Nearly Linear Time Algorithm for Mean Hitting Times of Random Walks on a Graph[*]

Zuobai Zhang
Fudan University
Shanghai, China
17300240035@fudan.edu.cn

Wanyue Xu
Fudan University
Shanghai, China
xuwy@fudan.edu.cn

Zhongzhi Zhang[†]
Fudan University
Shanghai, China
zhangzz@fudan.edu.cn

ABSTRACT

For random walks on a graph, the mean hitting time H_j from a vertex i chosen from the stationary distribution to the target vertex j can be used as a measure of importance for vertex j, while the Kemeny constant K is the mean hitting time from a vertex i to a vertex j selected randomly according to the stationary distribution. Both quantities have found a large variety of applications in different areas. However, their high computational complexity limits their applications, especially for large networks with millions of vertices. In this paper, we first establish a connection between the two quantities, representing K in terms of H_j for all vertices. We then express both quantities in terms of quadratic forms of the pseudoinverse for graph Laplacian, based on which we develop an efficient algorithm that provides an approximation of H_j for all vertices and K in nearly linear time with respect to the edge number, with high probability. Extensive experiment results on real-life and model networks validate both the efficiency and accuracy of the proposed algorithm.

CCS CONCEPTS

• **Information systems → Data mining**; • **Theory of computation → Graph algorithms analysis**; **Approximation algorithms analysis**.

KEYWORDS

Random walk; hitting time; Kemeny constant; spectral algorithm; complex network; node centrality

ACM Reference Format:
Zuobai Zhang, Wanyue Xu, and Zhongzhi Zhang. 2020. Nearly Linear Time Algorithm for Mean Hitting Times of Random Walks on a Graph. In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM '20), February 3–7, 2020, Houston, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3336191.3371777

[*]The authors are with the Shanghai Key Laboratory of Intelligent Information Processing, School of Computer Science, Fudan University, Shanghai, 200433, China.
[†]Corresponding author.

1 INTRODUCTION

As a powerful tool and method, random walks have found broad applications in various aspects. Frequently cited examples include image segmentation [13], random algorithm design [33], collaborative recommendation [11], community detection [21], among others. A fundamental quantity related to random walks is hitting time [27], also called first-passage time [9]. For a random walk on a graph, the hitting time H_{ij} from a vertex i to another vertex j is the expected time for the walker starting from i to visit j for the first time. Hitting time is related to many problems and has been successfully applied to diverse areas, such as Hanoi problem with random move [39], query suggestion [29], and clustering algorithm [6].

Except for the intrinsic interest of hitting time itself and its direct applications, many other relevant quantities related to random walks are encoded in or expressed in terms of this crucial quantity, for example, absorbing random-walk centrality [28] (or Markov centrality [38]) and Kemeny constant [15], both of which are a partial average of hitting times. As the name implies, the absorbing random-walk centrality is a measure for the importance of vertices on a graph. For a vertex j, its absorbing random-walk centrality H_j is defined by $H_j = \sum_i \rho(i) H_{ij}$, where $\rho(\cdot)$ is the starting probability distribution over all vertices. Different from the shortest-path based centrality measures, random-walk based centrality metrics include the contributions from essentially all paths [31], and thus have a better discriminating power.

For random walks on a graph with n vertices, the Kemeny constant K is defined as the expected time for the walker starting from one vertex to second vertex selected randomly from the graph according to the stationary distribution $\boldsymbol{\pi} = (\pi_1, \pi_2, \cdots, \pi_n)^\top$ of the random walk, that is, $K = \sum_j \pi_j H_{ij}$. The Kemeny constant has also found a wealth of applications in different fields [15]. It can be utilized to gauge the efficiency of user navigation through the World Wide Web (WWW) [22]. Moreover, the Kemeny constant is related to the mixing rate of an irreducible Markov chain [23], by regarding it as the expected time to mixing of the Markov chain [14]. Recently, the Kemeny constant has been applied to measure the efficiency of robotic surveillance in network environments [32] and to characterize the noise robustness of a class of protocols for formation control [16].

Despite the wide range of applications of the absorbing random-walk centrality and the Kemeny constant, it is a computational challenge to obtain their exact values. By definition, both the absorbing random-walk centrality and the Kemeny constant are a partial average of some hitting times. However, the exact value of hitting time between any pair of vertices in a graph involves all eigenvalues and eigenvectors of (normalized) Laplacian matrix associated with the graph [25, 27], the computation complexity of which is the cube of the vertex number. Thus, for large realistic networks with millions of vertices, we cannot obtain their absorbing random-walk centrality and the Kemeny constant by resorting this straightforward method for computing hitting time. It is then of theoretical and practical interest to seek for alternative approximate approaches that scale to large networks.

In this paper, we propose a nearly linear time algorithm to address the computation issue for absorbing random-walk centrality H_j and Kemeny constant K in large networks. We focus on a special absorbing random-walk centrality $H_j = \sum_i \rho(i)H_{ij}$, with the starting probability distribution $\rho(\cdot)$ being the stationary distribution $\boldsymbol{\pi} = (\pi_1, \pi_2, \cdots, \pi_n)^\top$ of the random walk. In other words, we study $H_j = \sum_i \pi_i H_{ij}$, which has received considerable attention [4, 5, 36]. We first express K in terms of H_j for all vertices, and further express H_j and K in terms of quadratic forms of pseudoinverse of the Laplacian matrix. We then propose a fast algorithm to compute approximate H_j for all vertices and K for the whole graph in nearly linear time of the number of edges, based on the Johnson-Lindenstrauss lemma [1] and the Laplacian solver [8, 18, 20, 26, 34, 35]. Finally, we experimentally demonstrate that our algorithm is accurate and is significantly faster than the direct exact computation of related quantities according to their definitions.

2 PRELIMINARY

In this section, we give a brief introduction to some basic concepts about graphs, Laplacian matrix, resistance distance, random walks, hitting times and some quantities derived from hitting times.

2.1 Graph and Laplacian Matrix

Let $\mathcal{G} = (V, E, w)$ denote a connected undirected weighted graph or network, where V is the set of vertices, E is the set of edges, and $w : E \longrightarrow \mathbb{R}_+$ is the positive edge weight function, with w_e being the weight for edge e. Then, there are total $n = |V|$ vertices and $m = |E|$ edges in graph $\mathcal{G}$. We use $u \sim v$ to indicate that two vertices u and v are connected by an edge. Let $w_{\max}$ and $w_{\min}$ denote the maximum edge weight and minimum edge weight, respectively. Namely, $w_{\max} = \max_{e\in E} w_e$ and $w_{\min} = \min_{e\in E} w_e$.

Mathematically, the topological and weighted properties of a graph $\mathcal{G}$ are encoded in its generalized adjacency matrix $\boldsymbol{A}$ with the entry a_{ij} denoting the adjacency relation between vertices i and j. If vertices i and j are linked to each other by an edge e, then $a_{ij} = a_{ji} = w_e > 0$. Otherwise, $a_{ij} = a_{ji} = 0$ indicating that vertices i and j are not adjacent. In a weighted graph $\mathcal{G}$, the strength s_i of a vertex i is defined by $s_i = \sum_{j=1}^n a_{ij}$ [2]. The diagonal strength matrix of graph $\mathcal{G}$ is defined to be $\boldsymbol{S} = \mathrm{diag}(s_1, s_2, \dots, s_n)$, and the Laplacian matrix of $\mathcal{G}$ is $\boldsymbol{L} = \boldsymbol{S} - \boldsymbol{A}$.

