael J. Franklin	Beng Chin Ooi	Kenneth A. Ross	Beng Chin Ooi
9	Beng Chin Ooi	Beng Chin Ooi	Haixun Wang	Kenneth A. Ross	Jeffrey F. Naughton	Beng Chin Ooi	Elke A. Rundensteiner	Raghu Ramakrishnan	AnHai Doan	Surajit Chaudhuri
10	Hector Garcia-Molina	Haixun Wang	H. V. Jagadish	Raghu Ramakrishnan	Michael Stonebraker	Hector Garcia-Molina	Hector Garcia-Molina	Charu C. Aggarwal	Jian Pei	Michael Stonebraker

(a) real-world datasets   (b) Barabasi-Albert graphs   (c) Erdos-Renyi graphs   (d) Hyperbolic graphs
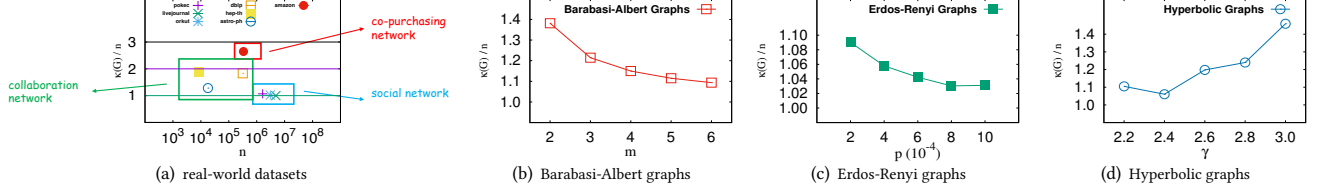
Figure 9: Kemeny constant for various networks

points in hyperbolic space and add edges between them according to the distance between them. The generated graphs have power-law degree distribution, small diameter and high clustering coefficient which is similar to real-world graphs [30]. There is one parameter to control the power-law degree γ, and the other parameter average degree $\bar{d}$, which we set as 40 by default. We generate 100 networks with 10^6 nodes, and vary m in BA as $m = \{2, 3, 4, 5, 6\}$; p in ER as $p = \{2, 4, 6, 8, 10\} \times 10^{-4}$ and γ as $\gamma = \{2.2, 2.4, 2.6, 2.8.3.0\}$. The results are shown in Fig. 9(b)-(d). We plot the parameter on the x-axis, and $\frac{\kappa(G)}{n}$ on the y-axis. As can be seen, as m grows larger (as p becomes smaller), $\frac{\kappa(G)}{n}$ decreases. Also, as γ becomes larger, $\frac{\kappa(G)}{n}$ becomes larger. However, the KC value is often very close to the node number n. The findings can help us design and generate networks, which can be applied in robot surveillance [25] and web search [36] where networks with low Kemeny constants (with low communication / navigation cost) are desired. This result further confirms that real-world graphs often have a near optimal KC value.

7 FURTHER RELATED WORK

Spanning tree sampling. Spanning tree sampling is a powerful tool for efficient approximation of graph-related quantities. It is first used by Hayashi et al. [31] for approximating spanning edge centrality, which is $(e_s - e_t)^T L^\dagger (e_s - e_t)$ (also $(L_s^{-1})_{tt}$) if there is an edge between s and t. Angriman et al. [4, 39] generalize the technique to approximate resistance distance $(e_s - e_t)^T L^\dagger (e_s - e_t)$ (also $(L_s^{-1})_{tt}$) for arbitrary s and t. However, how to use spanning tree sampling to approximate $L^\dagger$ or the arbitrary elements of L_v^{-1} still remains unresolved. Yusuf et al. [9, 49, 50] use spanning forests sampling for graph signal processing tasks. Their objective is to approximate $(L + qI)^{-1} qI$ and $Tr((L + qI)^{-1} qI)$. Recently, Liao et al. [40] uses spanning forests sampling to approximate personalized PageRank, which is $(L + \beta D)^{-1} \beta D$. These work can be regarded as a special case of our techniques for L_v^{-1}, because the matrix $L + qI$ and $L + \beta D$ are submatrix of the Laplacian of an extended graph with an additional dummy node.

Efficient sampling a spanning tree is a well-studied problem in theoretical computer science. Aldous-Broder algorithm [1] simulates random walks from a node until all nodes are covered. Russo et al. propose an algorithm [53] based on Makrov chain built over all spanning trees, but the algorithm requires complex data structure such as link-cut trees. For theoretical results, the algorithm [54] by Schild is the best with time complexity $O(m^{1+o(1)})$. For real-life large datasets, the Wilson algorithm that utilizing loop-erased walk sampling [62] is the most efficient algorithm, and it is chosen for

spanning tree sampling algorithms in previous work [4, 31, 39]. The time complexity of the Wilson algorithm is $Tr((I - P_v)^{-1})$ [43]. Specifically, Liao et. al. [39] gives a local algorithm for sampling only part of spanning trees which runs in time $h(s, v) + h(t, v)$ if one only wants the path between s and t.

Loop-erased walk sampling. Compared to spanning tree sampling, loop-erased walk sampling itself is less studied. Wilson et al. [62] prove that the expected running time of the Wilson algorithm with root v is $\sum_{u \in V} (\pi(u) h(u, v) + h(v, u) \pi(u))$. Marchal et al. [43] further proves that the time complexity can be formulated as $Tr((I - P_v)^{-1})$. Recently, Liao et al. [39] utilize loop-erased walk to approximate single source resistance distance, which is $(L_v^{-1})_{uu}$ for $u \in V$ and $u \neq v$. However, it is still unknown how to approximate all elements of L_v^{-1} and $L^\dagger$ using loop-erased walk sampling.

Other random walk based quantities computation. In addition to Laplacian pseudo-inverse, there also exist some other random walk based quantities, such as personalized PageRank (PPR) and resistance distance. Push-based methods [2, 3, 39] and random walk sampling based methods [7, 39, 47, 65] are developed to compute PPR and resistance distance. The advantage of such methods is that they are local, meaning that they often only need to visit a small portion of the graph instead of the entire graph. In comparison, Laplacian pseudo-inverse is a global quantity which cannot be approximated locally. Random walk sampling and push are combined to further improve the efficiency of PPR computation [38, 41, 59, 63] and resistance distance computation [39, 65]. However, when it comes to deriving the Laplacian pseudo-inverse, it is necessary to compute the entire diagonal of L_v^{-1}. This requires to invoke the random walk sampling and push algorithms for n times, which is clearly very expensive for large graphs.

8 CONCLUSION

In this paper, we study the problem of efficient computation of two kinds of Laplacian pseudo-inverse, with applications to electrical closeness centrality approximation and Kemeny constant approximation. We first give new formulas to express the pseudo-inverses of two types of Laplacians, including the combinatorial Laplacian and the normalized Laplacian, by L_v^{-1}. Then, we propose two new combinatorial interpretations of L_v^{-1} in terms of two interesting objects: spanning trees and loop-erased walks. Based on these theoretical results, we develop two novel Monte Carlo algorithms for both electrical closeness centrality and Kemeny constant computation. The results of extensive experiments on 5 real-life datasets demonstrate that our proposed algorithms substantially outperform the state-of-the-art algorithms in terms of both runtime and accuracy.

REFERENCES

[1] David J Aldous. 1990. The random walk construction of uniform spanning trees and uniform labelled trees. *SIAM Journal on Discrete Mathematics* 3, 4 (1990), 450–465.

[2] Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. 2007. Local Computation of PageRank Contributions. In *WAW*. 150–165.

[3] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. 2006. Local Graph Partitioning using PageRank Vectors. In *FOCS*. 475–486.

