Efficient Resistance Distance Computation: the Power of Landmark-based Approaches

Meihao Liao
Beijing Institute of Technology
Beijing, China
mhliao@bit.edu.cn

Rong-Hua Li
Beijing Institute of Technology
Beijing, China
lironghuabit@126.com

Qiangqiang Dai
Beijing Institute of Technology
Beijing, China
qiangd66@gmail.com

Hongyang Chen
Zhejiang Lab
Zhejiang, China
dr.h.chen@ieee.org

Hongchao Qin
Beijing Institute of Technology
Beijing, China
qhc.neu@gmail.com

Guoren Wang
Beijing Institute of Technology
Beijing, China
wanggrbit@126.com

ABSTRACT

Resistance distance is a fundamental metric to measure the similarity between two nodes in graphs which has been widely used in many real-world applications. In this paper, we study two problems on approximately computing resistance distance: (i) single-pair query which aims at calculating the resistance distance $r(s,t)$ for a given pair of nodes (s,t); and (ii) single-source query which is to compute all the resistance distances $r(s,u)$ for all nodes u in the graph with a given source node s. Existing algorithms for these two resistance distance query problems are often costly on large graphs. To efficiently solve these problems, we first establish several interesting connections among resistance distance, a new concept called v-absorbed random walk, random spanning forests, and a newly-developed v-absorbed push procedure. Based on such new connections, we propose three novel and efficient sampling-based algorithms as well as a deterministic algorithm for single-pair query; and we develop an online and two index-based approximation algorithms for single-source query. We show that the two index-based algorithms for single-source query take almost the same running time as the algorithms for single-pair query with the aid of a linear-size index. The striking feature of all our algorithms is that they are allowed to select an *easy-to-hit* node by random walks on the graph. Such an easy-to-hit landmark node v can make the v-absorbed random walk sampling, spanning tree sampling, as well as the v-absorbed push more efficient, thus significantly improving the performance of our algorithms. Extensive experiments on 10 real-life datasets show that our algorithms substantially outperform the state-of-the-art algorithms for two resistance distance query problems in terms of both running time and estimation errors.

ACM Reference Format:
Meihao Liao, Rong-Hua Li, Qiangqiang Dai, Hongyang Chen, Hongchao Qin, and Guoren Wang. 2022. Efficient Resistance Distance Computation: the Power of Landmark-based Approaches. In *Proceedings of SIGMOD '23: International Conference on Management of Data (SIGMOD '23).* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Proximity measures for nodes in networks play a crucial role in many network analysis tasks. Notable proximity measures include personalized PageRank [17, 24, 60], SimRank [23, 49], resistance distance [12, 53, 57], and Katz similarity [28, 42]. All of these proximity measures can be seen as measures based on random walks in graphs, which have been widely used in many real-world applications including web search [24, 57], recommendation system [26], and link predictions [31, 46].

In this paper, we focus on the resistance distance which is a fundamental graph metric for measuring the node similarity. Given a graph G, we can regard it as an electrical network, where each edge denotes a unit resistor and each node represents a junction connecting resistors. The resistance distance between two nodes s,t, denoted by $r(s,t)$, is the effective resistance between s and t in the electrical network G. It is well-known that the resistance distance has many interesting combinatorial explanations [8]. For example, $r(s,t)$ can be interpreted as the commute time of a random walk starting from s, visiting t, and then going back to s [44, 53]. It can also be explained as the normalized number of spanning 2-forests with s and t in two different connected components [8, 9], where a spanning 2-forest is a spanning forest exactly containing two connected components. Intuitively, based on these interpretations, $r(s,t)$ is smaller (i.e., s and t having a short commute time thus they are easy to hit each other, or s and t frequently co-occurring in the same component of the spanning 2-forests), the more similar s and t are. In addition, compared to the other classic distance metrics on graphs (e.g., shortest path), resistance distance is robust with respect to noises (small perturbations on graphs, e.g. a few edges are removed or inserted), as it takes all paths into consideration.

Due to such nice properties, resistance distance has been widely used in many real-world applications, including recommendation systems [20, 27], link prediction [46], query suggestion [41] and query expansion [57] in information retrieval, graph kernels [43, 64], oblivious routing [48], and path planing in road networks [51]. Note that although it is argued in [54] that the resistance distance $r(s,t)$ is dominated by $\frac{1}{d_s}+\frac{1}{d_t}$ in large random geometric graphs, a simple correction will lead to a useful distance metric, which is defined as $\tilde{r}(s,t)=\sqrt{r(s,t)-\frac{1}{d_s}-\frac{1}{d_t}}-(\frac{1}{d_s}-\frac{1}{d_t})^2$ [54]. Clearly, the key to compute $\tilde{r}(s,t)$ is to calculate $r(s,t)$. Thus, in this paper, we focus mainly on the resistance distance computation problem. In addition, resistance distance is also widely used in solving many theoretical problems including spectral sparsification of graphs [52], spectral graph clustering [3], and max-flow computations [15], where approximating all-pair resistance distances is an important primitive for accelerating computation of these problems.

Although many efforts have been made on resistance distance in both theory and applications, there are very few studies on developing efficient algorithms to compute the resistance distance on large graphs. Most previous algorithms to calculate the resistance distance are based on computing the pseudo-inverse of Laplacian [8] which are very costly for large graphs. Recently, Peng et al. [44] developed several efficient algorithms to estimate the resistance

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD '23, June 18–23, 2023, Seattle, WA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

distance based on random walk sampling, and two of them are the current state-of-the-art (SOTA) algorithms. Specifically, their first SOTA algorithm is based on the commute time interpretation of resistance distance. To estimate the resistance distance $r(s,t)$, their algorithm needs to simulate round-trip random walks from s to t and back to s. Such an algorithm is very fast when $r(s,t)$ is small. However, when $r(s,t)$ is large, this algorithm is inefficient because in this case, it is not easy to obtain a round-trip random walk. Their second SOTA algorithm is based on estimating the K-step transition probability matrix of the random walk. However, the limitation of this algorithm is that on large graphs, it requires a large K to achieve a good estimation accuracy, thus rendering high time overheads of the algorithm.

To overcome these issues, we propose four novel and efficient algorithms to compute the resistance distance $r(s,t)$ for a single-pair of nodes (s,t), based on several interesting and newly-established connections among resistance distance, v-absorbed random walk, spanning 2-forests, and v-absorbed push procedure. Three of them (AbWalk, LocalTree, Bipush) are sampling-based approximation algorithms and one of them (Push) is a deterministic algorithm. A remarkable feature of our algorithms is that they are allowed to select a landmark node v which can be the *easy-to-hit* node by random walks on the graph (e.g., the highest-degree node). With such a nice feature, we can efficiently estimate $r(s,t)$ by sampling two v-absorbed random walks from s to v and from t to v, instead of sampling round-trip random walks. Since the landmark node v is often easy to hit by random walks, the time overheads of simulating two v-absorbed random walks $s \rightsquigarrow v$ and $t \rightsquigarrow v$ are much lower than those of simulating round-trip random walks, thus the resulting algorithm AbWalk significantly improves the efficiency of the commute time based algorithm [44]. With a new and deep connection between resistance distance and spanning 2-forests, we develop a novel and local spanning tree sampling algorithm LocalTree which again only requires to simulate two random walks from both s and t to the landmark node v. In addition, we also develop a new deterministic variant of the v-absorbed random walk, called v-absorbed push. Based on the v-absorbed push procedure, we propose a new deterministic algorithm Push and a bidirectional algorithm Bipush which integrates both v-absorbed push and v-absorbed random walk sampling to efficiently compute $r(s,t)$. The efficiency of these two push-based algorithms can also benefit from the idea of selecting an easy-to-hit landmark node v.

In applications of link prediction [46] and recommendation systems [20], we typically need to compute the resistance distance from a given query node s to all other nodes. To solve such a single-source resistance distance query problem, existing algorithms [44] require to perform $n-1$ single-pair resistance distance queries, which is clearly inefficient for large graphs. To tackle this problem, we propose a new online approximate algorithm based on sampling of random spanning trees. Our algorithm is based on an interesting connection between the v-absorbed random walk and the classic loop-erased random walk [61] for random spanning tree sampling (see Section 5.2 for details). We prove that the time complexity of the proposed algorithm is lower than those of the algorithms based on processing $n-1$ single-pair queries. To further improve the efficiency, we also develop two novel index-based approximate algorithms. We show that the running time of our index-based algorithms for single-source query is almost the same as the algorithms for single-pair query (with only an $O(n)$ additional term to output the results). Moreover, our index takes only $O(n)$ space, and it can be constructed by running only one single-source query.

We conduct extensive experiments on 5 real-life graphs to evaluate the proposed algorithms. The results show that (1) for single-pair query, our best algorithm not only achieves more than two orders of magnitude speedup over the SOTA algorithms [44] on large graphs, but also has much lower estimation errors; and (2) for single-source query, our online algorithm is significantly faster than the baseline algorithms, while the proposed index-based algorithms are at least three orders of magnitude faster than our online algorithm. To summarize, the main contributions of this paper are as follows.

New theoretical results. We first derive a new formula to compute the resistance distances which relies on a selected landmark node v. Then, based on the new formula, we establish several novel connections among resistance distance, a new concept called v-absorbed random walk, spanning 2-forests, and a newly-proposed v-absorbed push procedure. Such novel and deep connections provide several interesting combinatorial explanations of resistance distance, based on which we can develop efficient algorithms to estimate the resistance distance. We believe that these novel combinatorial explanations of resistance distance could be of independent interest.