Let $\boldsymbol{B} \in \mathbb{R}^{|E|\times|V|}$ be the incidence matrix of $\mathcal{G}$. For each edge e with two end vertices i and j, a direction is assigned arbitrarily. Let $\boldsymbol{b}_e^\top$ be the row of matrix $\boldsymbol{B}$ associated with edge e. Then the element b_{eu} at row corresponding to edge e and column corresponding to vertex u is defined as follows: $b_{eu} = 1$ if vertex u is the tail of edge e, $b_{eu} = -1$ if vertex u is the head of edge e, and $b_{eu} = 0$ otherwise. Let $\boldsymbol{e}_u$ be the u-th canonical basis of the space $\mathbb{R}^{|V|}$, then for an edge e connecting two vertices i and j, $\boldsymbol{b}_e$ can also be recast as $\boldsymbol{b}_e = \boldsymbol{e}_i - \boldsymbol{e}_j$. Let $\boldsymbol{W} \in \mathbb{R}^{|E|\times|E|}$ be a diagonal matrix with the diagonal entry (e, e) being w_e. Then the Laplacian matrix $\boldsymbol{L}$ of graph $\mathcal{G}$ can be written as $\boldsymbol{L} = \boldsymbol{B}^T\boldsymbol{W}\boldsymbol{B} = \sum_{e\in E} w_e \boldsymbol{b}_e \boldsymbol{b}_e^\top$.

The Laplacian matrix $\boldsymbol{L}$ is symmetric and positive semidefinite. All its eigenvalues are non-negative, with a unique zero eigenvalue. Let $0 = \lambda_1 < \lambda_2 \le \lambda_3 \le \cdots \le \lambda_n$ be the n eigenvalues of $\boldsymbol{L}$, and let u_i, $i = 1, 2, \dots, n$, be their corresponding mutually orthogonal unit eigenvectors. Then, $\boldsymbol{L}$ has the following spectral decomposition: $\boldsymbol{L} = \sum_{i=2}^n \lambda_i u_i u_i^\top$. It is easy to verify that $\lambda_n \le n w_{\max}$ [24]. Since $\boldsymbol{L}$ is not invertible, we use $\boldsymbol{L}^\dagger$ to denote its pseudoinverse, which can be written as $\boldsymbol{L}^\dagger = \sum_{i=2}^n \frac{1}{\lambda_i} u_i u_i^\top$. Let $\boldsymbol{\mathcal{J}}$ denote the matrix with all entries being ones. Then the pseudoinverse $\boldsymbol{L}^\dagger$ can also be recast as $\left(\boldsymbol{L} + \frac{1}{n}\boldsymbol{\mathcal{J}}\right)^{-1} - \frac{1}{n}\boldsymbol{\mathcal{J}}$ [12]. Note that for a general symmetric matrix, it shares the same null space as its Moore-Penrose generalized inverse [3]. Since the null space of null of $\boldsymbol{L}$ is $\boldsymbol{1}$, it turns out that $\boldsymbol{L}\boldsymbol{1} = \boldsymbol{L}^\dagger\boldsymbol{1} = \boldsymbol{0}$.

2.2 Electrical Network and Resistance Distance

For an arbitrary graph $\mathcal{G} = (V, E, w)$, we can define its corresponding electrical network $\bar{\mathcal{G}} = (V, E, r)$, which is obtained from $\mathcal{G}$ by considering edges as resistors and considering vertices as junctions between resistors [10]. The resistor of an associated edge e is $r_e = w_e^{-1}$. For graph $\mathcal{G}$, the resistance distance R_{ij} between two vertices i and j is defined as the effective resistance between i and j in the corresponding electrical network $\bar{\mathcal{G}}$ [17], which is equal to the potential difference between i and j when a unit current enters one vertex and leaves the other one.

For graph $\mathcal{G}$, the resistance distance R_{ij} between two vertices i and j can be expressed in terms of the elements of $\boldsymbol{L}^\dagger$ as [17]:

$$R_{ij} = \boldsymbol{L}^\dagger_{ii} + \boldsymbol{L}^\dagger_{jj} - 2\boldsymbol{L}^\dagger_{ij}. \tag{1}$$

Define $\boldsymbol{R}$ as the $n \times n$ resistance matrix of graph $\mathcal{G}$, whose entry at row i and column j represents the resistance distance R_{ij} between vertices i and j.

Lemma 2.1. *[37] Let $\mathcal{G} = (V, E, w)$ be a simple connected graph with n vertices. Then the sum of weight times resistance distance over all pairs of adjacent vertices in $\mathcal{G}$ satisfies*

$$\sum_{i\sim j\in E} w_{ij} R_{ij} = n - 1.$$

2.3 Random Walk on a Graph

For a graph $\mathcal{G}$, one can define a discrete-time random walk on it. At any time step, the walker starting from its current location i moves to vertex j with probability a_{ij}/s_i. If $\mathcal{G}$ is finite and non-bipartite, the random walk has a unique stationary distribution [25]

$$\boldsymbol{\pi} = (\pi_1, \pi_2, \cdots, \pi_n)^\top = \left(\frac{s_1}{s}, \frac{s_2}{s}, \cdots, \frac{s_n}{s}\right)^\top, \tag{2}$$

where s is the sum of strengths over all vertices, namely $s = \sum_{i=1}^n s_i = \sum_{i=1}^n \sum_{j=1}^n a_{ij}$.

A fundamental quantity for random walks is hitting time [9, 27]. The hitting time H_{ij} from vertex i to vertex j, is the expected number of jumps for a walker starting from i to visit j for the first time. There is an intimate relationship between hitting time and resistance distance [37].

Lemma 2.2. *Let $\mathcal{G}$ be a connected weighted graph with resistance matrix $\boldsymbol{R}$. Let H_{ij} be the hitting time from vertex i to vertex j. Then,*

$$H_{ij} = \frac{1}{2}\sum_{z=1}^{n} s_z(R_{ij} + R_{jz} - R_{iz}). \tag{3}$$

A lot of interesting quantities can be derived from hitting times. Here we only consider two quantities, the absorbing random-walk centrality [28] and the Kemeny constant [15].

For a vertex j in graph $\mathcal{G} = (V, E, w)$, its absorbing random-walk centrality H_j is defined as $H_j = \sum_i \rho(i)H_{ij}$, where $\rho(\cdot)$ is the starting probability distribution over all vertices in V. By definition, H_j is a weighted average of hitting times to vertex j. The smaller the value of H_j, the more important the vertex j. The random-walk based centrality has an obvious advantage over those shortest-path based centrality measures [31]. In this paper, we concentrate on a natural choice of $\rho(\cdot)$ by selecting the starting vertex from the stationary distribution $\boldsymbol{\pi}$. In this case, $H_j = \sum_i \pi_i H_{ij}$, which has been much studied [4, 5, 36]. In the following text, we call $H_j = \sum_i \pi_i H_{ij}$ *walk centrality* for short.

Another quantity we are concerned with is the Kemeny constant K. For a graph $\mathcal{G}$, its Kemeny constant K is defined as the expected steps for a walker starting from vertex i to vertex j selected randomly from the vertex set V, according to the stationary distribution $\boldsymbol{\pi}$. That is, $K = \sum_{j=1}^{n} \pi_j H_{ij}$. The Kemeny constant has been used to measure the user navigation efficiency through the WWW [22] and robotic surveillance efficiency in network environments [32]. It can also measure the mixing rate of random walks [23].

Most quantities for random walks on graph $\mathcal{G}$ are determined by the eigenvalues and eigenvectors of the normalized Laplacian matrix [7], $\boldsymbol{S}^{-\frac{1}{2}}\boldsymbol{L}\boldsymbol{S}^{-\frac{1}{2}}$, of $\mathcal{G}$, including the walk centrality and Kemeny constant. By definition, $\boldsymbol{S}^{-\frac{1}{2}}\boldsymbol{L}\boldsymbol{S}^{-\frac{1}{2}}$ is a real, symmetric, semi-definitive matrix. Let $0 = \sigma_1 < \sigma_2 \le \sigma_3 \le \cdots \le \sigma_n$ be the n eigenvalues of the normalized Laplacian matrix $\boldsymbol{S}^{-\frac{1}{2}}\boldsymbol{L}\boldsymbol{S}^{-\frac{1}{2}}$. And let $\boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \boldsymbol{\psi}_3, \ldots, \boldsymbol{\psi}_n$ be their corresponding mutually orthogonal eigenvectors of unit length, where $\boldsymbol{\psi}_i = (\psi_{i1}, \psi_{i2}, \ldots, \psi_{in})^\top$. Then [5, 27],

$$H_j = \sum_{i=1}^{n} \pi_i H_{ij} = \frac{s}{s_j}\sum_{k=2}^{n}\frac{1}{\sigma_k}\psi_{kj}^2 \tag{4}$$

and

$$K = \sum_{j=1}^{n} \pi_j H_{ij} = \sum_{k=2}^{n}\frac{1}{\sigma_k}. \tag{5}$$

Equations (4) and (5) provide exact computation for the walk centrality and Kemeny constant, respectively. However, both formulas are expressed in terms of the eigenvalues and eigenvectors of the normalized Laplacian, the computation complexity for which scale as $O(n^3)$. Thus, direct computation for H_j and K using spectral method appears to be prohibitive for large networks, and is infeasible to those realistic networks with millions of vertices.