[4] Eugenio Angriman, Maria Predari, Alexander van der Grinten, and Henning Meyerhenke. 2020. Approximation of the Diagonal of a Laplacian's Pseudoinverse for Complex Network Analysis. In *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference) (LIPIcs, Vol. 173)*. 6:1–6:24.

[5] Anonymous Authors. 2024. Scaling up Graph Laplacian Pseudo-inverse Computation. *Full version: https://anonymous.4open.science/r/KC-vldb-2223* (2024).

[6] Luca Avena, Fabienne Castell, Alexandre Gaudillière, and Clothilde Mélot. 2018. Random forests and networks analysis. *Journal of Statistical Physics* 173, 3 (2018), 985–1027.

[7] Konstantin Avrachenkov, Nelly Litvak, Danil Nemirovsky, and Natalia Osipova. 2007. Monte Carlo Methods in PageRank Computation: When One Iteration is Sufficient. *SIAM J. Numer. Anal.* 45, 2 (2007), 890–904.

[8] Ravindra B Bapat. 2010. *Graphs and matrices*. Vol. 27. Springer.

[9] Simon Barthelmé, Nicolas Tremblay, Alexandre Gaudilliere, Luca Avena, and Pierre-Olivier Amblard. 2019. Estimating the inverse trace using random forests on graphs. *arXiv preprint arXiv:1905.02086* (2019).

[10] Dario Bini, Jeffrey J. Hunter, Guy Latouche, Beatrice Meini, and Peter G. Taylor. 2018. Why is Kemeny's constant a constant? *J. Appl. Probab.* 55, 4 (2018), 1025–1036.

[11] Daniel Boley. 2021. On fast computation of directed graph Laplacian pseudo-inverse. *Linear Algebra Appl.* 623 (2021), 128–148.

[12] Daniel Boley, Gyan Ranjan, and Zhi-Li Zhang. 2011. Commute times for a directed graph using an asymmetric Laplacian. *Linear Algebra Appl.* 435, 2 (2011), 224–242.

[13] Béla Bollobás. 1998. *Modern graph theory*. Vol. 184. Springer Science & Business Media.

[14] Phillip Bonacich and Paulette Lloyd. 2001. Eigenvector-like measures of centrality for asymmetric relations. *Soc. Networks* 23, 3 (2001), 191–201. https://doi.org/10.1016/S0378-8733(01)00038-7

[15] Enrico Bozzo. 2013. The Moore–Penrose inverse of the normalized graph Laplacian. *Linear Algebra Appl.* 439, 10 (2013), 3038–3043.

[16] Enrico Bozzo and Massimo Franceschet. 2012. Approximations of the Generalized Inverse of the Graph Laplacian Matrix. *Internet Math.* 8, 4 (2012), 456–481.

[17] Enrico Bozzo and Massimo Franceschet. 2013. Resistance distance, closeness, and betweenness. *Social Networks* 35, 3 (2013), 460–469.

[18] Minerva Catral, Stephen J. Kirkland, Michael Neumann, and Nung-Sing Sze. 2010. The Kemeny Constant for Finite Homogeneous Ergodic Markov Chains. *J. Sci. Comput.* 45, 1-3 (2010), 151–166.

[19] Seth Chaiken. 1982. A combinatorial proof of the all minors matrix tree theorem. *SIAM Journal on Algebraic Discrete Methods* 3, 3 (1982), 319–329.

[20] Pavel Chebotarev and Rafig Agaev. 2002. Forest matrices around the Laplacian matrix. *Linear algebra and its applications* 356, 1-3 (2002), 253–274.

[21] Fan Chung and Ji Zeng. 2023. Forest formulas of discrete Green's functions. *Journal of Graph Theory* 102, 3 (2023), 556–577.

[22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.

[23] Karel Devriendt, Samuel Martin-Gutierrez, and Renaud Lambiotte. 2022. Variance and Covariance of Distributions on Graphs. *SIAM Rev.* 64, 2 (2022), 343–359.

[24] Karel Devriendt and Piet Van Mieghem. 2019. The simplex geometry of graphs. *J. Complex Networks* 7, 4 (2019), 469–490.

[25] Xiaoming Duan, Mishel George, Rushabh Patel, and Francesco Bullo. 2020. Robotic Surveillance Based on the Meeting Time of Random Walks. *IEEE Trans. Robotics* 36, 4 (2020), 1356–1362.

[26] Katherine Fitch and Naomi Ehrich Leonard. 2013. Information centrality and optimal leader selection in noisy networks. In *52nd IEEE Conference on Decision and Control*. IEEE, 7510–7515.

[27] Santo Fortunato, Vito Latora, and Massimo Marchiori. 2004. Method to find community structures based on information centrality. *Physical review E* 70, 5 (2004), 056104.

[28] François Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens. 2007. Random-Walk Computation of Similarities between Nodes of a Graph with Application to Collaborative Recommendation. *IEEE Trans. Knowl. Data Eng.* 19, 3 (2007), 355–369.

[29] David F. Gleich. 2015. PageRank Beyond the Web. *SIAM Rev.* 57, 3 (2015), 321–363.

[30] Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. 2012. Random hyperbolic graphs: degree sequence and clustering. In *Automata, Languages, and Programming: 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II 39*. Springer, 573–585.

[31] Takanori Hayashi, Takuya Akiba, and Yuichi Yoshida. 2016. Efficient Algorithms for Spanning Tree Centrality. In *IJCAI*. 3733–3739.

[32] Michael F Hutchinson. 1990. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation* 19, 2 (1990), 433–450.

[33] Rasmus Kyng and Sushant Sachdeva. 2016. Approximate Gaussian Elimination for Laplacians - Fast, Sparse, and Simple. In *FOCS*. 573–582.

[34] Amy N Langville and Carl D Meyer. 2011. Google's PageRank and beyond. In *Google's PageRank and Beyond*. Princeton university press.

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

[36] Mark Levene and George Loizou. 2002. Kemeny's Constant and the Random Surfer. *Am. Math. Mon.* 109, 8 (2002), 741–745.

[37] Shiju Li, Xin Huang, and Chul-Ho Lee. 2021. An Efficient and Scalable Algorithm for Estimating Kemeny's Constant of a Markov Chain on Large Graphs. In *KDD*. 964–974.

[38] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, Hongyang Chen, Hongchao Qin, and Guoren Wang. 2023. Efficient Personalized PageRank Computation: The Power of Variance-Reduced Monte Carlo Approaches. *Proc. ACM Manag. Data* 1, 2 (2023), 160:1–160:26.

[39] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, Hongyang Chen, Hongchao Qin, and Guoren Wang. 2023. Efficient Resistance Distance Computation: The Power of Landmark-based Approaches. *Proc. ACM Manag. Data* 1, 1 (2023), 68:1–68:27.

[40] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, and Guoren Wang. 2022. Efficient Personalized PageRank Computation: A Spanning Forests Sampling Based Approach. In *SIGMOD*. ACM, 2048–2061.

[41] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2016. Personalized PageRank Estimation and Search: A Bidirectional Approach. In *WSDM*. 163–172.

[42] Fragkiskos D. Malliaros, Christos Giatsidis, Apostolos N. Papadopoulos, and Michalis Vazirgiannis. 2020. The core decomposition of networks: theory, algorithms and applications. *VLDB* (2020), 61–92.

[43] Philippe Marchal. 2000. Loop-erased random walks, spanning trees and Hamiltonian cycles. *Electronic Communications in Probability* 5 (2000), 39–50.

[44] Onuttom Narayan and Iraj Saniee. 2018. Scaling of Random Walk Betweenness in Networks. In *Complex Networks and Their Applications VII - Volume 1 Proceedings The 7th International Conference on Complex Networks and Their Applications COMPLEX NETWORKS 2018, Cambridge, UK, December 11-13, 2018 (Studies in Computational Intelligence, Vol. 812)*. Springer, 41–51.