Novel algorithms for resistance distance queries. We propose four novel algorithms to answer the single-pair query, including a v-absorbed random walk sampling algorithm AbWalk, a local spanning tree sampling algorithm LocalTree, a v-absorbed push algorithm Push and a bidirectional algorithm Bipush that combines v-absorbed push and v-absorbed random walk sampling. Tabel 1 summarizes the time complexity of all the proposed algorithms as well as the state-of-the-art algorithms for single-pair query. Except Push, all other algorithms are sampling-based approximate algorithms. Push is a deterministic algorithm with an additive error bound. The time complexity of all our algorithms relies mainly on the hitting time of the random walk from both s and t to the landmark node v, which is often lower than the commute time between s and t. For single-source query, we develop three new approximate algorithms, including an online algorithm LEwalk and two index-based algorithms AbWalk* and Push*. We show that our index-based algorithms are extremely fast which can answer single-source queries with time costs similar to the algorithms for answering single-pair query.

Extensive experiments. We conduct comprehensive experiments using 10 real-life graphs to evaluate the proposed algorithms. The results show that our algorithms substantially outperform the SOTA algorithms in terms both running time and estimation errors. In addition, we also conduct two case studies to evaluate the effectiveness of the resistance distance related metrics. The results show that the corrected resistance distance is indeed a very good node-similarity metric; and the proposed techniques are very useful for computing such a resistance distance related metric. For reproducibility purpose, the source code of this paper is released at an anonymous link https://github.com/mhLeon/Resistance-Landmark.

2 PRELIMINARIES

2.1 Problem Definition

Given an undirected, connected graph $G=(V,E)$ with $|V|=n$ nodes and $|E|=m$ edges. We use e_i to denote the unit vector where the i-th element equals 1, and the other elements are 0. We use A to denote the adjacency matrix of G; use D to denote the degree matrix of G, where $D_{ii}=d_i$ is the degree of node i. Let $L=D-A$ be the Laplacian matrix of G. Denote by $L=\sum_{i=1}^{n}\lambda_i\vec{u}_i\vec{u}_i^T$ the eigen-decomposition of L, where $0=\lambda_1<\lambda_2\leq\cdots\leq\lambda_n$ are the eigenvalues of L and $\vec{u}_i$ is the eigenvector corresponding to the eigenvalue λ_i. Let $L(u|v)$ be the submatrix of L after deleting the u-th row and v-th column of L. When $u=v$, it is simplified as L_v. Similarly, denote by $L(u_1,v_1|u_2,v_2)$ the submatrix of L after deleting the u_1-th and v_1-th row of L and the u_2-th and v_2-th column of L.

To define the resistance distance, we regard the graph as an electrical network, where each edge represents a unit resistor and each node represents a junction that connects resistors. For readers who are unfamiliar with the concepts of electrical networks

Table 1: Time Complexity of the state-of-the-art algorithms as well as the proposed algorithms for computing $r(s,t)$. Here v is an easy-to-hit landmark node (e.g., the highest-degree node), $h(s,t)$ denotes the hitting time from s to t, $\kappa(s,t) = h(s,t) + h(t,s)$, and $\kappa_v(s,t) = h(s,v) + h(t,v)$. Note that $\kappa_v(s,t)$ is often much smaller than $\kappa(s,t)$. T is the sample size and K is decided by the mixing time of G [44]. To achieve a small accuracy, K is often very large, and $r_{\max}$ controls the accuracy of Push and Bipush.

	Approximate Solutions					**Deterministic Solution**
Algorithm	Akp [44]	Commute [44]	AbWalk	LocalTree	Bipush	Push
Time Complexity	$O(TK^2)$	$O(T \times \kappa(s,t))$	$O(T \times \kappa_v(s,t))$	$O(T \times \kappa_v(s,t))$	$O(\frac{\kappa_v(s,t)}{r_{\max}} + T \times \kappa_v(s,t))$	$O(\frac{\kappa_v(s,t)}{r_{\max}})$

and resistance distance, we refer them to [8, 18] which provide a comprehensive introduction to these concepts. Generally, suppose that a unit of current flows in at s and out at t, we want to determine the current along all edges and the potential on each node in G. The resistance distance $r(s,t)$ is the potential difference between s and t. Let p be the potential vector which represents each node's potential. Then, by the Kirchhoff's current law (KCL) and Kirchhoff's voltage law (KVL), we can obtain that $Lp = e_s - e_t$ [19]. As a result, the resistance distance $r(s,t) = p_s - p_t$ can further be derived by the Moore-Penrose pseudo-inverse of the Laplacian matrix $L^\dagger \triangleq \sum_{i=2}^{n} \frac{1}{\lambda_i} \vec{u}_i \vec{u}_i^T$ as follows:

$$r(s,t) = (e_s - e_t)^T L^\dagger (e_s - e_t) = (L^\dagger)_{ss} + (L^\dagger)_{tt} - 2(L^\dagger)_{st}. \quad (1)$$

By Ohm's law, the current flow along an arbitrary edge $e = (u_1, u_2)$, denoted by $I(u_1, u_2, s, t)$, is the potential difference between u_1 and u_2. Note that the current flow along $e = (u_1, u_2)$ has a direction which can be either positive or negative. By the result shown in [11], the current flow can also be derived by $L^\dagger$:

$$\begin{aligned} I(u_1, u_2, s, t) &= (e_{u_1} - e_{u_2})^T L^\dagger (e_s - e_t) \\ &= (L^\dagger)_{u_1 s} - (L^\dagger)_{u_2 s} - (L^\dagger)_{u_1 t} + (L^\dagger)_{u_2 t}. \quad (2) \end{aligned}$$

Let $\mathcal{P}_{st} = \{s = u_1, \cdots, u_l = t\}$ be an l-length path between s and t in G. Then, by Eq. (1) and Eq. (2), we can derive the following result.

Lemma 2.1. *$r(s,t) = \sum_{k=1}^{l-1} I(u_l, u_{l+1}, s, t)$.*

Proof. The lemma can be proved by the following equalities.

$$\begin{aligned} \sum_{k=1}^{l-1} I(u_k, u_{k+1}, s, t) &= \sum_{k=1}^{l-1} ((L^\dagger)_{su_k} - (L^\dagger)_{su_{k+1}} - (L^\dagger)_{u_k t} + (L^\dagger)_{u_{k+1} t}) \\ &= (L^\dagger)_{ss} + (L^\dagger)_{tt} - (L^\dagger)_{st} - (L^\dagger)_{ts} = r(s,t). \end{aligned}$$

□

Lemma 2.1 shows that the resistance distance $r(s,t)$ is equal to the sum of all the current flows on the edges of any $s \sim t$ path.

Given a graph G, a simple random walk on G is a stochastic procedure where in each step a node u walks to a neighbor of u with a probability $\frac{1}{d_u}$. The commute time $\kappa(s,t)$ is defined as the expected number of steps of a random walk that starts at s, visits t, and finally comes back to s. It is well known that the resistance distance is closely related to the commute time of the random walk on the graph [53].

Theorem 2.2. *[53] $\kappa(s,t) = 2m \times r(s,t)$.*

Clearly, the smaller $r(s,t)$ is, the closer s and t are. Moreover, the resistance distance was shown to be a distance metric [12, 53]. Compared to the shortest path distance, resistance distance takes all paths into consideration, thus it is often more robust. Due to such nice properties, the resistance distance is widely used in many real-world applications, including query suggestion [41] and query expansion [57] in information retrieval, recommendation systems [20, 27], graph kernels [43, 64], oblivious routing [48], and path planing in road networks [51]. In those applications, it is often need to compute the resistance distance of a pair of nodes s and t, or compute the resistance distance from a source node s to all the other nodes in G. In this paper, we focus on both the single-pair resistance distance query and the single-source resistance distance query problems. Formally, we define these two problems as follows.

Definition 2.3. (Single-pair resistance distance query) Given a graph G and a pair of nodes (s,t) with $s \neq t$, the problem of single-pair resistance distance query is to compute the resistance distance $r(s,t)$.

Definition 2.4. (Single-source resistance distance query) Given a graph G and a source node s, the single-source resistance distance query problem is to compute $r(s,u)$ for all $u \in V$.

2.2 Existing solutions and their limitations

According to the above discussions, computing the resistance distance requires solving a linear Laplacian system $L\vec{x} = \vec{b}$, or equivalently computing the pseudo-inverse $L^\dagger$ [21]. Such a linear Laplacian system was well studied in the theoretical computer science community. Although much progress had been made, the fastest Laplacian solver still consumes $\tilde{O}(m \log n)$, which is still very costly for large real-world graphs (e.g., graphs with more than 1 million nodes).

Recently, several local algorithms for approximating the resistance distance are proposed in [44], which can answer a single-pair query by only exploring a small portion of the graph. Among all the methods proposed in [44], there are two most efficient algorithms. The first algorithm Commute is based on estimating the commute time of the random walk (Theorem 2.2). Specifically, the algorithm simulates random walks from s to t and back to s, and then estimates the expected steps of such round-trip random walks. Since each round-trip random walk consumes $O(\kappa(s,t))$ time, the time complexity of simulating T such round-trip random walks is $O(T \times \kappa(s,t)) = O(2mT \times r(s,t))$. This algorithm is fast when $r(s,t)$ is small [44]. However, when $r(s,t)$ is large, such a round-trip random walk based algorithm is very costly.