3 NEW FORMULAS FOR MARKOV CENTRALITY AND KEMENY CONSTANT

In this section, we first establish an explicit relation between the walk centrality H_j and the Kemeny constant K. Then we express both quantities in terms of quadratic forms of the pseudoinverse $\boldsymbol{L}^\dagger$ of graph Laplacian $\boldsymbol{L}$, based on which we put forward a random algorithm approximately computing H_j and K.

Although both the walk centrality H_j and the Kemeny constant K have attracted much attention from the scientific community, the relation between them for a generic graph $\mathcal{G}$ is still lacking. Below we show that the Kemeny constant K can be expressed in a linear combination of walk centrality H_j for all vertices in $\mathcal{G}$, as stated in the following lemma.

Lemma 3.1. *Let $\mathcal{G} = (V, E, w)$ be a connected weighted graph. Then, its Kemeny constant K and walk centrality H_j obey the following relation:*

$$K = \sum_{j=1}^{n} \pi_j H_j = \sum_{j=1}^{n} \frac{s_j}{s} H_j. \tag{6}$$

Proof. From (5), the Kemeny constant K is independent of the starting vertex i. Define $K_i = \sum_{j=1}^{n} \pi_j H_{ij}$. Then $K_i = K_j$ holds for any pair of vertices i and j. Thus, we have

$$\begin{aligned} K = K_i &= \sum_{i=1}^{n} \pi_i \left(\sum_{j=1}^{n} \pi_j H_{ij}\right) \\ &= \sum_{j=1}^{n} \pi_j \left(\sum_{i=1}^{n} \pi_i H_{ij}\right) \\ &= \sum_{j=1}^{n} \frac{s_j}{s} H_j, \end{aligned}$$

which establishes the lemma. □

After obtaining the relation governing the Kemeny constant K and walk centrality H_j, we continue to express them in terms of quadratic forms of matrix $\boldsymbol{L}^\dagger$.

Lemma 3.2. *Let $\mathcal{G} = (V, E, w)$ be a connected weighted graph with Laplacian matrix $\mathbf{L}$. Then, the walk centrality H_j and Kemeny constant K can be represented in terms of quadratic forms of the pseudoinverse $\boldsymbol{L}^\dagger$ of matrix $\boldsymbol{L}$ as:*

$$H_j = s(\boldsymbol{e}_j - \boldsymbol{\pi})^\top \boldsymbol{L}^\dagger (\boldsymbol{e}_j - \boldsymbol{\pi}) \tag{7}$$

and

$$K = \sum_{j=1}^{n} s_j (\boldsymbol{e}_j - \boldsymbol{\pi})^\top \boldsymbol{L}^\dagger (\boldsymbol{e}_j - \boldsymbol{\pi}). \tag{8}$$

Proof. We first prove (7). Inserting (3) and (1) into (4) leads to

$$\begin{aligned} H_j &= \frac{1}{2}\sum_{i=1}^{n}\pi_i \sum_{z=1}^{n} s_z (R_{ij} + R_{jz} - R_{iz}) \\ &= \frac{1}{s}\sum_{i=1}^{n} s_i \sum_{z=1}^{n} s_z \left(\boldsymbol{L}_{jj}^\dagger - \boldsymbol{L}_{ij}^\dagger - \boldsymbol{L}_{jz}^\dagger + \boldsymbol{L}_{iz}^\dagger\right) \end{aligned} \tag{9}$$

The four terms in the brackets of (9) can be sequentially calculated as follows:

$$\sum_{i=1}^{n} s_i \sum_{z=1}^{n} s_z \boldsymbol{L}_{jj}^\dagger = s^2 \boldsymbol{e}_j^\top \boldsymbol{L}^\dagger \boldsymbol{e}_j, \tag{10}$$

$$\sum_{i=1}^{n} s_i \sum_{z=1}^{n} s_z \boldsymbol{L}_{ij}^\dagger = \sum_{i=1}^{n} s_i \sum_{z=1}^{n} s_z \boldsymbol{L}_{jz}^\dagger = s^2 \boldsymbol{e}_j^\top \boldsymbol{L}^\dagger \boldsymbol{\pi}, \tag{11}$$

and

$$\sum_{i=1}^{n} s_i \sum_{z=1}^{n} s_z \boldsymbol{L}^\dagger_{iz} = s^2 \boldsymbol{\pi}^\top \boldsymbol{L}^\dagger \boldsymbol{\pi} \,. \tag{12}$$

Plugging (10), (11), and (12) into (9), we obtain

$$\begin{aligned} H_j &= s(\boldsymbol{e}_j^\top \boldsymbol{L}^\dagger \boldsymbol{e}_j - 2\boldsymbol{e}_j^\top \boldsymbol{L}^\dagger \boldsymbol{\pi} + \boldsymbol{\pi}^\top \boldsymbol{L}^\dagger \boldsymbol{\pi}) \\ &= s(\boldsymbol{e}_j - \boldsymbol{\pi})^\top \boldsymbol{L}^\dagger (\boldsymbol{e}_j - \boldsymbol{\pi}). \end{aligned} \tag{13}$$

Substituting (13) into (6) gives (8). □

4 NEARLY LINEAR TIME APPROXIMATION ALGORITHM

In the preceding section, we reduce the problem of computing H_j and K to evaluating the quadratic forms $(\boldsymbol{e}_u - \boldsymbol{\pi})^\top \boldsymbol{L}^\dagger (\boldsymbol{e}_u - \boldsymbol{\pi})$, $u = 1, 2, \dots, n$, of matrix $\boldsymbol{L}^\dagger$. However, this involves computing the pseudoinverse of $\boldsymbol{L}$, the straightforward computation for which still has a complexity of $O(n^3)$, making it infeasible to huge networks. Here, we present an algorithm to compute an approximation of H_u for all $u \in V$ and K in nearly linear time with respect to the number of edges, which has a strict theoretical guarantee with high probability.

Let $C(u) = (\boldsymbol{e}_u - \boldsymbol{\pi})^\top \boldsymbol{L}^\dagger (\boldsymbol{e}_u - \boldsymbol{\pi})$, which can be written in an Euclidian norm as

$$\begin{aligned} C(u) &= (\boldsymbol{e}_u - \boldsymbol{\pi})^\top \boldsymbol{L}^\dagger \boldsymbol{L} \boldsymbol{L}^\dagger (\boldsymbol{e}_u - \boldsymbol{\pi}) \\ &= (\boldsymbol{e}_u - \boldsymbol{\pi})^\top \boldsymbol{L}^\dagger \boldsymbol{B}^\top \boldsymbol{W} \boldsymbol{B} \boldsymbol{L}^\dagger (\boldsymbol{e}_u - \boldsymbol{\pi}) \\ &= (\boldsymbol{e}_u - \boldsymbol{\pi})^\top \boldsymbol{L}^\dagger \boldsymbol{B}^\top \boldsymbol{W}^{\frac{1}{2}} \boldsymbol{W}^{\frac{1}{2}} \boldsymbol{B} \boldsymbol{L}^\dagger (\boldsymbol{e}_u - \boldsymbol{\pi}) \\ &= \|\boldsymbol{W}^{\frac{1}{2}} \boldsymbol{B} \boldsymbol{L}^\dagger (\boldsymbol{e}_u - \boldsymbol{\pi})\|^2. \end{aligned} \tag{14}$$

This in fact equals the square of the distance between vectors $\boldsymbol{W}^{\frac{1}{2}} \boldsymbol{B} \boldsymbol{L}^\dagger \boldsymbol{e}_u$ and $\boldsymbol{W}^{\frac{1}{2}} \boldsymbol{B} \boldsymbol{L}^\dagger \boldsymbol{\pi}$, which can be evaluated by the Johnson-Lindenstraus Lemma [1].