[45] Mark Newman. 2018. *Networks*. Oxford university press.

[46] Kazuya Okamoto, Wei Chen, and Xiang-Yang Li. 2008. Ranking of Closeness Centrality for Large-Scale Social Networks. In *FAW (Lecture Notes in Computer Science, Vol. 5059)*. Springer, 186–195.

[47] Pan Peng, Daniel Lopatta, Yuichi Yoshida, and Gramoz Goranci. 2021. Local Algorithms for Estimating Effective Resistance. In *KDD*. 1329–1338.

[48] Yusuf Yigit Pilavci. 2022. *Wilson's Algorithm for Randomized Linear Algebra*. Ph. D. Dissertation. Université Grenoble Alpes [2020-....].

[49] Yusuf Yigit Pilavci, Pierre-Olivier Amblard, Simon Barthelmé, and Nicolas Tremblay. 2020. Smoothing Graph Signals via Random Spanning Forests. In *ICASSP*. 5630–5634.

[50] Yusuf Yigit Pilavci, Pierre-Olivier Amblard, Simon Barthelmé, and Nicolas Tremblay. 2021. Graph Tikhonov Regularization and Interpolation Via Random Spanning Forests. *IEEE Trans. Signal Inf. Process. over Networks* (2021), 359–374.

[51] Jim Pitman and Wenpin Tang. 2018. Tree formulas, mean first passage times and Kemeny's constant of a Markov chain. *Bernoulli* 24, 3 (2018), 1942 – 1972.

[52] Jiezhong Qiu, Laxman Dhulipala, Jie Tang, Richard Peng, and Chi Wang. 2021. LightNE: A Lightweight Graph Processing System for Network Embedding. In *SIGMOD*. ACM, 2281–2289.

[53] Luís MS Russo, Andreia Sofia Teixeira, and Alexandre P Francisco. 2018. Linking and cutting spanning trees. *Algorithms* 11, 4 (2018), 53.

[54] Aaron Schild. 2018. An almost-linear time algorithm for uniform random spanning tree generation. In *STOC*. 214–227.

[55] Jieming Shi, Nikos Mamoulis, Dingming Wu, and David W. Cheung. 2014. Density-based place clustering in geo-social networks. In *SIGMOD*. ACM, 99–110.

[56] Kumar Sricharan and Kamalika Das. 2014. Localizing anomalous changes in time-evolving graphs. In *SIGMOD*. ACM, 1347–1358.

[57] Piet Van Mieghem, Karel Devriendt, and H Cetinay. 2017. Pseudoinverse of the Laplacian and best spreader node in a network. *Physical Review E* 96, 3 (2017), 032311.

[58] Saurabh Verma and Zhi-Li Zhang. 2017. Hunt For The Unique, Stable, Sparse And Fast Feature Learning On Graphs. In *NIPS*. 88–98.

[59] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *KDD*. 505–514.

[60] Xiangrong Wang, Johan LA Dubbeldam, and Piet Van Mieghem. 2017. Kemeny's constant and the effective graph resistance. *Linear Algebra Appl.* 535 (2017), 231–244.

[61] Zhifang Wang, Anna Scaglione, and Robert J Thomas. 2010. Electrical centrality measures for electric power grid vulnerability analysis. In *49th IEEE conference on decision and control (CDC)*. IEEE, 5792–5797.

[62] David Bruce Wilson. 1996. Generating random spanning trees more quickly than the cover time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 296–303.

[63] 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.

[64] Wanyue Xu, Yibin Sheng, Zuobai Zhang, Haibin Kan, and Zhongzhi Zhang. 2020. Power-Law Graphs Have Minimal Scaling of Kemeny Constant for Random Walks. In *WWW*. 46–56.

[65] Renchi Yang and Jing Tang. 2023. Efficient Estimation of Pairwise Effective Resistance. *Proc. ACM Manag. Data* 1, 1 (2023), 16:1–16:27.
[66] Hongzhi Yin, Bin Cui, Jing Li, Junjie Yao, and Chen Chen. 2012. Challenging the Long Tail Recommendation. *VLDB* 5, 9 (2012), 896–907.
[67] Xiaohan Zhao, Adelbert Chang, Atish Das Sarma, Haitao Zheng, and Ben Y. Zhao. 2013. On the Embeddability of Random Walk Distances. *VLDB* 6, 14 (2013), 1690–1701.
[68] Bo Zhou and Nenad Trinajstić. 2008. A note on Kirchhoff index. *Chemical Physics Letters* 455, 1-3 (2008), 120–123.

A PSEUDO-CODE OF THE ALGORITHMS FOR KC COMPUTATION

Algorithm details of SpanTree **(KC).** The pseudo-code of the algorithm SpanTree (KC) is outlined in Algorithm 3. This algorithm is very similar to Algorithm 1. The difference is that the current flows we send on spanning trees are different (Line 5-7), and in the case of approximating Kemeny constant, we only need the (weighted) sum of the voltages instead of each one (Line 9-14). Finally, the two terms x_1 and x_2 are simply combined (Line 15). Since $E[x_1] = Tr((I - P_v)^{-1})$, $E[x_2] = \frac{1}{2m}\vec{d}_v^T L_v^{-1}\vec{d}_v$, we have $E[x_1 - x_2] = \kappa(G)$. After sampling spanning trees via the Wilson algorithm, we send electrical flows on spanning trees (Line 5-7), and in the case of approximating Kemeny constant, we update the (weighted) sum of the voltages accordingly (Line 9-14). Finally, the two terms x_1 and x_2 are simply combined (Line 15). Since $E[x_1] = Tr((I - P_v)^{-1})$, $E[x_2] = \frac{1}{2m}\vec{d}_v^T L_v^{-1}\vec{d}_v$, we have $E[x_1 - x_2] = \kappa(G)$.

Algorithm details of LEWalk **(KC).** The pseudo-code of the algorithm LEWalk (KC) is outlined in Algorithm 4. For each sample, it first simulates a loop-erased walk with root v and record the walk length y_1 (Line 2). We have $E[y_1] = Tr((I - P_v)^{-1})$. Then, it simulates a v-absorbed walk from a node sampled from the stationary distribution, record the walk length y_2 (Line 3). We have $E[y_2] = \frac{1}{2m}\vec{d}_v^T L_v^{-1}\vec{d}_v$. After that, we can update the approximated Kemeny constant $\tilde{\kappa}(G)$ accordingly (Line 4). Since $Tr(\mathcal{L}^\dagger) = Tr((I - P_v)^{-1}) - \frac{1}{2m}\vec{d}_v^T L_v^{-1}\vec{d}_v$, we have $E[y_1 - y_2] = Tr(\mathcal{L}^\dagger)$.