The second algorithm proposed in [44] is based on the transition probability matrix $P \triangleq D^{-1}A$ of the random walk. Peng et al. [44] show that the resistance distance can be represented by the transition probability matrix P as follows:

$$r(s,t) = (e_s - e_t)^T L^\dagger (e_s - e_t) = (e_s - e_t)^T (\sum_{k=0}^{\infty} P^k D^{-1})(e_s - e_t). \quad (3)$$

To estimate $r(s,t)$, we can truncate the summation with a sufficient large integer K in Eq. (3). The resistance distance then can be estimated either by sampling simple random walks with length no large than K or sampling collision random walks [44]. Such an algorithm is called Akp. Suppose that we draw T samples to estimate the transition probability and take the average. Clearly, for estimating P^i, we need to simulate a random walk with length i. In each sample, the random walk length can be bounded by $\sum_{i=1}^{K} i = O(K^2)$. Thus, the time complexity of Akp can be bounded $O(TK^2)$. However, the drawback of Akp is that in large real-life graphs, it often requires a large K to achieve a good estimation accuracy. As a result, the running time of the algorithm can be long.

Furthermore, all the state-of-the-art algorithms can only handle single-pair resistance distance query, and they are often very hard to efficiently extended to handle single-source query. In particular, to process a single-source query, these algorithms need to compute $O(n)$ single-pair queries, which is very costly for large graphs. To overcome these limitations, we will propose several novel and more efficient algorithms to handle both single-pair and single-source resistance distance queries.

3 NEW THEORETICAL RESULTS

In this section, we establish several new connections between resistance distance, random walk and spanning 2-forests. These new theoretical results lead to two efficient estimators (Lemma 3.5 and Lemma 3.9) for resistance distance, as well as a deterministic push algorithm for resistance distance computations.

3.1 New formula for computing resistance distance

It is well known that for any connected undirected graph, the Laplacian matrix L has a rank $n-1$, i.e., $\text{rank}(L) = n-1$, and thereby its inverse does not exist. As a result, to exactly compute the resistance distance, existing algorithms often relies on computing the Moore-Penrose pseudo-inverse of L which is defined as $L^\dagger = \sum_{i=2}^{n} \frac{1}{\lambda_i} \vec{u}_i \vec{u}_i^T$. It is easy to verify that $L^\dagger$ satisfies four properties [8]: $LL^\dagger L = L$, $L^\dagger L L^\dagger = L^\dagger$, $(LL^\dagger)^T = LL^\dagger$, $(L^\dagger L)^T = L^\dagger L$. Interestingly, although L is not invertible, a submatrix L_v of L is invertible for any node v. The following theorem indicates that we can represent resistance distance as well as the related quantities in terms of L_v^{-1}:

Theorem 3.1. *Let $\vec{x}_1$ and $\vec{x}_2$ be two vectors that are orthogonal to an all-one vector $\vec{1} = [1, \cdots, 1]^T$, i.e., $\vec{1}^T \vec{x}_1 = 0$, $\vec{1}^T \vec{x}_2 = 0$. Then, for any node v in G, we have:*

$$\vec{x}_1^T L^\dagger \vec{x}_2 = \vec{x}_1^T \begin{bmatrix} L_v^{-1} & \vec{0} \\ \vec{0}^T & 0 \end{bmatrix} \vec{x}_2, \tag{4}$$

where we assume without loss of generality that v is the last node arranged in L.

Proof. By definition, we can rephrase L as $L = \begin{bmatrix} L_v & \vec{x} \\ \vec{x}^T & d_v \end{bmatrix}$. Then, it is easy to derive that $L \begin{bmatrix} L_v^{-1} & \vec{0} \\ \vec{0}^T & 0 \end{bmatrix} L = \begin{bmatrix} L_v & \vec{x} \\ \vec{x}^T & \vec{x}^T L_v^{-1} \vec{x} \end{bmatrix}$. Since $\text{rank}(L) = \text{rank}(\begin{bmatrix} L_v & \vec{x} \\ \vec{x}^T & d_v \end{bmatrix}) = \text{rank}(\begin{bmatrix} L_v & \vec{x} \\ \vec{0}^T & d_v - \vec{x}^T L_v^{-1} \vec{x} \end{bmatrix}) = n-1$ and $\text{rank}(L_v) = n-1$, we have $d_v - \vec{x}^T L_v^{-1} \vec{x} = 0$. As a consequence, we have $L \begin{bmatrix} L_v^{-1} & \vec{0} \\ \vec{0}^T & 0 \end{bmatrix} L = L$.

Recall that $L\vec{1} = 0$ and $\text{rank}(L) = n-1$, we can conclude that $\vec{1}$ is the only vector in the null space of L. Since $\vec{x}_1$ and $\vec{x}_2$ are orthogonal to $\vec{1}$, both $\vec{x}_1$ and $\vec{x}_2$ are in the column space of L. Therefore, there exist two vectors $\vec{z}_1, \vec{z}_2$ such that $L\vec{z}_1 = \vec{x}_1$ and $L\vec{z}_2 = \vec{x}_2$. Then, we have

$$\begin{aligned} \vec{x}_1^T (L^\dagger - \begin{bmatrix} L_v^{-1} & \vec{0} \\ \vec{0}^T & 0 \end{bmatrix}) \vec{x}_2 &= \vec{z}_1^T L (L^\dagger - \begin{bmatrix} L_v^{-1} & \vec{0} \\ \vec{0}^T & 0 \end{bmatrix}) L \vec{z}_2 \\ &= \vec{z}_1^T (L L^\dagger L - L \begin{bmatrix} L_v^{-1} & \vec{0} \\ \vec{0}^T & 0 \end{bmatrix} L) \vec{z}_2 \\ &= \vec{z}_1^T (L - L) \vec{z}_2 \\ &= 0. \end{aligned}$$

This completes the proof. □

Note that Theorem 3.1 holds for any node v in G. For convenience, we refer to such a node v as a *landmark* node. Based on Theorem 3.1, we can derive a new formula to compute the resistance distance, as shown in the following corollary.

Corollary 1. *For any two nodes $s, t \neq v$, we have*

$$r(s,t) = (L_v^{-1})_{ss} + (L_v^{-1})_{tt} - (L_v^{-1})_{st} - (L_v^{-1})_{ts}. \tag{5}$$

For any node $u \neq v$, we have

$$r(u,v) = (L_v^{-1})_{uu} = (L_u^{-1})_{vv}, \tag{6}$$

Proof. Since the vector $e_s - e_t$ is orthogonal to $\vec{1}$, we can set $\vec{x}_1 = \vec{x}_2 = e_s - e_t$ in Theorem 3.1. Then, we can obtain that $r(s,t) = (L_v^{-1})_{ss} + (L_v^{-1})_{tt} - (L_v^{-1})_{st} - (L_v^{-1})_{ts}$. Similarly, by setting $\vec{x}_1 = \vec{x}_2 = e_u - e_v$ in Theorem 3.1, we are able to derive that $r(u,v) = (e_u - e_v)^T L^\dagger (e_u - e_v) = (L_v^{-1})_{uu} = (L_u^{-1})_{vv}$. □

Eq. (5) and Eq. (6) apparently give a new approach to compute the resistance distance. The most appealing feature is that no matter which v we choose, adding or subtracting four elements in the resulting inverse of the submatrix L_v maintains an invariant $r(s,t)$.

More interestingly, if we set $\vec{x}_1 = (e_{u_1} - e_{u_2})$ and $\vec{x}_2 = e_s - e_t$ in Theorem 3.1, the current flow on the edge (u_1, u_2) also maintains an invariant $I(u_1, u_2, s, t)$ for any landmark node v. Specifically, we have the following corollary.

Corollary 2. *For any four nodes $s, t, u_1, u_2 \neq v$, we have*

$$I(u_1, u_2, s, t) = (L_v^{-1})_{u_1 s} - (L_v^{-1})_{u_2 s} - (L_v^{-1})_{u_1 t} + (L_v^{-1})_{u_2 t}. \tag{7}$$

For any three nodes $u, u_1, u_2 \neq v$, we have

$$I(u_1, u_2, u, v) = (L_v^{-1})_{u u_1} - (L_v^{-1})_{u u_2}. \tag{8}$$

Proof. The corollary can be easily proved based on the Eq. (2) and the results in Theorem 3.1. □

The above two corollaries show that both the resistance distance and the current flow can be calculated by L_v^{-1}. However, computing the inverse of L_v is often very expensive, thus the algorithm only works for very small graphs. To overcome this problem, we will develop several interesting combinatorial explanations for L_v^{-1} in the following section, which result in novel and efficient algorithms to compute the resistance distance.

3.2 A v-absorbed random walk interpretation

In this subsection, we give a new combinatorial explanation for the elements of L_v^{-1} based on a concept called v-absorbed random walk. Based on such a new explanation, we are able to develop an efficient algorithm by sampling v-absorbed random walks to compute the resistance distance.

Definition 3.2. (v-absorbed random walk) Given a graph G and a node v, a v-absorbed random walk is a random walk that starts from an arbitrary node s and terminates when it hits the node v.

Let $\tau_v[s,u]$ be the expected number of visits on u for a v-random walk start from s. We define the degree-normalized expected number of visits on u for a v-random walk as $\tilde{\tau}[s,u] = \frac{\tau_v[s,u]}{d_u}$. Below, we show that $(L_v^{-1})_{su}$ has an interesting combinatorial explanation in terms of $\tilde{\tau}[s,u]$.

Let $P = D^{-1}A$ be the transition probability matrix of the traditional random walk. Then, we have $L = D(I-P)$. Let P_v be the submatrix of P which is obtained by deleting the v-th row and the v-th column of P. Denote by D_v the diagonal degree matrix except the node v. It is easy to verify that $L_v = D_v(I - P_v)$. The following lemma establishes a connection between $\tau_v[s,u]$ and P_v.