Lemma 4.1. *Given fixed vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n \in \mathbb{R}^d$ and $\epsilon > 0$, let $\boldsymbol{Q}_{k \times d}$, $k \geq 24 \log n / \epsilon^2$, be a matrix with each entry equal to $1/\sqrt{k}$ or $-1/\sqrt{k}$ with the same probability $1/2$. Then with probability at least $1 - 1/n$,*

$$(1 - \epsilon)\|\boldsymbol{v}_i - \boldsymbol{v}_j\|^2 \leq \|\boldsymbol{Q}\boldsymbol{v}_i - \boldsymbol{Q}\boldsymbol{v}_j\|^2 \leq (1 + \epsilon)\|\boldsymbol{v}_i - \boldsymbol{v}_j\|^2$$

for all pairs $i, j \leq n$.

Lemma 4.1 indicates that, the pairwise distances $\|\boldsymbol{v}_i - \boldsymbol{v}_j\|^2$ $(i, j = 1, 2, \dots, n)$ are almost preserved if we project the n vectors $\boldsymbol{v}_i$ $(i = 1, 2, \dots, n)$ into a lower-dimensional space, spanned by $O(\log n)$ random vectors.

In order to compute $C(u)$, we use Lemma 4.1 to reduce the dimension. Let $\boldsymbol{Q}$ be a $k \times m$ random projection matrix. Then $\|\boldsymbol{Q}\boldsymbol{W}^{\frac{1}{2}} \boldsymbol{B} \boldsymbol{L}^\dagger (\boldsymbol{e}_u - \boldsymbol{\pi})\|$ is a good approximation for $\|\boldsymbol{W}^{\frac{1}{2}} \boldsymbol{B} \boldsymbol{L}^\dagger (\boldsymbol{e}_u - \boldsymbol{\pi})\|$. Here we can use sparse matrix multiplication to compute $\boldsymbol{Q}\boldsymbol{W}^{\frac{1}{2}} \boldsymbol{B}$. However, computing $\boldsymbol{Z} = \boldsymbol{Q}\boldsymbol{W}^{\frac{1}{2}} \boldsymbol{B} \boldsymbol{L}^\dagger$ directly involves inverting $\boldsymbol{L} + \frac{1}{n}\boldsymbol{J}$. We avoid this by solving the system of equations $\boldsymbol{L}\boldsymbol{z}_i = \boldsymbol{q}_i$, $i = 1, \dots, k$, where $\boldsymbol{z}_i^\top$ and $\boldsymbol{q}_i^\top$ are, respectively, the i-th row of $\boldsymbol{Z}$ and $\boldsymbol{Q}\boldsymbol{W}^{\frac{1}{2}} \boldsymbol{B}$. For the convenience of description, in the sequel we use the notation $\widetilde{O}(\cdot)$ to hide poly log factors. By using Laplacian solvers [8, 18, 20, 26, 34, 35], $\boldsymbol{z}_i^\top$ can be efficiently approximated. We here use the solver from [8], the performance of which is characterized in the following lemma.

Lemma 4.2. *There is an algorithm $\boldsymbol{x} = \texttt{LaplSolve}(\boldsymbol{L}, \boldsymbol{y}, \delta)$ which takes a Laplacian matrix $\boldsymbol{L}$, a column vector $\boldsymbol{y}$, and an error parameter $\delta > 0$, and returns a column vector $\boldsymbol{x}$ satisfying $\boldsymbol{1}^\top \boldsymbol{x} = 0$ and*

$$\|\boldsymbol{x} - \boldsymbol{L}^\dagger \boldsymbol{y}\|_{\boldsymbol{L}} \leq \delta \|\boldsymbol{L}^\dagger \boldsymbol{y}\|_{\boldsymbol{L}},$$

where $\|\boldsymbol{y}\|_{\boldsymbol{L}} = \sqrt{\boldsymbol{y}^\top \boldsymbol{L} \boldsymbol{y}}$. The algorithm runs in expected time $\widetilde{O}(m \log(1/\delta))$.

By Lemmas 4.1 and 4.2, one can approximate $C(u)$ arbitrarily well.

Lemma 4.3. *Given an approximate factor $\epsilon \leq 1/2$ and a $k \times n$ matrix $\boldsymbol{Z}$ that satisfies*

$$(1 - \epsilon) C(u) \leq \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2 \leq (1 + \epsilon) C(u),$$

for any vertex $u \in V$ and

$$\begin{aligned} &(1 - \epsilon)\|\boldsymbol{W}^{\frac{1}{2}} \boldsymbol{B} \boldsymbol{L}^\dagger (\boldsymbol{e}_u - \boldsymbol{e}_v)\|^2 \\ \leq &\|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{e}_v)\|^2 \\ \leq &(1 + \epsilon)\|\boldsymbol{W}^{\frac{1}{2}} \boldsymbol{B} \boldsymbol{L}^\dagger (\boldsymbol{e}_u - \boldsymbol{e}_v)\|^2 \end{aligned}$$

for any pair of vertices $u, v \in V$. Let $\boldsymbol{z}_i$ be the i-th row of $\boldsymbol{Z}$ and let $\tilde{\boldsymbol{z}}_i$ be an approximation of $\boldsymbol{z}_i$ for all $i \in \{1, 2, \dots, k\}$, satisfying

$$\|\boldsymbol{z}_i - \tilde{\boldsymbol{z}}_i\|_{\boldsymbol{L}} \leq \delta \|\boldsymbol{z}_i\|_{\boldsymbol{L}}, \tag{15}$$

where

$$\delta \leq \frac{\epsilon}{3} \frac{s - s_u}{s} \sqrt{\frac{(1 - \epsilon) w_{\min}}{(1 + \epsilon) n^4 w_{\max}}}. \tag{16}$$

Then for any vertex u belonging to V,

$$(1 - \epsilon)^2 C(u) \leq \|\tilde{\boldsymbol{Z}}(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2 \leq (1 + \epsilon)^2 C(u), \tag{17}$$

where $\tilde{\boldsymbol{Z}} = [\tilde{\boldsymbol{z}}_1, \tilde{\boldsymbol{z}}_2, \dots, \tilde{\boldsymbol{z}}_k]^\top$.

Proof. To prove (17), it is sufficient to show that for an arbitrary vertex u,

$$\begin{aligned} &\left| \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2 - \|\tilde{\boldsymbol{Z}}(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2 \right| \\ = &\left| \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\| - \|\tilde{\boldsymbol{Z}}(\boldsymbol{e}_u - \boldsymbol{\pi})\| \right| \times \\ &\left| \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\| + \|\tilde{\boldsymbol{Z}}(\boldsymbol{e}_u - \boldsymbol{\pi})\| \right| \\ \leq &\left(\frac{2\epsilon}{3} + \frac{\epsilon^2}{9} \right) \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2, \end{aligned} \tag{18}$$

which is obeyed if

$$\left| \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\| - \|\tilde{\boldsymbol{Z}}(\boldsymbol{e}_u - \boldsymbol{\pi})\| \right| \leq \frac{\epsilon}{3} \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\| \,. \tag{19}$$

This can be understood from the following arguments. On the one hand, if $\left| \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2 - \|\tilde{\boldsymbol{Z}}(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2 \right| \leq \left(\frac{2\epsilon}{3} + \frac{\epsilon^2}{9} \right) \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2$, then

$$\begin{aligned} &\left(1 - \frac{2\epsilon}{3} - \frac{\epsilon^2}{9} \right) \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2 \\ \leq &\|\tilde{\boldsymbol{Z}}(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2 \\ \leq &\left(1 + \frac{2\epsilon}{3} + \frac{\epsilon^2}{9} \right) \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2, \end{aligned}$$

which, combining with $\epsilon \leq 1/2$ and the assumption that $(1 - \epsilon) C(u) \leq \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2 \leq (1 + \epsilon) C(u)$, leads to Eq. (17). On the other hand, if Eq. (19) is true, we have $\|\tilde{\boldsymbol{Z}}(\boldsymbol{e}_u - \boldsymbol{\pi})\| \leq (1 + \frac{\epsilon}{3}) \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\|$. Thus,

$$\left| \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\| + \|\tilde{\boldsymbol{Z}}(\boldsymbol{e}_u - \boldsymbol{\pi})\| \right| \leq (2 + \frac{\epsilon}{3}) \|\boldsymbol{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\| \,,$$

which results in Eq. (18).