B MISSING PROOFS
B.1 Proof of Theorem 3.1

PROOF. Since $L\vec{1} = 0$, every column and row of L sums up to 0. Thus, the v-th row of L can be represented as $[-L_v\vec{1}; \vec{1}^T L_v\vec{1}]^T$. As a result, L can also be written as $L = \begin{pmatrix} I \\ -\vec{1}^T \end{pmatrix} L_v \begin{pmatrix} I & -\vec{1} \end{pmatrix}$. It is easy to verify that the matrix satisfies the four properties of the unique Moore-Penrose pseudo-inverse: (in the following formulas, the dimension of all-one matrix and identity matrix are adapted to their representations)

(i) $(L^\dagger L)^T = L^\dagger L$; (ii) $(LL^\dagger)^T = LL^\dagger$:

$$L^\dagger L = \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T \\ -\frac{1}{n}\vec{1}^T \end{pmatrix} L_v^{-1} \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T & -\frac{1}{n}\vec{1} \end{pmatrix} \begin{pmatrix} I \\ -\vec{1}^T \end{pmatrix} L_v \begin{pmatrix} I & -\vec{1} \end{pmatrix}$$

$$= \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T \\ -\frac{1}{n}\vec{1}^T \end{pmatrix} \begin{pmatrix} I & -\vec{1} \end{pmatrix} = I - \frac{1}{n}\vec{1}\vec{1}^T,$$

$$LL^\dagger = \begin{pmatrix} I \\ -\vec{1}^T \end{pmatrix} L_v \begin{pmatrix} I & -\vec{1} \end{pmatrix} \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T \\ -\frac{1}{n}\vec{1}^T \end{pmatrix} L_v^{-1} \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T & -\frac{1}{n}\vec{1} \end{pmatrix}$$

$$= \begin{pmatrix} I \\ -\vec{1}^T \end{pmatrix} \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T & -\frac{1}{n}\vec{1} \end{pmatrix} = I - \frac{1}{n}\vec{1}\vec{1}^T.$$

Since $LL^\dagger = L^\dagger L = I - \frac{1}{n}\vec{1}\vec{1}^T$, the conditions (i) and (ii) are both satisfied;

Algorithm 3: SpanTree (KC)

Input: Graph G, a landmark node v, sample size ω
Output: $\tilde{\kappa}(G)$ as an unbiased estimation of $\kappa(G)$

1 Pre-compute a BFS tree $\mathcal{T}_{\text{BFS}}$ with root v;
2 **for** $i = 1 : \omega$ **do**
3    $x_1 \leftarrow 0$, $x_2 \leftarrow 0$;
4   Uniformly sample a spanning tree $\mathcal{T}$ (using the Wilson algorithm);
5   **for** *each* $u \in V$ **do**
6   **for** *each edge* $e = (e_1, e_2) \in \mathcal{P}_\mathcal{T}[u]$ **do**
7          $\text{ps}[e_1] \leftarrow \text{ps}[e_1] + \frac{d_u}{2m}$;   // Compute path support ps in $\mathcal{T}$;
8   Do DFS on $\mathcal{T}$ from node v, compute the visit time $\text{vis}[u]$ and the finish time $\text{fin}[u]$ for all nodes $u \in V$;
9   **for** *each* $u \in V$ **do**
10   **for** *each edge* $e = (e_1, e_2) \in \mathcal{P}_{\mathcal{T}_{\text{BFS}}}[u]$ **do**
11   **if** $(e_1, e_2) \in \mathcal{T}$ and $[\text{vis}[e_1], \text{fin}[e_1]] \subseteq [\text{vis}[u], \text{fin}[u]]$ **then**
12            $x_1 \leftarrow x_1 + \frac{d_u}{\omega}$, $x_2 \leftarrow x_2 + \frac{\text{ps}[e_1]d_u}{\omega}$;
13   **if** $(e_2, e_1) \in \mathcal{T}$ and $[\text{vis}[e_2], \text{fin}[e_2]] \subseteq [\text{vis}[u], \text{fin}[u]]$ **then**
14            $x_1 \leftarrow x_1 - \frac{d_u}{\omega}$, $x_2 \leftarrow x_2 - \frac{\text{ps}[e_2]d_u}{\omega}$;
15 $\tilde{\kappa}(G) \leftarrow x_1 - x_2$;
16 **return** $\tilde{\kappa}(G)$ as an unbiased estimation of $\kappa(G)$;

Algorithm 4: LEWalk (KC)

Input: Graph G, a landmark node v, sample size ω
Output: $\tilde{\kappa}(G)$ as an unbiased estimation of $\kappa(G)$

1 **for** $i = 1 : \omega$ **do**
2   Run a loop-erased random walk with root v, record the walk length y_1;
3   Sample a node s from the stationary distribution $\boldsymbol{\pi}$, run a v-absorbed walk from s, record the walk length y_2;
4    $\tilde{\kappa}(G) \leftarrow \tilde{\kappa}(G) + \frac{y_1}{\omega} - \frac{y_2}{\omega}$;
5 **return** $\tilde{\kappa}(G)$;

(iii) $LL^\dagger L = L$:

$$LL^\dagger L = \begin{pmatrix} I \\ -\vec{1}^T \end{pmatrix} L_v \begin{pmatrix} I & -\vec{1} \end{pmatrix} \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T \\ -\frac{1}{n}\vec{1}^T \end{pmatrix} L_v^{-1} \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T & -\frac{1}{n}\vec{1} \end{pmatrix}$$

$$\begin{pmatrix} I \\ -\vec{1}^T \end{pmatrix} L_v \begin{pmatrix} I & -\vec{1} \end{pmatrix}$$

$$= \begin{pmatrix} I \\ -\vec{1}^T \end{pmatrix} L_v \begin{pmatrix} I & -\vec{1} \end{pmatrix} = L;$$

(iv) $L^\dagger L L^\dagger = L^\dagger$:

$$L^\dagger L L^\dagger = \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T \\ -\frac{1}{n}\vec{1}^T \end{pmatrix} L_v^{-1} \left(I - \tfrac{1}{n}\vec{1}\vec{1}^T \quad -\tfrac{1}{n}\vec{1}\right) \begin{pmatrix} I \\ -\vec{1}^T \end{pmatrix} L_v \left(I \quad -\vec{1}\right)$$

$$\begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T \\ -\frac{1}{n}\vec{1}^T \end{pmatrix} L_v^{-1} \left(I - \tfrac{1}{n}\vec{1}\vec{1}^T \quad -\tfrac{1}{n}\vec{1}\right)$$

$$= \begin{pmatrix} I - \frac{1}{n}\vec{1}\vec{1}^T \\ -\frac{1}{n}\vec{1}^T \end{pmatrix} L_v^{-1} \left(I - \tfrac{1}{n}\vec{1}\vec{1}^T \quad -\tfrac{1}{n}\vec{1}\right) = L^\dagger.$$

As a result, the matrix defined satisfies four properties of Moore-Penrose of L. Since the Moore-Penrose inverse is unique, the theorem is established. □

B.2 Proof of Theorem 3.2

PROOF. Since $\mathcal{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$, each element of $\mathcal{L}$ can be represented as $(\mathcal{L})_{st} = \frac{(L)_{st}}{\sqrt{d_s d_t}}$. The s-th element of $\mathcal{L}\sqrt{\pi}$ is $\mathcal{L}\sqrt{\pi}(s) = \sum_{u \in V} \frac{(L)_{su}}{\sqrt{d_s d_u}} \sqrt{\frac{d_u}{2m}} = 0$. Thus, we have $\mathcal{L}\sqrt{\pi} = 0$. The v-th row of $\mathcal{L}$ can be represented as $[-\frac{\mathcal{L}_v \sqrt{\pi_v}}{\pi(v)}; \quad \frac{1}{\pi(v)} \sqrt{\pi_v}^T \mathcal{L}_v \sqrt{\pi_v}]^T$. As a result, $\mathcal{L} = \begin{pmatrix} I \\ -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}^T \end{pmatrix} \mathcal{L}_v \left(I \quad -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}\right)$. Next, we verify that the matrix satisfies four properties of the unique Moore-Penrose pseudo-inverse: (in the following formulas, the dimension of identity matrix are adapted to their representations)