Lemma 3.3. $\tau_v[s,u] = (I - P_v)^{-1}_{su}$.

Proof. Let $p_v^k(s,u)$ be the probability that a v-absorbed random walk starts from s and passes u at the k-th step. Clearly, by definition, $(P_v)_{su}$ is the probability that s jumps to u in a certain random walk step. Note that in each row corresponding to a node $u \in N(v)$, the probabilities in such a row of P_v does not sum up to 1 (less than 1), because there exists a probability to hit the absorbed node v for $u \in N(v)$. Then, by definition, we have $p_v^k(s,u) = (P_v^k)_{su}$. As a result, $\tau_v[s,u] = \sum_{k=0}^{\infty} p_v^k(s,u) = \sum_{k=0}^{\infty} (P_v^k)_{su} = (I - P_v)^{-1}_{su}$. □

Based on Lemma 3.3, we can obtain the following result.

Lemma 3.4. $\tilde{\tau}[s,u] = \frac{\tau_v[s,u]}{d_u} = (L_v^{-1})_{su}$.

Proof. Following the definition of $\tilde{\tau}[s,u]$ and Lemma 3.3, we have $\tilde{\tau}[s,u] = \frac{\tau_v[s,u]}{d_u} = ((I - P_v)^{-1} D_v^{-1})_{su} = (L_v^{-1})_{su}$. □

By Corollary 1 and Lemma 3.4, we can derive that

$$r(s,t) = \tilde{\tau}_v[s,s] + \tilde{\tau}_v[t,t] - \tilde{\tau}_v[s,t] - \tilde{\tau}_v[t,s]. \qquad (9)$$

Armed with Eq. (9), we can compute the resistance distance $r(s,t)$ by sampling v-absorbed random walks. Specifically, we are able to derive a novel unbiased estimator for $r(s,t)$ by sampling v-absorbed random walks.

Lemma 3.5. *For $v \neq s, t$, suppose that we simulate a v-absorbed random walk from s (t, resp.). Let X_s (Y_s, resp.) be the number of visits on s and X_t (Y_t, resp.) be the number of visits on t by such a v-absorbed random walk. Then, $\hat{r} = \frac{X_s}{d_s} - \frac{X_t}{d_t} - \frac{Y_s}{d_s} + \frac{Y_t}{d_t}$ is an unbiased estimator of $r(s,t)$, i.e., $E[\hat{r}] = r(s,t)$.*

Proof. By the linearity of expectation, we have $E[\hat{r}] = E[\frac{X_s}{d_s} - \frac{X_t}{d_t} - \frac{Y_s}{d_s} + \frac{Y_t}{d_t}] = E[\frac{X_s}{d_s}] - E[\frac{X_t}{d_t}] - E[\frac{Y_s}{d_s}] + E[\frac{Y_t}{d_t}] = (L_v^{-1})_{ss} - (L_v^{-1})_{st} - (L_v^{-1})_{ts} + (L_v^{-1})_{tt} = r(s,t)$. □

3.3 A spanning forest interpretation

In this subsection, we first give a spanning forest interpretation for the resistance distance $r(s,t)$, based on the classic matrix-tree theorem [8, 13]. Then, we show that such a spanning forest interpretation is not sufficient to derive an efficient estimator for $r(s,t)$. To achieve this, we establish a novel connection between spanning tree and current flow on the graph, based on which we are able to construct efficient estimator for $r(s,t)$.

Given a graph G with n nodes, a spanning tree is a connected subgraph of G which has n nodes and $n-1$ edges. Let $\mathcal{T}$ be the set of spanning trees of G. The classic matrix-tree theorem states that the number of spanning trees in a graph G, denoted by $|\mathcal{T}|$, is equal to the determinant of L_v for any node $v \in G$ [8], i.e., $\det(L_v) = |\mathcal{T}|$.

A spanning forest is a subgraph of G that has n nodes and it does not contain any cycle. Clearly, a spanning forest may have several connected components (each connected component is a tree). If a spanning forest exactly has two connected components, we refer to it as a spanning 2-forest. Given two nodes s, t, we use $\mathcal{F}_{s|t}$ to denote the set of spanning 2-forests such that s is in one connected component, and t is in the other component. Similarly, denote by $\mathcal{F}_{v|s,t}$ the set of spanning 2-forests such that v is in one connected component, and both s and t are in the other component. Then, the classic all-minors matrix tree theorem [13] states that

$$\det(L(v,s|v,t)) = |\mathcal{F}_{v|s,t}|, \qquad (10)$$

where $\det(L(v,s|v,t))$ is the determinant of the matrix $L(v,s|v,t)$ (obtained by deleting v-th, s-th rows and v-th, t-th columns of L).

Note that s and t in Eq. (10) can be the same, i.e., $s = t = u$. In this case, the spanning forest set will degrade to $\mathcal{F}_{v|u}$. Based on this, we can obtain a well-known result.

Theorem 3.6. *[8] $r(s,t) = \frac{|\mathcal{F}_{s|t}|}{|\mathcal{T}|}$.*

Proof. By Eq. (6), we have $r(s,t) = (L_s^{-1})_{tt} = \frac{det(L(s|t))}{det(L_s)} = \frac{|\mathcal{F}_{s|t}|}{|\mathcal{T}|}$, where the second equality is due to the Cramer's rule. □

Theorem 3.6 shows that the resistance distance $r(s,t)$ is proportional to the number of spanning 2-forests with s and t belonging to two different components. Further, based on Theorem 3.6, we can derive an interesting result on resistance distance and spanning 2-forests when a landmark node v is considered.

Theorem 3.7. *For $s, t \neq v$, we have $r(s,t) = \frac{|\mathcal{F}_{v|s}| + |\mathcal{F}_{v|t}| - 2|\mathcal{F}_{v|s,t}|}{|\mathcal{T}|}$.*

Proof. To prove the lemma, it is sufficient to show $|\mathcal{F}_{s|t}| = |\mathcal{F}_{v|s}| + |\mathcal{F}_{v|t}| - 2|\mathcal{F}_{v|s,t}|$. Note that the set of spanning 2-forests $|\mathcal{F}_{s|t}|$ can be divided into two categories: i) v and s are in the same component, and ii) v and t are in the same component. Clearly, we have $|\mathcal{F}_{s|t}| = |\mathcal{F}_{s,v|t}| + |\mathcal{F}_{s|t,v}|$. Since $|\mathcal{F}_{s,v|t}| = |\mathcal{F}_{v|t}| - |\mathcal{F}_{v|s,t}|$ and $|\mathcal{F}_{s|t,v}| = |\mathcal{F}_{v,t|s}| = |\mathcal{F}_{v|s}| - |\mathcal{F}_{v|s,t}|$, we have $|\mathcal{F}_{s|t}| = |\mathcal{F}_{v|s}| + |\mathcal{F}_{v|t}| - 2|\mathcal{F}_{v|s,t}|$. This completes the proof. □

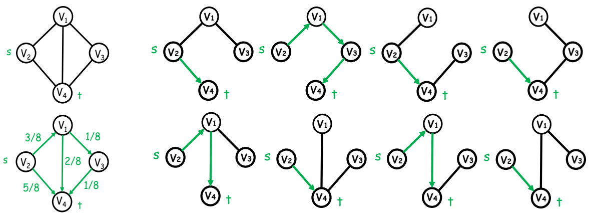
Note that both Theorem 3.6 and Theorem 3.7 provide a spanning forest explanation of resistance distance $r(s,t)$, but both of them are hard to use to compute $r(s,t)$ in practice. The reason is that both Theorem 3.6 and Theorem 3.7 require to exactly count the spanning 2-forests, which is often costly for large graphs. One possible method is to uniformly sample the spanning 2-forest, and then construct an unbiased estimator to approximate the count of the spanning 2-forests. But unfortunately, no efficient algorithm that can uniformly sample a spanning 2-forest is known, because the distribution of the spanning 2-forests of a graph is very complicated [9].

Interestingly, if we only need to compute the resistance distance for each edge (s,t) in G, then we can efficiently estimate $r(s,t)$ based on the classic random spanning tree sampling algorithm [22, 61]. This is because there is a one-to-one mapping between a spanning 2-forest in $\mathcal{F}_{s|t}$ and a spanning tree Γ containing an edge (s,t). To see this, we can remove the edge $(s,t) \in \Gamma$ to obtain a spanning 2-forest in $\mathcal{F}_{s|t}$; for a spanning 2-forest in $\mathcal{F}_{s|t}$, we can add back the edge (s,t) into the spanning 2-forest which results in a spanning tree. As a consequence, if there is an edge (s,t) in G, sampling a spanning 2-forest in $\mathcal{F}_{s|t}$ is equivalent to sampling a spanning tree Γ that contains the edge (s,t). By applying Theorem 3.6, we can obtain an unbiased estimator for $r(s,t)$ based on the random spanning tree sampling (compute the proportion of sampled spanning tree that contains (s,t)).

However, if $(s,t) \notin E$, there does not exist a one-to-one mapping between a spanning 2-forest in $\mathcal{F}_{s|t}$ and the spanning tree Γ, thus existing spanning tree sampling techniques cannot be used to estimate $r(s,t)$. Moreover, as we discussed previously, no algorithm that can sample spanning 2-forests uniformly is known. To circumvent this challenging problem, we develop a novel technique based on an interesting connection of the spanning tree of a graph and the current flow on an electrical network.