We now prove that (19) holds true. By applying triangle inequality and Cauchy-Schwarz inequality, we obtain

$$\begin{aligned}
&\left|\|Z(\boldsymbol{e}_u - \boldsymbol{\pi})\| - \|\tilde{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\|\right| \\
\le\ & \left\|(Z - \tilde{Z})(\boldsymbol{e}_u - \boldsymbol{\pi})\right\| \\
=\ & \frac{1}{s}\left\|\sum_{v=1}^{n} s_v (Z - \tilde{Z})(\boldsymbol{e}_u - \boldsymbol{e}_v)\right\| \\
\le\ & \frac{1}{s}\sum_{v=1}^{n} s_v \left\|(Z - \tilde{Z})(\boldsymbol{e}_u - \boldsymbol{e}_v)\right\| \\
\le\ & \frac{1}{s}\sqrt{\sum_{v=1}^{n} s_v^2 \sum_{v=1}^{n}\left\|(Z - \tilde{Z})(\boldsymbol{e}_u - \boldsymbol{e}_v)\right\|^2} \\
\le\ & \sqrt{\sum_{v=1}^{n}\left\|(Z - \tilde{Z})(\boldsymbol{e}_u - \boldsymbol{e}_v)\right\|^2},
\end{aligned}$$

where the last inequality is due to the fact that $s = \sum_{v=1}^{n} s_v \ge \sqrt{\sum_{v=1}^{n} s_v^2}$.

Since we only consider connected graphs, there exists a simple path $\mathcal{P}_v$ between u and any other vertex v. Applying the triangle inequality along path $\mathcal{P}_v$, the square of the last sum term in the above equation can be evaluated as:

$$\begin{aligned}
&\sum_{v=1}^{n}\left\|(Z - \tilde{Z})(\boldsymbol{e}_u - \boldsymbol{e}_v)\right\|^2 \\
\le\ & \sum_{v=1}^{n}\left(\sum_{a\sim b\in\mathcal{P}_v}\left\|(Z - \tilde{Z})(\boldsymbol{e}_a - \boldsymbol{e}_b)\right\|\right)^2 \\
\le\ & n\sum_{v=1}^{n}\sum_{a\sim b\in\mathcal{P}_v}\left\|(Z - \tilde{Z})(\boldsymbol{e}_a - \boldsymbol{e}_b)\right\|^2 \\
\le\ & n^2\sum_{a\sim b\in E}\left\|(Z - \tilde{Z})(\boldsymbol{e}_a - \boldsymbol{e}_b)\right\|^2 \\
=\ & n^2\left\|(Z - \tilde{Z})\boldsymbol{B}^\top\right\|_F^2 \\
=\ & n^2\left\|\boldsymbol{B}(Z - \tilde{Z})^\top\right\|_F^2 \\
\le\ & \frac{n^2}{w_{\min}}\left\|\boldsymbol{W}^{1/2}\boldsymbol{B}(Z - \tilde{Z})^\top\right\|_F^2,
\end{aligned}$$

where $\|\boldsymbol{M}\|_F$ denotes the Frobenius norm of matrix $\boldsymbol{M}$, defined as the square root of the trace for matrix $\boldsymbol{M}^\top\boldsymbol{M}$. The last term can be bounded as

$$\begin{aligned}
&\frac{n^2}{w_{\min}}\left\|\boldsymbol{W}^{1/2}\boldsymbol{B}(Z - \tilde{Z})^\top\right\|_F^2 \\
=&\frac{n^2}{w_{\min}}\mathrm{Tr}\left((Z - \tilde{Z})\boldsymbol{B}^\top\boldsymbol{W}\boldsymbol{B}(Z - \tilde{Z})^\top\right) \\
=&\frac{n^2}{w_{\min}}\mathrm{Tr}\left((Z - \tilde{Z})\boldsymbol{L}(Z - \tilde{Z})^\top\right) \\
=&\frac{n^2}{w_{\min}}\sum_{i=1}^{k}(z_i - \tilde{z}_i)^\top\boldsymbol{L}(z_i - \tilde{z}_i) \\
\le&\frac{n^2\delta^2}{w_{\min}}\sum_{i=1}^{k} z_i^\top\boldsymbol{L}z_i \\
=&\frac{n^2\delta^2}{w_{\min}}\mathrm{Tr}\left(Z\boldsymbol{L}Z^\top\right) \\
=&\frac{n^2\delta^2}{w_{\min}}\left\|\boldsymbol{W}^{1/2}\boldsymbol{B}Z^\top\right\|_F^2,
\end{aligned}$$

where the inequality follows from (15) and the last term can be further evaluated by Lemma 2.1 as

$$\begin{aligned}
&\frac{n^2\delta^2}{w_{\min}}\left\|\boldsymbol{W}^{1/2}\boldsymbol{B}Z^\top\right\|_F^2 \\
=&\frac{n^2\delta^2}{w_{\min}}\sum_{a\sim b\in E} w_{a\sim b}\left\|Z(\boldsymbol{e}_a - \boldsymbol{e}_b)\right\|^2 \\
\le&\frac{\delta^2 n^2(1+\epsilon)}{w_{\min}}\sum_{a\sim b\in E} w_{a\sim b}\left\|\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{B}\boldsymbol{L}^\dagger(\boldsymbol{e}_a - \boldsymbol{e}_b)\right\|^2 \\
=&\frac{\delta^2 n^2(1+\epsilon)}{w_{\min}}\sum_{a\sim b\in E} w_{a\sim b}(\boldsymbol{e}_a - \boldsymbol{e}_b)^\top\boldsymbol{L}^\dagger\boldsymbol{B}^\top\boldsymbol{W}\boldsymbol{B}\boldsymbol{L}^\dagger(\boldsymbol{e}_a - \boldsymbol{e}_b) \\
=&\frac{\delta^2 n^2(1+\epsilon)}{w_{\min}}\sum_{a\sim b\in E} w_{a\sim b}(\boldsymbol{e}_a - \boldsymbol{e}_b)^\top\boldsymbol{L}^\dagger\boldsymbol{L}\boldsymbol{L}^\dagger(\boldsymbol{e}_a - \boldsymbol{e}_b) \\
=&\frac{\delta^2 n^2(1+\epsilon)}{w_{\min}}\sum_{a\sim b\in E} w_{a\sim b}(\boldsymbol{e}_a - \boldsymbol{e}_b)^\top\boldsymbol{L}^\dagger(\boldsymbol{e}_a - \boldsymbol{e}_b) \\
=&\frac{\delta^2 n^2(1+\epsilon)}{w_{\min}}\sum_{a\sim b\in E} w_{a\sim b}R_{ab} \\
=&\frac{\delta^2 n^2(n-1)(1+\epsilon)}{w_{\min}}.
\end{aligned}$$

In addition, $\|Z(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2$ can also be bounded by

$$\begin{aligned}
&\|Z(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2 \ge (1-\epsilon)C(u) \\
&= (1-\epsilon)(\boldsymbol{e}_u - \boldsymbol{\pi})^\top\boldsymbol{L}^\dagger(\boldsymbol{e}_u - \boldsymbol{\pi}) \\
&\ge (1-\epsilon)\lambda_n^{-1}\|\boldsymbol{e}_u - \boldsymbol{\pi}\|^2 \\
&\ge (1-\epsilon)(n w_{\max})^{-1}\frac{(s - s_u)^2}{s^2}.
\end{aligned}$$

In the above equation, the first inequation follows due to the following reason. Note that $\boldsymbol{e}_u - \boldsymbol{\pi}$ is orthogonal to vector $\boldsymbol{1}$, which is an eigenvector of $\boldsymbol{L}^\dagger$ corresponding to the 0 eigenvalue. Hence, $(\boldsymbol{e}_u - \boldsymbol{\pi})^\top\boldsymbol{L}^\dagger(\boldsymbol{e}_u - \boldsymbol{\pi}) \ge \lambda_n^{-1}\|\boldsymbol{e}_u - \boldsymbol{\pi}\|^2$ always holds true.