(i) $(\mathcal{L}^\dagger \mathcal{L})^T = \mathcal{L}^\dagger \mathcal{L}$; (ii) $(\mathcal{L}\mathcal{L}^\dagger)^T = \mathcal{L}\mathcal{L}^\dagger$:

$$\mathcal{L}^\dagger \mathcal{L} = \begin{pmatrix} I - \sqrt{\pi_v}\sqrt{\pi_v}^T \\ -\sqrt{\pi(v)}\sqrt{\pi_v}^T \end{pmatrix} \mathcal{L}_v^{-1} \left(I - \sqrt{\pi_v}\sqrt{\pi_v}^T \quad -\sqrt{\pi(v)}\sqrt{\pi_v}\right)$$

$$\begin{pmatrix} I \\ -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}^T \end{pmatrix} \mathcal{L}_v \left(I \quad -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}\right)$$

$$= \begin{pmatrix} I - \sqrt{\pi_v}\sqrt{\pi_v}^T \\ -\sqrt{\pi(v)}\sqrt{\pi_v}^T \end{pmatrix} \left(I \quad -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}\right) = I - \sqrt{\pi}\sqrt{\pi}^T,$$

$$\mathcal{L}\mathcal{L}^\dagger = \begin{pmatrix} I \\ -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}^T \end{pmatrix} \mathcal{L}_v \left(I \quad -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}\right) \begin{pmatrix} I - \sqrt{\pi_v}\sqrt{\pi_v}^T \\ -\sqrt{\pi(v)}\sqrt{\pi_v}^T \end{pmatrix}$$

$$\mathcal{L}_v^{-1} \left(I - \sqrt{\pi_v}\sqrt{\pi_v}^T \quad -\sqrt{\pi(v)}\sqrt{\pi_v}\right)$$

$$= \begin{pmatrix} I \\ -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}^T \end{pmatrix} \left(I - \sqrt{\pi_v}\sqrt{\pi_v}^T \quad -\sqrt{\pi(v)}\sqrt{\pi_v}\right)$$

$$= I - \sqrt{\pi}\sqrt{\pi}^T$$

Since $\mathcal{L}\mathcal{L}^\dagger = \mathcal{L}^\dagger \mathcal{L} = I - \sqrt{\pi}\sqrt{\pi}^T$, the conditions (i) and (ii) are both satisfied;

(iii) $\mathcal{L}\mathcal{L}^\dagger \mathcal{L} = \mathcal{L}$:

$$\mathcal{L}\mathcal{L}^\dagger \mathcal{L} = \begin{pmatrix} I \\ -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}^T \end{pmatrix} \mathcal{L}_v \left(I \quad -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}\right) \begin{pmatrix} I - \sqrt{\pi_v}\sqrt{\pi_v}^T \\ -\sqrt{\pi(v)}\sqrt{\pi_v}^T \end{pmatrix}$$

$$\mathcal{L}_v^{-1} \left(I - \sqrt{\pi_v}\sqrt{\pi_v}^T \quad -\sqrt{\pi(v)}\sqrt{\pi_v}\right) \begin{pmatrix} I \\ -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}^T \end{pmatrix}$$

$$\mathcal{L}_v \left(I \quad -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}\right) \begin{pmatrix} I - \sqrt{\pi_v}\sqrt{\pi_v}^T \\ -\sqrt{\pi(v)}\sqrt{\pi_v}^T \end{pmatrix}$$

$$= \begin{pmatrix} I \\ -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}^T \end{pmatrix} \mathcal{L}_v \left(I \quad -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}\right) \begin{pmatrix} I - \sqrt{\pi_v}\sqrt{\pi_v}^T \\ -\sqrt{\pi(v)}\sqrt{\pi_v}^T \end{pmatrix}$$

$$= \mathcal{L};$$

(iv) $\mathcal{L}^\dagger \mathcal{L}\mathcal{L}^\dagger = \mathcal{L}^\dagger$:

$$\mathcal{L}^\dagger \mathcal{L}\mathcal{L}^\dagger = \begin{pmatrix} I - \sqrt{\pi_v}\sqrt{\pi_v}^T \\ -\sqrt{\pi(v)}\sqrt{\pi_v}^T \end{pmatrix} \mathcal{L}_v^{-1} \left(I - \sqrt{\pi_v}\sqrt{\pi_v}^T \quad -\sqrt{\pi(v)}\sqrt{\pi_v}\right)$$

$$\begin{pmatrix} I \\ -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}^T \end{pmatrix} \mathcal{L}_v \left(I \quad -\frac{1}{\sqrt{\pi(v)}}\sqrt{\pi_v}\right) \begin{pmatrix} I - \sqrt{\pi_v}\sqrt{\pi_v}^T \\ -\sqrt{\pi(v)}\sqrt{\pi_v}^T \end{pmatrix}$$

$$\mathcal{L}_v^{-1} \left(I - \sqrt{\pi_v}\sqrt{\pi_v}^T \quad -\sqrt{\pi(v)}\sqrt{\pi_v}\right)$$

$$= \begin{pmatrix} I - \sqrt{\pi_v}\sqrt{\pi_v}^T \\ -\sqrt{\pi(v)}\sqrt{\pi_v}^T \end{pmatrix} \mathcal{L}_v^{-1} \left(I - \sqrt{\pi_v}\sqrt{\pi_v}^T \quad -\sqrt{\pi(v)}\sqrt{\pi_v}\right)$$

$$= \mathcal{L}^\dagger.$$

As a result, the matrix defined satisfies four properties of Moore-Penrose of $\mathcal{L}$. Since the Moore-Penrose inverse is unique, the theorem is established. □

B.3 Proof of Lemma 3.6

PROOF. According to Lemma 3.5, $(L_v^{-1})_{su}$ is the voltage at node u in an electrical network that unit current flow comes in through node s and unit current flow comes out through node v. In such an electrical network, let $I(s, v, e_1, e_2)$ denote the current flow along the edge (e_1, e_2). According to Kirchhoff's current law and Ohm's law, the voltage $(L_v^{-1})_{su} = \sum_{(e_1, e_2) \in \mathcal{P}^{u,v}} I(s, v, e_1, e_2)$. It remains to prove that the current flows through (e_1, e_2) is $\frac{|\mathcal{T}_{e_1,e_2}^{s,v}| - |\mathcal{T}_{e_2,e_1}^{s,v}|}{|\mathcal{T}|}$. According to Lemma 3.5, the current is $(L_v^{-1})_{se_1} - (L_v^{-1})_{se_2}$. According to Lemma 3.4, the current can be written as number of 2-spanning forests $\frac{|\mathcal{F}_{v|s,e_1}| - |\mathcal{F}_{v|s,e_2}|}{|\mathcal{T}|}$. We can partition $\mathcal{F}_{v|s,e_1}$ into two sets: (i) e_2 in the component that v belongs to ($\mathcal{F}_{v,e_2|s,e_1}$); and (ii) e_2 in the component that s, e_1 belong to ($\mathcal{F}_{v|s,e_1,e_2}$). We have $|\mathcal{F}_{v|s,e_1}| - |\mathcal{F}_{v|s,e_2}| = |\mathcal{F}_{v,e_2|s,e_1}| + |\mathcal{F}_{v|s,e_1,e_2}| - (|\mathcal{F}_{v,e_1|s,e_2}| + |\mathcal{F}_{v|s,e_1,e_2}|) = |\mathcal{F}_{s,e_1|e_2,v}| - |\mathcal{F}_{s,e_2|e_1,v}|$. Consider the number of 2-spanning trees $|\mathcal{F}_{s,e_1|e_2,v}|$ that s, e_1 are in one component, e_2, v are in the other component. There is a bijection between the spanning tree set $\mathcal{T}_{e_2,e_1}^{s,v}$ and the 2-spanning forest set $\mathcal{F}_{s,e_1|e_2,v}$. This is because we can add (e_1, e_2) in a 2-spanning forest in $\mathcal{F}_{s,e_1|e_2,v}$, then we will get a spanning tree in $\mathcal{T}_{e_2,e_1}^{s,v}$, and the reverse is also true. As a result, we prove that the current is $\frac{|\mathcal{F}_{v|s,e_1}| - |\mathcal{F}_{v|s,e_2}|}{|\mathcal{T}|}$, which completes the proof. □