Clearly, for any two nodes s, t, there is a unique simple path between s and t in a spanning tree of G. Let $\mathcal{T}^{s,t}_{u_1,u_2}$ be the set of spanning trees that contains an edge (u_1, u_2) and the path from s to t in the tree passes along the edge (u_1, u_2). Suppose that a unit of current flows in at s and out at t. Denote by $I(u_1, u_2, s, t)$ the current flow along the edge $e = (u_1, u_2)$. Then, we have the following result.

Lemma 3.8. *Given two distinct nodes s and t, for each edge $(u_1, u_2) \in E$, we have $I(u_1, u_2, s, t) = \frac{|\mathcal{T}^{s,t}_{u_1,u_2}| - |\mathcal{T}^{s,t}_{u_2,u_1}|}{|\mathcal{T}|}$.*



(a) G, an s-t-flow (b) s-t-flow on spanning trees

Figure 1: Illustration of sending flows on spanning trees. (a) A graph G and an s-t-flow when a unit current flows in v_2 and flows out v_4; (b) There are 8 spanning trees. The current flows along an edge (u, v) in G equals the average current flows on (u, v) on each spanning tree.

Proof. Recall that by Eq. (8), we have $I(u_1, u_2, s, t) = (L_t^{-1})_{su_1} - (L_t^{-1})_{su_2}$. Since $(L_t^{-1})_{su_1} = \frac{\det(L(t,s|t,u_1))}{\det(L_t^{-1})}$ (by the Cramer's rule), we have $(L_t^{-1})_{su_1} = \frac{|\mathcal{F}_{t|s,u_1}|}{|\mathcal{T}|}$ based on the all minors matrix-tree theorem (see Eq. (10)). As a result, we have $I(u_1, u_2, s, t) = \frac{|\mathcal{F}_{t|s,u_1}| - |\mathcal{F}_{t|s,u_2}|}{|\mathcal{T}|}$. Note that we can partition the spanning forest set $\mathcal{F}_{t|s,u_1}$ into two subsets based on which component u_2 belongs to. That is, $|\mathcal{F}_{t|s,u_1}| = |\mathcal{F}_{t,u_2|s,u_1}| + |\mathcal{F}_{t|s,u_1,u_2}|$. Similarly, we have $|\mathcal{F}_{t|s,u_2}| = |\mathcal{F}_{t,u_1|s,u_2}| + |\mathcal{F}_{t|s,u_1,u_2}|$. With these equalities, we can derive that $|\mathcal{F}_{t|s,u_1}| - |\mathcal{F}_{t|s,u_2}| = |\mathcal{F}_{s,u_1|u_2,t}| - |\mathcal{F}_{s,u_2|u_1,t}|$, and thereby $I(u_1, u_2, s, t) = \frac{|\mathcal{F}_{s,u_1|u_2,t}| - |\mathcal{F}_{s,u_2|u_1,t}|}{|\mathcal{T}|}$. Then, we claim that there is a one-to-one mapping between $F \in \mathcal{F}_{s,u_1|u_2,t}$ and $\Gamma \in \mathcal{T}^{s,t}_{u_2,u_1}$. To see this, remove the edge (u_1, u_2) in the unique path between s and t in Γ, it will obtain a spanning 2-forest such that s, u_1 and t, u_2 belong to different components. Reversely, if we add an edge (u_1, u_2) in F, we will obtain a spanning tree, where the unique path between s and t must contain the edge (u_1, u_2). Putting it all together, we have $I(u_1, u_2, s, t) = \frac{|\mathcal{F}_{s,u_1|u_2,t}| - |\mathcal{F}_{s,u_2|u_1,t}|}{|\mathcal{T}|} = \frac{|\mathcal{T}^{s,t}_{u_1,u_2}| - |\mathcal{T}^{s,t}_{u_2,u_1}|}{|\mathcal{T}|}$. This completes the proof. □

Lemma 3.8 suggests that we can estimate the current $I(u_1, u_2, s, t)$ by sampling spanning trees. Specifically, when we sample a spanning tree Γ, we can obtain a unique $s \sim t$ path from Γ. Then, we can calculate the proportion of the spanning trees in $\mathcal{T}^{s,t}_{u_1,u_2}$ over all the samples (similar computation for $\mathcal{T}^{s,t}_{u_2,u_1}$) which results in an unbiased estimator for $I(u_1, u_2, s, t)$ based on Lemma 3.8. As an intuitive example shown in Fig. 1, the current flowing along an edge (u, v) in a graph G can be estimated by randomly sampling spanning trees, and the s-t-flow on each spanning tree is an unbiased estimator of the current flowing from s to t on G. Armed with this estimator, we are able to estimate the resistance distance $r(s, t)$ for any two distinct nodes s and t, based on the results in Lemma 2.1 by summing all estimated current along a fixed $s \sim t$ path $\mathcal{P}_{st}$. More specifically, we can derive the following unbiased estimator for $r(s, t)$.

Lemma 3.9. *Given two distinct nodes s and t, we let $\mathcal{P}_{st}$ be a fixed path between s and t. Suppose that a spanning tree $T \in \mathcal{T}$ is sampled uniformly and $\tilde{\mathcal{P}}_{st}$ is the unique path between s and t in T. Let X_+ be the number of edges that belong to $\mathcal{P}_{st}$ and appear in $\tilde{\mathcal{P}}_{st}$, X_- be the number of edges that belong to $\mathcal{P}_{st}$ and appear in $\tilde{\mathcal{P}}_{st}$ as an opposite direction, then $\hat{r} = X_+ - X_-$ is an unbiased estimator of $r(s, t)$, i.e., $E[\hat{r}] = r(s, t)$.*

Proof. With the fixed path $\mathcal{P}_{st} = \{s = u_1, u_2, \cdots, u_l = t\}$, we have $r(s, t) = \sum_{k=1}^{l-1} I(u_k, u_{k+1}, s, t)$ by Lemma 2.1. For each edge (u_k, u_{k+1}) in $\mathcal{P}_{st}$, we denote X_k as a random variable. If the edge (u_k, u_{k+1}) appears in the unique path $\tilde{\mathcal{P}}_{st}$ in a sampled spanning tree T, then $X_k = 1$, which corresponds to a spanning tree in $\mathcal{T}^{s,t}_{u_k,u_{k+1}}$. If the opposite direction of the edge (u_k, u_{k+1}) appears in the unique path $\tilde{\mathcal{P}}_{st}$ in T (or equivalently (u_{k+1}, u_k) appears in the $s \sim t$ path $\tilde{\mathcal{P}}_{st}$), then $X_k = -1$, which corresponds to a spanning tree in $\mathcal{T}^{s,t}_{u_{k+1},u_k}$. By the linearity of expectation, we have $E[\hat{r}] = E[X_+ - X_-] = \sum_{k=1}^{l-1} E[X_k] = \sum_{k=1}^{l-1} I(u_k, u_{k+1}, s, t) = r(s, t)$. □

3.4 A deterministic v-absorbed push procedure

Recall that on a node u, the personalized PageRank random walk stays at u with probability α, and with probability $1 - \alpha$ randomly jumps to one of its neighbor. Such a random walk can be interpreted as a *deterministic push procedure* [5, 10, 34, 35]. Specifically, let $r[u]$ be the value of a node u (initially $r[u] = 0$ if u is not a source node, and $r[s] = 1$ if s is the source). Then, the *deterministic push procedure* propagates $(1 - \alpha) \times r[u]/d_u$ to each of its neighbor, and reserves $\alpha \times r[u]$ at u. The push procedure terminates when no nodes's value is changed. It was shown that the reserved value on each node is a good approximation of its PageRank value when the push procedure terminates. The key property of such a push procedure is that an invariant $p(s, w) = \hat{p}(s, w) + \sum_{u \in V} r_s[u] p(u, w)$ is maintained during the push procedure, where $p(s, w)$ is the exact personalized PageRank value of w with respect to source s, $\hat{p}(s, w)$ denotes the reserved value on w. Such an invariant property guarantees the correctness of the push algorithm.

Motivated by the deterministic push procedure for personalized PageRank, we propose a novel deterministic v-absorbed push procedure which can be regarded as a deterministic variant of the v-absorbed random walk. Unlike the personalized PageRank random walk, on each node u, there is no probability of the walker staying at u for the v-absorbed random walk (all the probability masses are uniformly propagated to u's neighbors). As a result, no value is reserved at u and the invariant maintained by the personalized PageRank push procedure cannot be directly generalized to the v-absorbed random walk.

To achieve our goal, the challenging issues needed to be tackled are (1) how to define a *push operator* for the v-absorbed random walk, and (2) how to derive an invariant for the newly-defined *push operator*? Our approach to overcome these challenges is based on the following property of the v-absorbed random walk.