Thus, we have

$$\begin{aligned}
&\frac{\left|\|Z(\boldsymbol{e}_u - \boldsymbol{\pi})\| - \|\tilde{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\|\right|}{\|Z(\boldsymbol{e}_u - \boldsymbol{\pi})\|} \\
\le\ & \delta\left(\frac{n^2(n-1)(1+\epsilon)}{w_{\min}}\right)^{1/2}\left(\frac{n w_{\max}}{1-\epsilon}\right)^{1/2}\frac{s}{s - s_u} \\
\le\ & \frac{\epsilon}{3},
\end{aligned}$$

where δ is given by (16). □

Lemma 4.3 leads to the following theorem.

Theorem 4.4. *There is a $\widetilde{O}(m\log c/\epsilon^2)$ time algorithm, which inputs a scalar $0 < \epsilon < 1$ and a graph $\mathcal{G} = (V, E, w)$ where $c = \frac{w_{\max}}{w_{\min}}$, and returns a $(24\log n/\epsilon^2)\times n$ matrix $\tilde{Z}$ such that with probability at least $1 - 1/n$,*

$$(1-\epsilon)^2 C(u) \le \|\tilde{Z}(\boldsymbol{e}_u - \boldsymbol{\pi})\|^2 \le (1+\epsilon)^2 C(u)$$

for any vertex $u \in V$.

Based on Theorem 4.4, we present an algorithm Approx$\mathcal{HK}$ to approximately compute the walk centrality H_u for all vertices $u \in V$ and the Kemeny constant K, the pseudocode of which is provided in Algorithm 1.

```
Algorithm 1: Approx$\mathcal{HK}(\mathcal{G}, \epsilon)$
  Input  : $\mathcal{G}$: a connected undirected graph.
           $\epsilon$: an approximation parameter
  Output : $\tilde{H} = \{u, \tilde{H}_u | u \in V\}$: $\tilde{H}_u$ is an approximation of the
           walk centrality $H_u$ of vertex $u$; $\tilde{K}$: approximation
           of the Kemeny constant $K$
1  $\boldsymbol{L}$ = Laplacian of $\mathcal{G}$, $s_u$ = the strength of $u$ for all $u \in V$,
     $s = \sum_{u \in V} s_u$
2  Construct a matrix $\boldsymbol{Q}_{k \times m}$, where $k = \lceil 24 \log n / \epsilon^2 \rceil$ and
     each entry is $\pm 1/\sqrt{k}$ with identical probability
3  for $i = 1$ to $k$ do
4      $\boldsymbol{q}_i^\top$=the $i$-th row of $\boldsymbol{Q}_{k \times m} \boldsymbol{W}^{1/2} \boldsymbol{B}$
5      $\tilde{z}_i$ = LaplSolve($\boldsymbol{L}, \boldsymbol{q}_i, \delta$) where parameter $\delta$ is given
         by (16)
6  Calculate the constant vector $\mathbf{p} = \tilde{\boldsymbol{Z}} \boldsymbol{\pi}$
7  for each $u \in V$ do
8      $\tilde{H}_u = s \| \tilde{\boldsymbol{Z}}_{:,u} - \mathbf{p} \|^2$
9  $\tilde{K} = \sum_{u \in V} \frac{s_u}{s} \tilde{H}_u$
10 return $\tilde{H} = \{u, \tilde{H}_u | u \in V\}$ and $\tilde{K}$
```

5 NUMERICAL EXPERIMENTS

In this section, we present experimental results for various real and model networks to demonstrate the efficiency and accuracy of our approximation algorithm Approx$\mathcal{HK}$ for computing the walk centrality H_u of every vertex and the Kemeny constant K.

All experiments are conducted on a machine with 4-core 4.2GHz Intel i7-7700K CPU and with 32GB of RAM. We implement the approximation algorithm Approx$\mathcal{HK}$ in *Julia v0.6.0*, where the `LaplSolve` is from [20].

5.1 Results for Vertex Centrality of Real Networks

We first test the algorithm for approximating centrality H_u on a large set of realistic networks from different domains. The data of these networks are taken from the Koblenz Network Collection [19]. For those networks that are disconnected originally, we perform our experiments on their largest connected components (LCC). Related information about the number of vertices and edges for the studied real networks and their LCC is shown in Table 1, where networks are listed in an increasing order of the number of vertices in the original networks. The smallest network has 198 vertices, while the largest one consists of more than 1.3×10^6 vertices.

We now investigate the efficiency of our approximation algorithm Approx$\mathcal{HK}$. To this end, in Table 2, we report the running time of Approx$\mathcal{HK}$ and that of the direct accurate algorithm called Exact$\mathcal{HK}$ that calculates the centrality H_u for each $u \in V$ by calculating the pseudoinverse of $\boldsymbol{L}$ as given in Lemma 3.2. To objectively evaluate the running time, for both Exact$\mathcal{HK}$ and Approx$\mathcal{HK}$ on all considered networks except the last seven ones marked with $*$, we enforce the program to run on a single thread. From Table 2 we can see that for moderate approximation parameter ϵ, the computational time for Approx$\mathcal{HK}$ is significantly smaller than that for Exact$\mathcal{HK}$, especially for large-scale networks tested. Thus, Approx$\mathcal{HK}$ can significantly improve the performance compared with Exact$\mathcal{HK}$. For the seven networks marked with $*$, the number of vertices for which ranges from 10^5 to 10^6, we cannot run the Exact$\mathcal{HK}$ algorithm on the machine due to the limits of memory and time. However, for these networks, we can approximately compute their walk centrality for all vertices by using algorithm Approx$\mathcal{HK}$, which further show that Approx$\mathcal{HK}$ is efficient and scalable to large networks.

In addition to the high efficiency, our algorithm Approx$\mathcal{HK}$ also provides a desirable approximation $\hat{H}_u$ for the walk centrality H_u. To show the accuracy of Approx$\mathcal{HK}$, we compare the approximate results for Approx$\mathcal{HK}$ with the exact results computed by Lemma 3.2. In Table 3, we report the mean relative error σ of algorithm Approx$\mathcal{HK}$, with σ being defined by $\sigma = \frac{1}{n} \sum_{u \in V} |H_u - \tilde{H}_u| / H_u$. From Table 3 we can see that the actual mean relative errors for all ϵ and all networks are very small, and are almost negligible for smaller ϵ. More interestingly, for all networks tested, σ are magnitudes smaller than the theoretical guarantee. Therefore, the approximation algorithm Approx$\mathcal{HK}$ provides very accurate results in practice.

5.2 Results for Kemeny Constant of Model Networks

To further demonstrate the performance of our proposed algorithm Approx$\mathcal{HK}$, we use it to compute the Kemeny constant for some model networks. Although for a general graph, the exact result for its Kemeny constant is difficult to obtain, for some model networks generated by an iterative way, their Kemeny constant can be determined explicitly. For example, for the pseudofractal scale-free web [41] and the Koch network [40], one can obtain exact expressions for their Kemeny constant.

Table 1: Statistics of the collection of datasets used in our experiments. For a network with n vertices and m edges, we denote the number of vertices and edges in its largest connected component by n' and m', respectively.