B.4 Proof of Corollary 1

PROOF. According to Lemma 3.5 and Lemma 3.6, $(L_v^{-1})_{su}$ is the voltage at node u in an electrical network that unit current flow comes in through node s and unit current flow comes out through node v. In such an electrical network, let $I(s, v, e_1, e_2)$ denote the current flow along the edge (e_1, e_2). According to Kirchhoff's current law and Ohm's law, the voltage $(L_v^{-1})_{su} = \sum_{(e_1, e_2) \in \mathcal{P}^{u,v}} I(s, v, e_1, e_2)$, and $I(s, v, e_1, e_2) = \frac{|\mathcal{T}_{e_1,e_2}^{s,v}| - |\mathcal{T}_{e_2,e_1}^{s,v}|}{|\mathcal{T}|}$. Since if we send flows from s to v on a spanning tree T, then there is a current flow passes through (e_1, e_2) if and only if $T \in \mathcal{T}_{e_1,e_2}^{s,v}$. The theorem is established because any electrical system is a linear combination of a simple electrical system in Lemma 3.5. □

B.5 Proof of Lemma 3.7

PROOF. Given the stack representation of LE_v. Since the stacks are independent of the node ordering, each node can be the first node, while the same stack representation will be obtained. When we select u as the first node to construct $\mathcal{P}_{\mathsf{LE}_v}[u]$, the distribution of $\mathcal{P}_{\mathsf{LE}_v}[u]$ is the same as the distribution of $\mathcal{P}_v[u]$, where $\mathcal{P}_v[u]$

denotes the path of a v-absorbed walk trajectory from u. From Lemma 3.3, we have $\tilde{\tau}_{\mathsf{LE}_v}[s, u] = \tilde{\tau}_v[s, u] = \frac{\tau_v[s,u]}{d_u} = (L_v^{-1})_{su}$. When a random walk from u reaches v in LE_v, LE_v will never pass u again. Thus $\tilde{\tau}_{\mathsf{LE}_v}[s, s] = (L_v^{-1})_{ss}$ is also the expected number of passes of u in the whole trajectory of LE_v. □

B.6 Proof of Lemma 4.1

PROOF. Consider the electrical system as a linear combination of n electrical systems that $\frac{1}{n}$ flow comes in through node $u \in V$, comes out through node v. According to Lemma 3.5, the voltage at node a node $s \neq v$ is $\frac{1}{n}(L_v^{-1})_{us}$. When combining n such electrical systems, the voltage at node u is $\mathbf{x}(u) = \frac{1}{n}\sum_{s \in V}(L_v^{-1})_{us} = \frac{1}{n}\mathbf{e}_u^T L_v^{-1}\vec{1}$, this is based on the setting $\mathbf{x}(v) = 0$. Since the column of $L^\dagger$ sums up to 0, when setting the voltages to satisfy all voltages sum up to 0, $-\mathbf{x}(v)$ then is the average of such voltages. We have $\mathbf{x}(v) = -\frac{1}{n^2}\vec{1}^T L_v^{-1}\vec{1} = -(L^\dagger)_{vv}$ (based on Equation (4)). The voltage at node u is $\mathbf{x}(u) = -\frac{1}{n^2}\vec{1}^T L_v^{-1}\vec{1} + \frac{1}{n}\mathbf{e}_s^T L_v^{-1}\vec{1} = -(L^\dagger)_{vu}$ (based on Equation (3)). According to Ohm's law, the current flow along edge (e_1, e_2) is the difference of voltages, $(L^\dagger)_{ve_2} - (L^\dagger)_{ve_1}$. Finally, the energy of the system is the product of current and voltage, which is $\sum_{u \in V}\frac{1}{n}\frac{1}{n}\mathbf{e}_u^T L_v^{-1}\vec{1} = \frac{1}{n^2}\vec{1}^T L_v^{-1}\vec{1} = (L^\dagger)_{vv}$. Notice that the energy also equals the square sum of the currents on all edges. □

B.7 Proof of Lemma 4.2

PROOF. Precomputing a BFS tree costs time $O(n + m)$. In each sample, sampling a spanning tree using the Wilson algorithm costs time $Tr((I - P_v)^{-1})$. Suppose Δ_G is the diameter of G, the complexity of computing the path support can be bounded by $O(n\Delta_G)$. Then, a DFS on the spanning tree costs time $O(n)$. traverse each in the path $\mathcal{P}_b[u]$ for all nodes $u \in V$ also needs time $O(n\Delta_G)$. In practice, as we have discussed in Section 3.3, $O(Tr(I - P_v)^{-1})$ can be as small as $O(n)$. Since real-world graphs often has an $O(\log n)$ diameter (small world [45]), the complexity can be simplified to $O(n \log n)$. As a result, the time complexity of the algorithm is $O(n \log n)$ on real-life large graphs. □

B.8 Proof of Lemma 4.3

To prove Lemma 4.3, we need to introduce the well-known Hoeffding's inequality first.

LEMMA B.1. *(Hoeffding's inequality) Let $X_1, \cdots, X_\omega$ be independent random variables in $[a, b]$, where $-\infty < a \leq b < \infty$, for any $0 < \epsilon < 1$, we have*

$$Pr[|\frac{1}{\omega}\sum_{i=1}^{\omega} X_i - E[X]| \geq \epsilon] \leq 2exp(-\frac{2\epsilon^2\omega}{(b-a)^2}).$$

PROOF. Let X_1, X_2 and X_3 be the random variables which represent the amount we update $\mathbf{x}_1(u)$, $\mathbf{x}_2(u)$ and $\mathbf{x}_2(v)$ in Algorithm 1. We have $(L^\dagger)_{uu} = E[X_1 + 2X_2 - X_3]$ and $\widetilde{(L^\dagger)}_{uu} = \frac{X_1}{\omega} + 2\frac{X_2}{\omega} - \frac{X_3}{\omega}$. Since the length of the path in the BFS tree can be bounded by Δ_G, $|X_1| \leq \Delta_G$, $|X_3| \leq \Delta_G$. At the same time, the path support we computed in Line 6-8 is at most 1, we also have $|X_2| \leq \Delta_G$. Thus, we have $|X_1 + 2X_2 - X_3| \leq 4\Delta_G$. Let $X = X_1 + 2X_2 - X_3$, we have $E[X] = (L^\dagger)_{uu}$ and $\widetilde{(L^\dagger)}_{uu} = \frac{\sum_{i=1}^{\omega}X_{(i)}}{\omega}$. By applying the Hoeffding's inequality, if $\omega \geq \frac{32\Delta_G^2\log(\frac{2n}{p_f})}{\epsilon^2}$, we can obtain that $Pr(|\frac{\sum_{i=1}^{\omega}X_{(i)}}{\omega} - (L^\dagger)_{uu}| \geq \epsilon) \leq 2exp(-\frac{2\epsilon^2\omega}{(8\Delta_G)^2}) \leq 2exp(-\frac{2\epsilon^2\frac{32\Delta_G^2\log(\frac{2n}{p_f})}{\epsilon^2}}{64\Delta_G^2}) \leq \frac{p_f}{n}$. By union

bound, the probability that all nodes $u \in V$ satisfy such a condition is at least $1 - p_f$. □