Lemma 3.10. *For any nodes $u, t \neq v$, let $\tau_v[u, t]$ be the expected number of visits on t of a v-absorbed random walk that starts from u. Then, we have $\tau_v[u, t] = \delta\{u = t\} + \sum_{w \in N(u), w \neq v} \frac{1}{d_u} \tau_v[w, t]$, where $\delta\{u = t\}$ is an indicator variable such that $\delta\{u = t\} = 1$ if $u = t$ and 0 otherwise.*

Proof. By definition, we can derive that $\tau_v[u, \cdot] = \sum_{k=0}^{\infty} e_u^T P_v^k$, where e_u is a unit vector in which u-th element is 1 and the other elements are 0. Then, we can reformulate the equation as: $\tau_v[u, \cdot] = e_u^T + \sum_{k=1}^{\infty} e_u^T P_v^k = e_u^T + \sum_{w \in N(u), w \neq v} \frac{1}{d_u} \sum_{k=0}^{\infty} e_w^T P_v^k$. Since $\sum_{k=0}^{\infty} e_w^T P_v^k = \tau_v[w, \cdot]$, we have $\tau_v[u, \cdot] = e_u^T + \sum_{w \in N(u), w \neq v} \frac{1}{d_u} \tau_v[w, \cdot]$. As a result, we have $\tau_v[u, t] = (e_u^T + \sum_{w \in N(u), w \neq v} \frac{1}{d_u} \tau_v[w, \cdot])_t = \delta\{u = t\} + \sum_{w \in N(u), w \neq v} \frac{1}{d_u} \tau_v[w, t]$, which completes the proof. □

Lemma 3.10 indicates that the expected number of visits on a node u by a v-absorbed random walk can be represented by the expected number of visits on its neighbors $w \in N(u)$. With this

Algorithm 1: The v-absorbed push procedure

```
Input: A graph $G$, a source node $s$, a landmark node $v$, a
       threshold $r_{\max}$
Output: $\hat{\tau}_v[s,u]$ and residual $r[u]$ for all $u \in V, u \neq v$
1 for each $u \in V, u \neq v$ do
2     $r[u] = 0$, $\hat{\tau}_v[s,u] = 0$;
3 $r[s] = 1$;
4 while $\exists u \in V$ such that $r[u] \geq d_u r_{\max}$ do
5     $\hat{\tau}_v[s,u]$ += $r(u)$;
6     for each $w \in N(u)$, $w \neq v$ do
7         $r[w]$ += $r[u]/d_u$;
8     $r[u] = 0$;
9 return $\hat{\tau}_v[s,u]$, $r[u]$ for all $u \in V, u \neq v$;
```

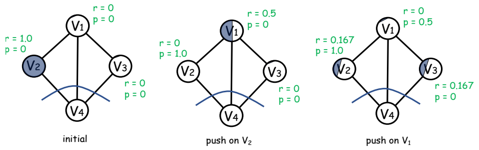


Figure 2: Illustration of the v-absorbed push with $r_{\max} = 0.2$ (v_4 is the absorbed node). The Push operations are conducted on v_2 and v_1. On each node u, p is an estimation of $\tau_{v_4}[v_2, u]$ ($u = v_1, v_2, v_3$). The exact value is 0.75, 1.25, 0.25.

property, we devise a novel push operator for the v-absorbed random walk. Similar to the PageRank push procedure, our push procedure also maintains two vectors q and r, where $q[u]$ denotes the reserve of u and $r[u]$ is the residual of u. The objective of the push procedure is to compute all $q[u]$ as an estimation of $\tau_v[s,u]$. The residual for each node $u \neq v$ and $u \neq s$ is initialized as 0, and $r[s]$ is initialized as 1. Then, the push procedure iteratively conducts push operators on the nodes, until all residuals are below a given threshold. We will show that during the push procedure, an invariant is maintained and the reserve $q[u]$ will approach to $\tau_v[s,u]$. Specifically, we can formally define the v-absorbed push operator as follows.

Definition 3.11. (v-absorbed push operator) A v-absorbed push operator on a node u includes three sequential steps: i) $q[u] \leftarrow q[u] + r[u]$; ii) $r[w] \leftarrow r[w] + \frac{r[u]}{d_u}$ for each node $w \in N(u)$ except v; and iii) $r[u] \leftarrow 0$.

By Definition 3.11, the v-absorbed push operator on a node u first adds the residual $r[u]$ to itself, and then uniformly propagates another copy of $r[u]$ to its neighbors except the absorbed node v. Clearly, if the absorbed node v is a neighbor of u, then v will absorb the value of $r[u]/d_u$ in v. Therefore, the total residual of all nodes in the graph decreases when the push procedure hits a neighbor of the absorbed node v, indicating that the push procedure converges (because all residuals will be absorbed by v). As an illustrative example shown in Fig. 2, when applying a push operation on v_2, 0.5 will be pushed to v_1 and another 0.5 will be absorbed by v_4 (the middle figure). After that, when applying a push on v_1, $0.5/3 = 0.167$ will be pushed to v_2 and v_3 respectively, and the remaining values 0.167 are absorbed by v_4.

The proposed push procedure for estimating $\tau_v[s,u]$ with $u \neq v$ is given in Algorithm 1. With Lemma 3.10, we can prove that the following invariant is maintained during the push procedure, which results in an efficient deterministic algorithm for approximating the resistance distance $r(s,t)$. (See Section 4.3).

Lemma 3.12. *(invariant by the v-absorbed push) For each node $t \in V$ and $t \neq v$, the reserve $q[t]$ and the residues $r[t]$ satisfy the following invariant during the v-absorbed push procedure:*

$$\tau_v[s,t] = q[t] + \sum_{w \neq v} r[w]\tau_v[w,t]. \tag{11}$$

Proof. The invariant can be proved by induction. Initially, the residue is $r[s] = 1$ and 0 otherwise. As a result, we have $\tau_v[s,t] = 0 + \tau_v[s,t]$, and thus the invariant holds at the initial stage. Assume that Eq. (11) holds before performing a push operator on u. Then, we show that it still holds after performing a push operator on u. By definition, for a v-absorbed push operator on u, the reserve $q[u]$ increase by $r[u]$. At the same time, the residue of all neighbors of u except v, i.e., $q[w]$ for each $w \in N(v)$ and $w \neq v$, increases by $\frac{r[u]}{d_u}$, and the residue $r[u]$ decreases to 0. Therefore, after performing a push operator on u, the right hand side of Eq. (11) is $q[t] + r[u]\delta\{u = t\} + \sum_{w \neq v} r[w]\tau_v[w,t] + \sum_{w \in N(u), w \neq v} \frac{r[w]}{d_u}\tau_v[w,t] - r[u]\tau_v[u,t]$. That is to say, the right hand side increases by $\Delta = r[u]I\{u = t\} - r[u]\tau_v[u,t] + \sum_{w \in N(u), w \neq v} \frac{r[w]}{d_u}\tau_v[w,t]$. Note that by Lemma 3.10, we can easily verify that $\Delta = 0$. As a result, the invariant holds during the push procedure, which completes the proof. □

Note that for sampling-based algorithms, the solutions are unbiased estimators, thus both over-estimation and under-estimation are possible. Compared to the sampling-based algorithms, v-absorbed push is a deterministic algorithm where only under-estimate is possible. Based on the established invariant, we can easily derive an additive error guarantee of Algorithm 1 in term of $r_{\max}$.

Lemma 3.13. *The additive error of the estimation $\hat{\tau}_v[s,u]$ in Algorithm 1 can be bounded by $\sum_{w \neq v} \tau_v[w,u] d_w r_{\max}$.*

Proof. By Eq. (11), $\tau_v[s,u] = q[u] + \sum_{w \neq v} r[w]\tau_v[w,u]$. In Algorithm 1, $r[w]$ is bounded by $d_w r_{\max}$, so the additive error can be bounded by $\sum_{w \neq v} \tau_v[w,u] d_w r_{\max}$. □

Note that the result in Lemma 3.13 is the worst-case bound. However, in real-world graphs, the proposed v-absorbed push algorithm is often very accurate as confirmed in our experiments. Additionally, it is worth mentioning that once we obtain a deterministic estimation of $\tau_v[s,u]$ for all $u \in V$, we can easily derive a deterministic estimation for the resistance distance $r(s,t)$ based on Eq. (9).

The time complexity of Algorithm 1 is dependent on the absorbed node v, i.e., the landmark node. In general, if v is easier to hit, the residuals will decrease faster, and the algorithm will also terminate faster. Let $h(s,v)$ be the hitting time from s to v by the v-absorbed random walk. Here the hitting time denotes the expected number of nodes that are visited by the v-absorbed random walk that starts from s before hitting t. Then, we have the following result.

Lemma 3.14. *$h(s,v) = \sum_{u \neq v} \tau_v[s,u]$.*

Proof. Note that $\sum_{u \neq v} \tau_v[s,u]$ is the sum of the expected number of visits on all nodes except v by the v-absorbed random walk that starts from s before hitting v. Since the v-absorbed random walk visits only one node at each step, such a summation is exactly equal to the hitting time $h(s,v)$ (by definition). □

Based on Lemma 3.14, we analyze the time complexity of Algorithm 1 in Theorem 3.15.