Network	n	m	n'	m'
Jazz musicians	198	2,742	198	2,742
Chicago	1,467	1,298	823	822
Hamster full	2,426	16,631	2,000	16,098
Facebook (NIPS)	4,039	88,234	4,039	88,234
CA-GrQc	5,242	14,496	4,158	13,422
Reactome	6,327	147,547	5,973	145,778
Route views	6,474	13,895	6,474	12,572
Pretty Good Privacy	10,680	24,316	10,680	24,316
CA-HepPh	12,008	118,521	11,204	117,619
Astro-ph	18,772	198,110	17,903	196,972
CAIDA	26,475	53,381	26,475	53,381
Brightkite	58,228	214,078	56,739	212,945
Livemocha*	104,103	2,193,083	104,103	2,193,083
WordNet*	146,005	656,999	145,145	656,230
Gowalla*	196,591	950,327	196,591	950,327
com-DBLP*	317,080	1,049,866	317,080	1,049,866
Amazon*	334,863	925,872	334,863	925,872
Pennsylvania*	1,088,092	1,541,898	1,087,562	1,541,514
roadNet-TX*	1,379,917	1,921,660	1,351,137	1,879,201

Table 2: The running time (seconds, s) of Exact$\mathcal{HK}$ and Approx$\mathcal{HK}$ with various ϵ on several realistic networks.

Network	Exact$\mathcal{HK}$ (s)	Approx$\mathcal{HK}$ (s) with various ϵ					
		0.3	0.25	0.2	0.15	0.1	0.05
Jazz musicians	0.001	0.020	0.026	0.043	0.071	0.144	0.613
Chicago	0.033	0.007	0.009	0.013	0.023	0.050	0.204
Hamster full	0.385	0.226	0.265	0.405	0.703	1.591	6.317
Facebook (NIPS)	2.889	0.965	1.338	1.972	3.408	7.501	30.92
CA-GrQc	3.127	0.306	0.383	0.590	1.039	2.336	9.345
Reactome	8.824	1.753	2.470	3.709	6.008	13.88	56.09
Route views	11.35	0.286	0.351	0.535	0.980	1.993	7.907
Pretty Good Privacy	47.79	0.746	0.907	1.450	2.494	5.928	21.17
CA-HepPh	54.39	1.937	2.496	3.792	7.018	15.35	57.16
Astro-ph	219.2	4.959	4.789	7.223	13.32	29.61	112.9
CAIDA	706.3	1.724	1.777	2.781	4.731	10.51	42.90
Brightkite	4589	8.595	9.090	13.81	24.50	49.39	220.7
Livemocha*	–	57.33	79.59	125.7	210.0	464.3	1957
WordNet*	–	23.03	31.51	48.15	96.36	188.4	1112
Gowalla*	–	36.76	52.33	73.40	130.5	312.0	1189
com-DBLP*	–	57.13	90.41	143.4	248.8	511.2	2250
Amazon*	–	86.32	116.6	170.9	303.0	709.0	2523
Pennsylvania*	–	290.4	433.1	647.1	1103	2701	13233
roadNet-TX*	–	372.4	601.7	926.1	1555	3301	17522

Table 3: Mean relative error σ of Approx$\mathcal{HK}$ ($\times 10^{-2}$).

Network	Mean relative error for various ϵ			
	0.3	0.2	0.1	0.05
Jazz musicians	10.2	7.23	3.89	1.79
Chicago	9.01	5.98	3.33	1.48
Hamster full	8.88	6.07	2.94	1.48
Facebook (NIPS)	9.25	5.78	2.86	1.51
CA-GrQc	7.94	5.25	2.64	1.33
Reactome	8.39	5.61	2.81	1.43
Route views	5.74	3.90	1.92	0.97
Pretty Good Privacy	7.03	4.75	2.34	1.19
CA-HepPh	7.67	5.20	2.57	1.28
Astro-ph	7.85	5.24	2.63	1.32
CAIDA	5.32	3.54	1.75	0.88
Brightkite	5.90	3.95	1.99	0.98

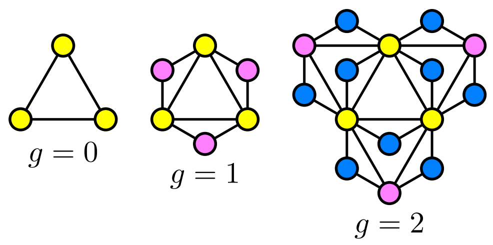

Figure 1: Illustration of the first several iterations of the pseudofractal scale-free web.

In the following we use algorithm Approx$\mathcal{HK}$ to approximately compute the Kemeny constant for the pseudofractal scale-free web and the Koch network. One main justification for selecting these two networks is that they display the remarkable scale-free small-world properties as observed in most real networks [30]. Both the pseudofractal scale-free web and the Koch network are constructed in an iterative way. Their particular constructions allow to explicitly determine their Kemeny constant.

Let $\mathcal{F}_g$ ($g \geq 0$) denote the pseudofractal scale-free web after g iterations. For $g = 0$, $\mathcal{F}_0$ consists of a triangle of three vertices and three edges. For $g > 0$, $\mathcal{F}_g$ is obtained from $\mathcal{F}_{g-1}$ as follows. For each existing edge in $\mathcal{F}_{g-1}$, a new vertex is created and linked to both end vertices of the edge. Figure 1 schematically illustrates the construction process of the pseudofractal scale-free web. In network $\mathcal{F}_g$, there are $(3^{g+1} + 3)/2$ vertices and 3^{g+1} edges. It has been shown [41] that the Kemeny constant $K(\mathcal{F}_g)$ for $\mathcal{F}_g$ is

$$K(\mathcal{F}_g) = \frac{5}{2} \times 3^g - \frac{5}{3} \times 2^g + \frac{1}{2}\,. \tag{20}$$

The Koch network is also built in an iterative way. Let $\mathcal{M}_g$ ($g \geq 0$) denote the Koch network after g iterations. Initially ($g = 0$), $\mathcal{M}_0$ is a triangle with three vertices and three edges. For $g \geq 1$, $\mathcal{M}_g$ is obtained from $\mathcal{M}_{g-1}$ by performing the following operations. For each of the three vertices in every existing triangle in $\mathcal{M}_{g-1}$, two new vertices are created, both of which and their "mother" vertices are connected to one another forming a new triangle. Figure 2 illustrates the growth process of the Koch network. In network $\mathcal{M}_g$, the number of vertices is $2 \times 4^g + 1$, and the number of edges is 3×4^g. In [40], the Kemeny constant $K(\mathcal{M}_g)$ for $\mathcal{M}_g$ was obtained to be

$$K(\mathcal{M}_g) = (1 + 2g) \times 4^g + \frac{1}{3}\,. \tag{21}$$

Table 4: Exact Kemeny constant K, their approximation $\tilde{K}$, relative error $\rho = (K - \tilde{K})/K$, and running time (seconds, s) for $\tilde{K}$ on networks $\mathcal{F}_{12}$ and $\mathcal{M}_{10}$. K is obtained via (20) and (21), while $\tilde{K}$ is obtained through algorithm Approx$\mathcal{HK}$ with $\epsilon = 0.1$.

Network	Vertices	Edges	K	$\tilde{K}$	Error ρ	Time
$\mathcal{F}_{12}$	797,163	1,594,323	1,321,776	1,321,956	0.00014	207
$\mathcal{M}_{10}$	2,097,153	3,145,728	22,020,096	22,018,022	0.000094	1917

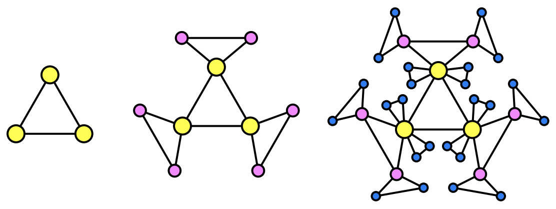

Figure 2: Construction process for the Koch network.

We use our algorithm Approx$\mathcal{HK}$ to compute the Kemeny constant on pseudofractal scale-free web $\mathcal{F}_{12}$ and the Koch network $\mathcal{M}_{10}$. The numerical results are reported in Table 4, which shows that the approximation algorithm Approx$\mathcal{HK}$ works effectively for both networks. This again demonstrates the advantage of our proposed algorithm for large networks.

6 CONCLUSIONS

The hitting time of random walks arises in many practical scenarios. However, the time cost of exactly computing hitting time is prohibitively expensive. In this paper, we studied a walk centrality and Kemeny constant of a graph, both of which are actually weighted average of hitting times and have found wide applications. We established a link between the two quantities, and reformulated them in terms of quadratic forms of the pseudoinverse of graph Laplacian. Moreover, we provided a randomized approximation algorithm with probabilistic guarantee, which computes the walk centrality for all vertices and Kemeny constant in nearly linear time with respect to the number of edges. Finally, we conducted extensive experiments on various real-world and model networks, which show that the proposed algorithm is both efficient and accurate, especially for large-scale networks.