B.9 Proof of Lemma 4.4

PROOF. According to Lemma 3.7, $(L_v^{-1})_{ui} = \tilde{\tau}_{\mathsf{LE}_v}[u, i] = \frac{\tau_{\mathsf{LE}_v}[u,i]}{d_i}$ is the expected normalized number of passes to a node i in a loop-included path $\mathcal{P}_{\mathsf{LE}_v}[u]$. Since a path length equals the sum of the number of passes to each node $u \in V$, $\mathbf{e}_u^T L_v^{-1}\vec{1} = \sum_{i \in V, i \neq v}(L_v^{-1})_{ui} = \sum_{i \in V, i \neq v}\tilde{\tau}_{\mathsf{LE}_v}[u, i] = E[|\mathcal{P}_{\mathsf{LE}_v}[u]|_n]$ is the expected normalized length of the loop-included path. □

B.10 Proof of Lemma 4.5

PROOF. According to Lemma 3.7, in a loop-erased walk LE_v, $(L_v^{-1})_{ui} = (L_v^{-1})_{iu} = \tilde{\tau}_{\mathsf{LE}_v}[i, u] = \frac{\tau_{\mathsf{LE}_v}[i,u]}{d_u}$ is the expected normalized number of passes to a node u in a loop-included path $\mathcal{P}_{\mathsf{LE}_v}[i]$ from i. Since $\tilde{\tau}_{\mathsf{LE}_v}[s, s] = 0$, the expected normalized number of passes of node u in a loop-included path $\mathcal{P}_{\mathsf{LE}_v}[s]$ from a node s sampled uniformly from V is $\sum_{s \in V}Pr[s = u]\tilde{\tau}_{\mathsf{LE}_v}[s, u] = \sum_{s \in V, s \neq v}Pr[s = u]\tilde{\tau}_{\mathsf{LE}_v}[s, u] = \sum_{s \in V, s \neq v}\frac{1}{n}(L_v^{-1})_{su} = \frac{1}{n}\mathbf{e}_u^T L_v^{-1}\vec{1}$. □

B.11 Proof of Lemma 4.6

PROOF. In the first phase of sampling loop-erased walks (Line 3-14 in Algorithm 2), the time complexity is $Tr(I - P_v)^{-1}$. In the second phase of sampling v-absorbed walks (Line 15-19 in Algorithm 2), the expected length of a v-absorbed walk is $\sum_{u \in V}Pr[u = s]h(s, v) = \frac{1}{n}\vec{1}^T(I - P_v)^{-1}\vec{1}$. In real life graphs, there often exists an easy-to-hit node v (e.g., the highest-degree node). For such v, $\frac{1}{n}\vec{1}^T(I - P_v)^{-1}\vec{1}$ is much smaller than $Tr(I - P_v)^{-1}$. As a result, the time complexity of LEWalk is $O(Tr(I - P_v)^{-1})$ for each sample. □

B.12 Proof of Lemma 5.1

PROOF. According to Equation (6), $(\mathcal{L}^\dagger)_{ss} = \pi(s)(2m\mathbf{e}_s^T L_v^{-1}\mathbf{e}_s - 2\mathbf{e}_s^T L_v^{-1}\vec{\mathbf{d}}_v + \frac{1}{2m}\vec{\mathbf{d}}_v^T L_v^{-1}\vec{\mathbf{d}}_v)$ for $s \neq v$; According to Equation (8), $(\mathcal{L}^\dagger)_{vv} = \pi(v)(\frac{1}{2m}\vec{\mathbf{d}}_v^T L_v^{-1}\vec{\mathbf{d}}_v)$. Since $\pi(u) = \frac{d_u}{2m}$ and $(I - P_v)^{-1} = L_v^{-1}D_v$, the trace can be formulated as:

$$Tr(\mathcal{L}^\dagger) = \sum_{u \in V}(\mathcal{L}^\dagger)_{uu}$$
$$= \sum_{u \in V, u \neq v}2m\pi(u)\mathbf{e}_u^T L_v^{-1}\mathbf{e}_u - \sum_{u \in V, u \neq v}2\pi(u)\mathbf{e}_u^T L_v^{-1}\vec{\mathbf{d}}_v$$
$$\quad + \sum_{u \in V}\frac{\pi(u)}{2m}\vec{\mathbf{d}}_v^T L_v^{-1}\vec{\mathbf{d}}_v$$
$$= Tr((I - P_v)^{-1}) - \frac{1}{2m}\vec{\mathbf{d}}_v^T L_v^{-1}\vec{\mathbf{d}}_v$$
$$= Tr((I - P_v)^{-1}) - \frac{(\mathcal{L}^\dagger)_{vv}}{\pi(v)}.$$

□

B.13 Proof of Corollary 2

PROOF. $\frac{(\mathcal{L}^\dagger)_{vv}}{\pi(v)} = \frac{1}{2m}\vec{\mathbf{d}}_v^T L_v^{-1}\vec{\mathbf{d}}_v$. Consider the electrical system as a linear combination of n electrical systems that $\frac{d_u}{2m}$ flow comes in through node $u \in V$, comes out through node v. According to Lemma 3.5, the voltage at node a node $s \neq v$ is $\frac{d_u}{2m}(L_v^{-1})_{us}$. When combining n such electrical systems, according to Kirchhoff's current law and Ohm's law, the voltage at node s is $\mathbf{x}(s) = $
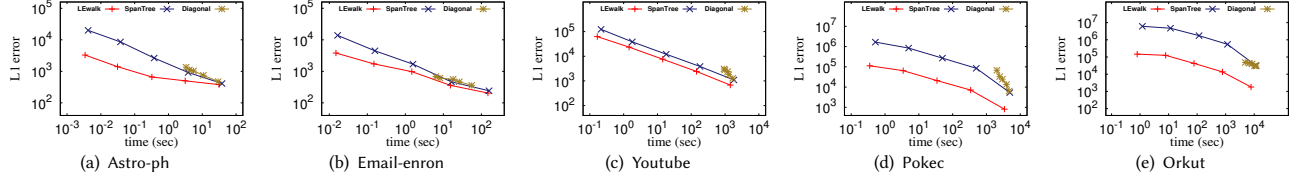
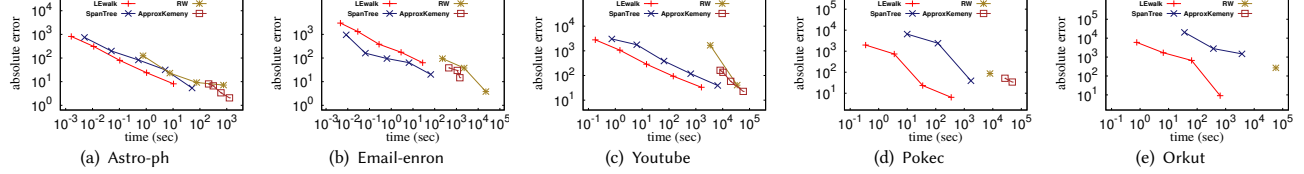**Figure 10: $L1$-error v.s. runtime of different algorithms for approximating electrical closeness centrality**

Figure 11: Absolute error v.s. runtime of different algorithms for approximating Kemeny constant
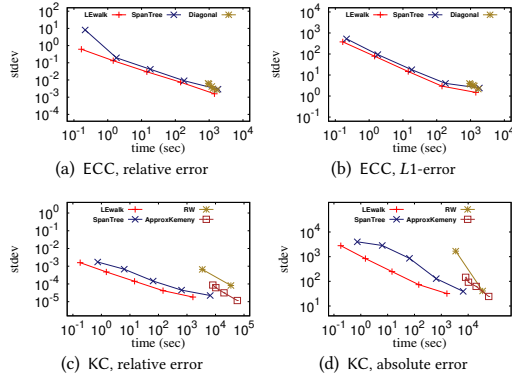