Theorem 3.15. *The time complexity of Algorithm 1 is $O(\frac{h(s,v)}{r_{\max}})$.*

Proof. In Algorithm 1, the estimation $\hat{\tau}_v[s,u]$ is initialized as 0, and it is updated in Line 5. Note that $\hat{\tau}_v[s,u]$ never decreases. Each time $\hat{\tau}_v[s,u]$ is updated as in Line 5, the total amount of the vector increases by at least $d_u r_{\max}$. Each update procedure costs

Algorithm 2: AbWalk

```
   Input: A graph $G$, a source node $s$, a target node $t$ ($s \neq t$), a
          landmark $v$, sample size $T$
   Output: $\hat{r}(s,t)$
1  $\hat{r}(s,t) \leftarrow 0$;
2  if $t = v$ (resp., $s = v$) then
3      for $i = 1 : T$ do
4          Simulate a $v$-absorbed random walk from $s$ (resp., $t$);
             let $\tau_s$ (resp., $\tau_t$) be the number of visits on node $s$
             (resp., $t$);
5          $\hat{r}(s,t) \leftarrow \hat{r}(s,t) + \frac{\tau_s}{d_s T}$ (resp., $\hat{r}(s,t) \leftarrow \hat{r}(s,t) + \frac{\tau_t}{d_t T}$);
6  else
7      for $i = 1 : T$ do
8          Simulate a $v$-absorbed random walk from $s$; let $\tau_{ss}$
             be the number of visits on node $s$, $\tau_{st}$ be the
             number of visits on node $t$;
9          Simulate a $v$-absorbed random walk from $t$; let $\tau_{tt}$ be
             the number of visits on node $t$, $\tau_{ts}$ be the number
             of visits on node $s$;
10         $\hat{r}(s,t) \leftarrow \hat{r}(s,t) + \frac{\tau_{ss}}{d_s T} + \frac{\tau_{tt}}{d_t T} - \frac{\tau_{st}}{d_t T} - \frac{\tau_{ts}}{d_s T}$;
11 return $\hat{r}_{s,t}$
```

$O(d_u)$, so an *atom update* results in at least $r_{\max}$ increment. When the algorithm terminates, the total amount of the estimation is upper bounded by $\sum_{u \neq v} \tau_v[s,u] = h(s,v)$. Therefore, there are at most $O(\frac{h(s,v)}{r_{\max}})$ atom operations. The time complexity is then $O(\frac{h(s,v)}{r_{\max}})$. □

4 SINGLE-PAIR QUERY COMPUTATION

In this section, we develop several novel algorithms for answering the single-pair resistance distance query, based on our theoretical results established in Section 3. We first propose an algorithm AbWalk based on the v-absorbed random walk interpretation of resistance distance in Section 4.1. Then, we develop an algorithm LocalTree based on the spanning forest explanation. After that, we present a deterministic algorithm Push based on the v-absorbed push procedure. Finally, we propose a bidirectional algorithm Bipush which combines Push and AbWalk to improve the accuracy of the algorithm.

4.1 A v-absorbed random walk based algorithm

By Lemma 3.5, we are able to estimate the resistance distance by simulating v-absorbed random walks. We refer to such an algorithm as AbWalk. The pseudo-code of AbWalk is outlined in Algorithm 2.

First, we choose a node v as a landmark node (e.g., the maximum degree node). Basically, according to Eq. (5) and Eq. (6), there are two cases needed to consider. If either s or t equals the landmark node v, we only need to estimate a term $(L_v^{-1})_{tt}$ or $(L_v^{-1})_{ss}$. This is exactly the degree-normalized expected number of visits on a node v for a v-absorbed random walk. Therefore, we sample T such random walks, and take the average degree-normalized visits as the result (Lines 2-5). If $s, t \neq v$, $r(s,t)$ can be represented as four terms of the elements of L_v^{-1} which requires two v-absorbed random walks independently sampled from s and t. Again, the combination of four degree-normalized number of average visits is used as the estimator (Lines 7-10).

Note that the estimator $\hat{r}_{s,t}$ in Algorithm 2 is an unbiased estimator according to Lemma 3.5. Since the random variables τ_{st} in Algorithm 2 are unbounded, the sample size T is very hard to determined in theory. However, as we observed in the experiments, AbWalk performs very well on large real-life datasets using only 10^4 samples.

Let $h(s,v)$ be the hitting time from s to v by the v-absorbed random walk. The time complexity of Algorithm 2 relies on the hitting time which can be easily derived by definition.

Theorem 4.1. *The time complexity of* AbWalk *is* $O(T \times h(t,v))$ *if* $s = v$, $O(T \times h(s,v))$ *if* $t = v$, *and* $O(T \times (h(s,v) + h(t,v)))$ *if* $s, t \neq v$.

Compared to the state-of-the-art commute-time based algorithm [44], simulating two v-absorbed random walks separately from s and t is intuitively faster than simulating a round-trip random walk from s to t, and then back to s, especially when s and t are hard to hit from each other. This is because we can choose a landmark node v as an easy-to-hit node. Generally, such a node exists in real-life networks, e.g. the largest-degree nodes or the hub nodes. Although it may be difficult for a traditional random walk that starts from s to hit t, it is often easy to hit the largest-degree node from either s or t, as confirmed in our experiments (see Section 6).

4.2 A local spanning tree sampling algorithm

In this subsection, we develop an interesting spanning tree sampling algorithm for estimating resistance distance $r(s,t)$ based on the theoretical results presented in Section 3.3. The novelty of our algorithm, denoted by LocalTree, is twofold. First, LocalTree is the first spanning tree sampling algorithm that can estimate $r(s,t)$ for any pair of nodes s and t, while existing spanning tree sampling algorithm can only estimate the resistance distance $r(s,t)$ when (s,t) is an edge in G [22]. Second, existing spanning tree sampling algorithm is a global algorithm [22], as it needs to traverse the whole graph to sample spanning trees. Interestingly, we show that our algorithm is a local algorithm which only visits a small portion of the graph. In particular, LocalTree only needs to perform the first two steps of the spanning tree sampling algorithm (the classic Wilson algorithm [61]) which is sufficient to determine the unique path between s and t in the corresponding sampled spanning tree. Such a local algorithm can significantly reduce the cost for drawing a sample.

Loop-erased random walk. The loop-erased random walk is a type of random walk where we obtain the random walk trajectory by erasing all its loops. The Wilson algorithm [61] is a well-known algorithm for uniformly sampling spanning trees based on loop-erased random walks. It starts by fixing an arbitrary node ordering and initializing the spanning tree Γ with a root node v (initially, $\Gamma = \{v\}$). Then, it runs loop-erased random walks following the fixed node ordering, until the walk hits Γ. Once the random walk hits any node in Γ, the loop-erased random walk trajectory is added into Γ. When all nodes are covered, a spanning tree Γ is sampled uniformly [61]. Obviously, such an algorithm is a global algorithm which is often costly to sample a spanning tree on large graphs.

To make the sampling procedure locally, the key idea of LocalTree is that the node ordering in the Wilson algorithm is arbitrary which means that we can choose s and t as the first two nodes. As long as the two nodes are added into the sampled spanning tree, the unique path between them in the corresponding spanning tree can be determined. This is because when s and t are added into Γ, then there must exist a path $v \sim t$ and a path $v \sim s$ in Γ, thus a $s \sim t$ path can be obtained. This result suggests that we can sample a spanning tree *locally* by only running the first two steps of the Wilson algorithm.

The pseudo-code of LocalTree is shown in Algorithm 3. According to Lemma 3.9, to estimate $r(s,t)$, we first fix a path $\mathcal{P}_{st}$ between s

Algorithm 3: LocalTree

```
Input: A graph $G$, a source node $s$, a target node $t$, a
       landmark node $v$, a pre-computed BFS tree $\Gamma_{BFS}$,
       sample size $T$
Output: $\hat{r}(s,t)$
1  Let $\mathcal{P}_{st}$ be the directed path from $s$ to $t$ in $\Gamma_{BFS}$;
2  for $i = 1 : T$ do
3      Simulate a random walk from $s$, until it hits $v$, let $\gamma_{sv}$ be
         the trajectory of the random walk after erasing loops;
4      Simulate a random walk from $t$, until it hits $\gamma_{sv}$, let $\gamma_{tv}$
         be the trajectory of the random walk after erasing
         loops;
5      $\gamma_{st} = \{(i,j)|(i,j) \in \gamma_{sv}\} \cup \{(j,i)|(i,j) \in \gamma_{tv}\}$;
6      Let $\tilde{\mathcal{P}}_{s,t}$ be the directed path from $s$ to $t$ in $\gamma_{st}$;
7      for each edge $(i,j) \in \mathcal{P}_{st}$ do
8          if $(i,j) \in \tilde{\mathcal{P}}_{s,t}$ then $\hat{r}(s,t) \leftarrow \hat{r}(s,t) + \frac{1}{T}$;
9          if $(j,i) \in \tilde{\mathcal{P}}_{s,t}$ then $\hat{r}(s,t) \leftarrow \hat{r}(s,t) - \frac{1}{T}$;
10 return $\hat{r}(s,t)$;
```

and t, and then compare the unique path $\tilde{\mathcal{P}}_{s,t}$ between s and t in the sampled spanning tree with $\mathcal{P}_{st}$. To this end, we can pre-compute a BFS (Breadth-First Search) tree with a root node v (Line 1). We can obtain a path from s to t by traversing the BFS tree. We first sample a random walk from s until it hits v, and then we erase all its loops and record the trajectory (Lines 3). Then, we sample a random walk from t until it hits the trajectory (Line 4). Note that the unique path $\tilde{\mathcal{P}}_{s,t}$ in the sampled spanning tree can be determined after these two steps. After that, we update the estimation by traversing the BFS tree (Lines 6-9). To illustrate implementation details, the pseudo-code of LocalTree (Implementation Details) is also shown in Algorithm 4. According to Lemma 2.1 and Lemma 3.9, to estimate $r(s,t)$, we first fix a path $\mathcal{P}_{st}$ between s and t, and then compare the unique path $\tilde{\mathcal{P}}_{s,t}$ between s and t in the sampled spanning tree with $\mathcal{P}_{st}$. To this end, we can pre-compute a BFS (Breadth-First Search) tree with a root node v. The BFS tree can be recorded by a vector bfs where $bfs[u]$ represents the parent node of u in the BFS tree. We can obtain a path from s to t by traversing the BFS tree. We use $InTree$ to record whether a node is added into a spanning tree or not, $Next$ to record the next node in the random walk trajectory. Initially, the landmark node v is set as the root node (Line 3). We first sample a random walk from s until it hits v (Lines 4-6), and then we erase all its loops by retracing the trajectory (Lines 7-8). The same operations are performed to sample a random walk from t until it hits the trajectory (Lines 9-13). Note that the unique path $\tilde{\mathcal{P}}_{s,t}$ in the sampled spanning tree can be determined after these two steps. We use a vector $reverse$ to record the edge direction of the path (Line 13). After that, we update the estimation by traversing the BFS tree from s (Lines 14-19) and t (Lines 20-25). The same as the Wilson algorithm [45], it can be guaranteed that the spanning tree is sampled uniformly, although we only obtain the path between s and t. The correctness of Algorithm 3 is guaranteed by Lemma 3.9. The time complexity of LocalTree is shown in the following theorem.