ACKNOWLEDGMENTS

The work was supported by Shanghai Municipal Science and Technology Major Project (No. 2018SHZDZX01) and ZJLab, the National Key R & D Program of China (No. 2018YFB1305104), the National Natural Science Foundation of China under Grant 61872093. Zuobai Zhang was also supported by Fudan's Undergraduate Research Opportunities Program (FDUROP) under Grant No. 19914.

REFERENCES

[1] Dimitris Achlioptas. 2001. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM, 274–281.

[2] A Barrat, M Barthelemy, R Pastor-Satorras, and A Vespignani. 2004. The architecture of complex weighted networks. *Proceedings of the National Academy of Sciences of the United States of America* 101, 11 (MAR 16 2004), 3747–3752.

[3] Adi Ben-Israel and Thomas N. E Greville. 1974. *Generalized inverses: theory and applications*. J. Wiley. 406–413 pages.

[4] Andrew Beveridge. 2009. Centers for random walks on trees. *SIAM Journal on Discrete Mathematics* 23, 1 (2009), 300–318.

[5] Andrew Beveridge. 2016. A Hitting Time Formula for the Discrete Green's Function. *Combinatorics, Probability and Computing* 25, 3 (2016), 362–379.

[6] M. Chen, J. Z. Liu, and X. Tang. 2008. Clustering via Random Walk Hitting Time on Directed Graphs. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*. AAAI, 616–621.

[7] Fan RK Chung. 1997. *Spectral Graph Theory*. American Mathematical Society, Providence, RI.

[8] Michael B Cohen, Rasmus Kyng, Gary L Miller, Jakub W Pachocki, Richard Peng, Anup B Rao, and Shen Chen Xu. 2014. Solving SDD linear systems in nearly m log 1/2 n time. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*. ACM, 343–352.

[9] S Condamin, O Bénichou, V Tejedor, R Voituriez, and J Klafter. 2007. First-passage times in complex scale-invariant media. *Nature* 450, 7166 (2007), 77–80.

[10] Peter G Doyle and J Laurie Snell. 1984. *Random Walks and Electric Networks*. Mathematical Association of America.

[11] Francois Fouss, Alain Pirotte, J-M Renders, and Marco Saerens. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering* 19, 3 (2007), 355–369.

[12] Arpita Ghosh, Stephen Boyd, and Amin Saberi. 2008. Minimizing effective resistance of a graph. *SIAM Rev.* 50, 1 (2008), 37–66.

[13] Leo Grady. 2006. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, 11 (2006), 1768–1783.

[14] Jeffrey J Hunter. 2006. Mixing times with applications to perturbed Markov chains. *Linear Algebra Appl.* 417, 1 (2006), 108–123.

[15] Jeffrey J Hunter. 2014. The role of Kemeny's constant in properties of Markov chains. *Communications in Statistics — Theory and Methods* 43, 7 (2014), 1309–1321.

[16] Ali Jadbabaie and Alex Olshevsky. 2019. Scaling Laws for Consensus Protocols Subject to Noise. *IEEE Trans. Automat. Control* 64, 4 (2019), 1389–1402.

[17] Douglas J Klein and Milan Randić. 1993. Resistance distance. *Journal of Mathematical Chemistry* 12, 1 (1993), 81–95.

[18] Ioannis Koutis, Gary L Miller, and Richard Peng. 2011. A nearly-m log n time solver for SDD linear systems. In *Proceedings of 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. IEEE, 590–598.

[19] Jérôme Kunegis. 2013. KONECT: The Koblenz Network Collection. In *Proceedings of the 22nd International Conference on World Wide Web*. ACM, New York, USA, 1343–1350.

[20] Rasmus Kyng and Sushant Sachdeva. 2016. Approximate Gaussian elimination for Laplacians - fast, sparse, and simple. In *Proceedings of IEEE 57th Annual Symposium on Foundations of Computer Science*. IEEE, 573–582.

[21] Renaud Lambiotte, Jean-Charles Delvenne, and Mauricio Barahona. 2014. Random walks, Markov processes and the multiscale modular organization of complex networks. *IEEE Transactions on Network Science and Engineering* 1, 2 (2014), 76–90.

[22] Mark Levene and George Loizou. 2002. Kemeny's Constant and the Random Surfer. *The American Mathematical Monthly* 109, 8 (2002), 741–745.

[23] David Asher Levin, Yuval Peres, and Elizabeth Lee Wilmer. 2009. *Markov Chains and Mixing Times*. American Mathematical Society, Providence, RI.

[24] Huan Li and Aaron Schild. 2018. Spectral Subspace Sparsification. In *Proceedings of 2018 IEEE 59th Annual Symposium on Foundations of Computer Science*. IEEE, 385–396.

[25] Yuan Lin and Zhongzhi Zhang. 2013. Random walks in weighted networks with a perfect trap: An application of Laplacian spectra. *Physical Review E* 87, 6 (2013), 062140.

[26] Oren E Livne and Achi Brandt. 2012. Lean algebraic multigrid (LAMG): Fast graph Laplacian linear solver. *SIAM Journal on Scientific Computing* 34, 4 (2012), B499–B522.

[27] László Lovász. 1993. Random walks on graphs: A survey. *Combinatorics, Paul Erdös is eighty* 2, 1 (1993), 1–46.

[28] Charalampos Mavroforakis, Michael Mathioudakis, and Aristides Gionis. 2015. Absorbing random-walk centrality: Theory and algorithms. In *Proceedings of the 15th IEEE International Conference on Data Mining*. IEEE, 901–906.

[29] Qiaozhu Mei, Dengyong Zhou, and Kenneth Church. 2008. Query suggestion using hitting time. In *Proceedings of the 17th ACM conference on Information and Knowledge Management*. ACM, 469–478.

[30] M. E. J. Newman. 2003. The Structure and Function of Complex Networks. *SIAM Rev.* 45, 2 (2003), 167–256.

[31] Mark E. J. Newman. 2005. A measure of betweenness centrality based on random walks. *Social Networks* 27, 1 (2005), 39–54.

[32] Rushabh Patel, Pushkarini Agharkar, and Francesco Bullo. 2015. Robotic surveillance and Markov chains with minimal weighted Kemeny constant. *IEEE Transactions on Automatic Control* 60, 12 (2015), 3156–3167.

[33] Anand Dilip Sarwate and Alexandros G Dimakis. 2012. The impact of mobility on gossip algorithms. *IEEE Transactoins on Information Theory* 58, 3 (2012), 1731–1742.

[34] Daniel A Spielman. 2010. Algorithms, graph theory, and linear equations in Laplacian matrices. In *Proceedings of the International Congress of Mathematicians*. World Scientific, 2698–2722.

[35] Daniel A Spielman and Shang-Hua Teng. 2004. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth Annual ACM Symposium on Theory of Computing*. ACM, 81–90.

[36] V Tejedor, O Bénichou, and R Voituriez. 2009. Global mean first-passage times of random walks on complex networks. *Physical Review E* 80, 6 (2009), 065104.

[37] Prasad Tetali. 1991. Random walks and the effective resistance of networks. *Journal of Theoretical Probability* 4, 1 (1991), 101–109.

[38] Scott White and Padhraic Smyth. 2003. Algorithms for estimating relative importance in networks. In *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 266–275.

[39] Shunqi Wu, Zhongzhi Zhang, and Guanrong Chen. 2011. Random walks on dual Sierpiński gaskets. *The European Physical Journal B* 82, 1 (2011), 91–96.

[40] Pinchen Xie, Yuan Lin, and Zhongzhi Zhang. 2015. Spectrum of walk matrix for Koch network and its application. *Journal of Chemical Physics* 142, 22 (2015), 224106.

[41] Pinchen Xie, Zhongzhi Zhang, and Francesc Comellas. 2016. On the spectrum of the normalized Laplacian of iterated triangulations of graphs. *Appl. Math. Comput.* 273 (2016), 1123–1129.