Figure 12: Standard deviation of error v.s. average runtime of different algorithms on Youtube
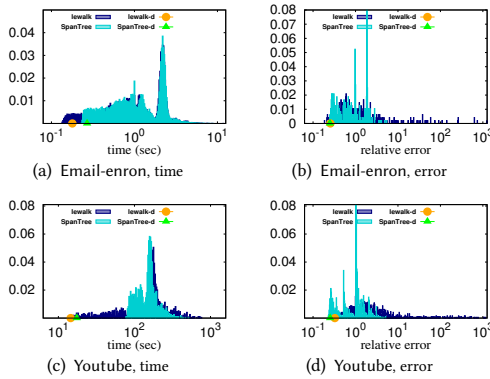

Figure 13: Distribution of performance of algorithms for different choices of landmark node (electrical closeness centrality)

$\frac{1}{2m}\sum_{u\in V}(L_v^{-1})_{su}d_u = \frac{1}{2m}\mathbf{e}_s^T L_v^{-1}\vec{\mathbf{d}}_v$, this is based on the setting $\mathbf{x}(v) = 0$. As a result, the weighted sum of the voltages is exactly $\sum_{s\in V}\mathbf{x}(s)d_s = \frac{1}{2m}\vec{\mathbf{d}}_v^T L_v^{-1}\vec{\mathbf{d}}_v$. Thus, the corollary is established. □

B.14 Proof of Lemma 5.2

PROOF. Let X_1 and X_2 be the random variables which represent the amount we update x_1 and x_2 in Algorithm 3 (Line 10-14). We

have $\kappa(G) = E[X_1 - X_2]$ and $\tilde{\kappa}(G) = \frac{X_1}{\omega} - \frac{X_2}{\omega}$. Notice that the length of the path $\mathcal{P}_{\mathcal{T}_{BFS}}$ in the BFS tree can be bounded by Δ_G. The BFS tree stores paths from each node $u \in V$ to the root. In the path from $\mathcal{P}_{\mathcal{T}_{BFS}}$, if an edge is sampled in a random spanning tree, it updates x_1 by d_u, we have $|X_1| \leq \sum_{u\in V} d_u\Delta_G = m\Delta_G$. At the same time, the path support $ps[u]$ we compute (Line 5-7) is at most 1, we also have $|X_2| \leq m\Delta_G$. Thus, we have $|X_1 - X_2| \leq 2m\Delta_G$. Let $X = X_1 - X_2$, we have $E[X] = \kappa(G)$ and $\tilde{\kappa}(G) = \frac{\sum_{i=1}^{\omega} X_{(i)}}{\omega}$. By applying the Hoeffding's inequality, if $\omega \geq \frac{8m^2\Delta_G^2 \log(\frac{2}{p_f})}{\epsilon^2}$, we can obtain that $Pr(|\frac{\sum_{i=1}^{\omega} X_{(i)}}{\omega} - \kappa(G)| \geq \epsilon) \leq 2exp(-\frac{2\epsilon^2\omega}{(4m\Delta_G)^2}) \leq 2exp(-\frac{2\epsilon^2 \frac{8m^2\Delta_G^2 \log(\frac{2}{p_f})}{\epsilon^2}}{16m^2\Delta_G^2}) \leq p_f$. Thus, the theorem is established. □

C ADDITIONAL EXPERIMENTAL RESULTS

C.1 The $L1$ error results for ECC computation

In this experiment, we evaluate the performance of algorithms for approximating ECC using the $L1$-error to evaluate the errors. Similar to the experiment in Section 6.2, we also plot the error v.s. time results for the two proposed algorithms SpanTree and LEWalk, as well as the SOTA method Diagonal [4]. For SpanTree and LEWalk, we vary the sample size ω from 10^0 to 10^5. For Diagonal, we vary ϵ from 0.9 to 0.3 following the settings in [4]. We evaluate the error of ECC vector by the $L1$-error which is defined as $\sum_{u\in V}|\tilde{\mathbf{c}}(u) - \mathbf{c}(u)|$. We run each algorithm for 100 times and report the average results. The results are shown in Fig. 10. As can be seen, SpanTree can achieve comparable and slightly better $L1$-error compared to Diagonal within a similar time. Note that SpanTree avoids the calling of highly-optimized Laplacian solvers as invoked in Diagonal. It is a "pure" spanning tree sampling algorithm which is much easier to implement. LEWalk can achieve a certain accuracy in terms of $L1$-error in much less time compared to SpanTree and Diagonal. These results demonstrate the high efficiency of our loop-erased walk sampling approach.

C.2 The absolute error results for KC computation

In this experiment, we further demonstrate the results of the performance evaluation of KC computation using the absolute error to evaluate the errors. Similar to the experiment in Section 6.2, we also plot the error v.s. time results for two proposed algorithms SpanTree and LEWalk, as well as two SOTA methods RW [37] and ApproxKemeny [64]. For SpanTree, LEWalk and RW, we vary the

sample size ω from 10^0 to 10^5, following a similar experimental setting in [37]. For ApproxKemeny, we vary the accuracy parameter ϵ from 0.9 to 0.3 following [64]. We measure the quality of results by the absolute error defined as $|\tilde{\kappa}(G) - \kappa(G)|$. We run each algorithm for 100 times and report the average results. We omit the results when the running time of an experiment exceeds 24 hours. The results are shown in Fig. 11. As can be seen, the results are similar to the relative error results presented in Section 6.2. Specifically, the running time of our algorithms are orders of magnitude faster than the SOTA algorithms. For the proposed algorithms, LEWalk is slightly faster than SpanTree. These results demonstrate that our loop-erased walk sampling based algorithms are very efficient in computing Kemeny constant.

C.3 Variance analysis of our algorithms.

The standard deviation of Monte Carlo algorithms is an important metric to evaluate the stability and, thus the quality of the results. In former experiments, we only report the average results. Here we evaluate the standard deviation of the proposed algorithms for both ECC approximation and KC approximation. We conduct the experiments following the former experimental settings. Since the running time of the algorithms is quite consistent (the standard deviation is very small), we plot the standard deviation of the error (maximum relative error and $L1$-error for ECC approximation; relative error and absolute error for KC approximation) v.s. average runtime of all the experiments. Fig. 12 shows the results on Youtube. Similar results can also be observed on the other datasets. We can see that as the time grows, the standard deviation becomes smaller together with the average error. In similar running time,

our algorithms have a smaller standard deviation compared to the SOTA algorithms.

C.4 Effect of landmark node v.

Here, we plot the runtime and error (relative error) distribution of the proposed electrical closeness centrality computation algorithms with different v. The results can be found in Fig. 13. We can see that the highest-degree node is almost the best choice for achieving short time and low error. The reason is that the highest-degree node is often easy-to-hit, thus the length of the loop-included path in LE_v may be short, leading to both low traversal cost and small variance. These results confirm our theoretical analysis in Section 3.

D ADDITIONAL RELATED WORK

Efficient sampling a spanning tree is a well-studied problem in theoretical computer science. Aldous-Broder algorithm [1] simulates random walks from a node until all nodes are covered. Russo et al. propose an algorithm [53] based on Makrov chain built over all spanning trees, but the algorithm requires complex data structure such as link-cut trees. For theoretical results, the algorithm [54] by Schild is the best with time complexity $O(m^{1+o(1)})$. For real-life large datasets, the Wilson algorithm that utilizing loop-erased walk sampling [62] is the most efficient algorithm, and it is chosen for spanning tree sampling algorithms in previous work [4, 31, 39]. The time complexity of the Wilson algorithm is $Tr((I - P_v)^{-1})$ [43]. Specifically, Liao et. al. [39] gives a local algorithm for sampling only part of spanning trees which runs in time $h(s, v) + h(t, v)$ if one only wants the path between s and t.