Theorem 4.2. *The time complexity of Algorithm 3 is bounded by $O(T(h(s,v)+h(t,v)))$, where $h(s,v)$ is the hitting time of the random walk from s to hit v.*

Proof. To draw a sample, Algorithm 3 first takes $O(h(s,v))$ to run a loop-erased random walk starting from s to hit v. Then, the algorithm performs a loop-erased random walk starting from v to hit the loop-erased random walk trajectory from s to v, which takes at most $O(h(t,v))$ time. As a result, the total time to draw T samples is bounded by $O(T(h(s,v)+h(t,v)))$. □

Algorithm 4: LocalTree (Implementation Details)

```
Input: A graph $G$, a source node $s$, a target node $t$, a
       landmark node $v$, a pre-computed BFS tree $bfs$,
       sample size $T$
Output: $\hat{r}(s,t)$
1  for $i = 1 : T$ do
2      $InTree[u] \leftarrow false$, $Next[u] \leftarrow -1$, $reverse[u] \leftarrow 1$
         for all $u \in V$;
3      $InTree[v] \leftarrow true$;
4      if $!InTree[s]$ then
5          $u \leftarrow s$; while $!InTree[u]$ do
6              $Next[u] \leftarrow RandomNeighbor(u)$,
                 $u \leftarrow Next[u]$;
7          $u \leftarrow s$; while $!InTree[u]$ do
8              $InTree[u] \leftarrow true$, $u \leftarrow Next[u]$;
9      if $!InTree[t]$ then
10         $u \leftarrow t$; while $!InTree[u]$ do
11             $Next[u] \leftarrow RandomNeighbor(u)$,
                 $u \leftarrow Next[u]$;
12         $u \leftarrow t$; while $!InTree[u]$ do
13             $InTree[u] \leftarrow true$, $reverse[u] = -1$,
                 $u \leftarrow Next[u]$;
14     $u \leftarrow s$; while $u! = v$ do
15         if $Next[u] == bfs[u]$ and $InTree[u]$ and
             $InTree[Next[u]]$ then
16             $\hat{r}(s,t) \leftarrow \hat{r}(s,t) + \frac{reverse[u]}{T}$;
17         else if $u == bfs[Next[u]]$ and $InTree[u]$ and
             $InTree[Next[u]]$ then
18             $\hat{r}(s,t) \leftarrow \hat{r}(s,t) - \frac{reverse[u]}{T}$;
19         $u \leftarrow bfs[u]$;
20     $u \leftarrow t$; while $u! = v$ do
21         if $Next[u] == bfs[u]$ and $InTree[u]$ and
             $InTree[Next[u]]$ then
22             $\hat{r}(s,t) \leftarrow \hat{r}(s,t) + \frac{reverse[u]}{T}$;
23         else if $u == bfs[Next[u]]$ and $InTree[u]$ and
             $InTree[Next[u]]$ then
24             $\hat{r}(s,t) \leftarrow \hat{r}(s,t) - \frac{reverse[u]}{T}$;
25         $u \leftarrow bfs[u]$;
26 return $\hat{r}(s,t)$;
```

Note that LocalTree is similar to AbWalk in that both of them simulate two random walks from s and t respectively. However, unlike AbWalk, the random variables used in LocalTree are bounded, thus we are able to derive a sample size bound for LocalTree. Specifically, since the $s \sim t$ path in LocalTree is fixed, the related random variables can be bounded by the length of path. As we use the BFS tree to obtain the path, the length of the path between any two nodes in the BFS tree is bounded by the diameter of the graph G, denoted by Δ_G.

We utilize the following Hoeffding's inequality to show that LocalTree achieves an absolute error with a probability at least $1 - p_f$, where p_f is a small failure probability.

Lemma 4.3. *(Hoeffding's inequality) Let $X_1, \cdots, X_T$ be independent random variables in $[a, b]$, where $-\infty < a \le b < \infty$, for any $0 < \epsilon < 1$, we have*

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

Theorem 4.4. *If the sample size $T \ge \frac{2\Delta_G^2 log(\frac{2}{p_f})}{\epsilon^2}$, Algorithm 3 outputs $\hat{r}(s,t)$ that satisfies $|\hat{r}(s,t) - r(s,t)| \le \epsilon$ with a probability at least $1 - p_f$.*

Proof. Let $\hat{r} = X_+ - X_-$ be the random variable defined in Lemma 3.9. $|\hat{r}|$ is bounded by the length of the path $\mathcal{P}_{s,t}$, which is further bounded by Δ_G. Thus, we have $-\Delta_G \le \hat{r} \le \Delta_G$. According to Lemma 3.9, we have $E[\hat{r}] = r(s,t)$. By applying the Hoeffding's inequality, if $T \ge \frac{4\Delta_G^2 log(\frac{1}{p_f})}{\epsilon^2}$, we can obtain that $Pr(|\frac{\sum_{i=1}^{T}\hat{r}_i}{T} - r(s,t)| \ge \epsilon) \le 2exp(-\frac{2\epsilon^2 T}{4\Delta_G^2}) \le 2exp(-\frac{2\epsilon^2 \frac{2\Delta_G^2 log(\frac{2}{p_f})}{\epsilon^2}}{4\Delta_G^2}) \le p_f$, thus the theorem is established. □

4.3 A v-absorbed push based algorithm

Based on the v-absorbed push procedure proposed in Section 3.4, we can easily develop a deterministic push algorithm to approximate the single-pair resistance distance $r(s,t)$. Recall that $r(s,t) = \frac{\tau_v[s,s]}{d_s} + \frac{\tau_v[t,t]}{d_t} - \frac{\tau_v[s,t]}{d_t} - \frac{\tau_v[t,s]}{d_s}$ (Eq. (9)). For a fixed absorbed node v (i.e., the landmark node), if $s = v$, we have $\tau_v[s,s] = \tau_v[v,v] = 1$, $\tau_v[s,t] = \tau_v[v,t] = 0$, and $\tau_v[t,s] = \tau_v[t,v] = 1$; and thus we have $r(s,t) = r(v,t) = \frac{\tau_v[t,t]}{d_t}$. Similarly, if $t = v$, we have $r(s,t) = r(s,v) = \frac{\tau_v[s,s]}{d_s}$. As a consequence, if either s or t equals v, we only need to invoke the v-absorbed push procedure (Algorithm 1) once to estimate $r(s,t)$. However, if both s and t do not equal to v, it is easy to show that we need to invoke Algorithm 1 twice to estimate $r(s,t)$ (one from s and the other one from t). The pseudo-code of our push algorithm is shown in Algorithm 5. We can easily show that the worst-case time complexity of Algorithm 5 is $O(\frac{h(s,v)+h(t,v)}{r_{\max}})$. Also, the additive error bound can be easily derived based on Lemma 3.13 which is at most twice of the error bounds in Lemma 3.13. In our experiments, we will show that Push is extremely fast and also achieves a low error in practice.

4.4 A bidirectional approach

Motivated by the bidirectional algorithm for personalized PageRank computation [34, 60], we show that the resistance distance can also be estimated in a similar way by integrating both the proposed Push algorithm and the v-absorbed random walk algorithm. For convenience, we refer to such a bidirectional algorithm as Bipush. The key idea of Bipush is that it first applies Push with a source node s to obtain an estimation of $r(s,t)$ with an additive error guarantee, and then uses the v-absorbed random walk algorithm to estimate the additive error. The pseudo-code of Bipush is shown in Algorithm 6.

When $t = v$, $r(s,t) = \tilde{\tau}_t[s,s] = \frac{\tau_t[s,s]}{d_s}$. For an estimation of $\tau_t[s,s]$, it firstly invokes a t-absorbed push algorithm with threshold $r_{\max}$ from s (Line 2), and $\frac{q[s]}{d_s}$ is the current estimation of $r(s,t)$ (Line 4). After that, the invariant of Eq. (11) is maintained. We have $\tau_t[s,s] = q[s] + \sum_{w \ne v} r[w]\tau_t[w,s]$. It remains to estimate the error term $\sum_{w \ne v} r[w]\tau_t[w,s] = \sum_{w \ne v} \frac{d_s}{d_w} r[w]\tau_t[s,w]$. Here the equality is due to that $(L_t)^{-1}$ is a symmetric matrix, thus we have $(L_v^{-1})_{sw} = \tilde{\tau}_t[w,s] = \frac{\tau_t[w,s]}{d_s} = \frac{\tau_t[s,w]}{d_w} = \tilde{\tau}_t[s,w] = (L_v^{-1})_{ws}$. Dividing all terms by d_s, we have $r(s,t) = \tilde{\tau}_t[s,s] = \frac{\tau_t[s,s]}{d_s} = \frac{q[s]}{d_s} + \sum_{w \ne v} \frac{r[w]}{d_w}\tau_t[s,w]$. Then, we sample T t-absorbed random walks from s (Lines 5-9). Note that the number of visits on a node $w \ne t$ by a t-absorbed random walk is an unbiased estimator of $\tau_t[s,w]$ for $w \in V$ and $w \ne t$. For each node visited by the walk, we can update the estimation $\hat{r}(s,t)$ by adding $\frac{r[u]}{d_u}$ in Line 6. This leads to a more precise unbiased estimator of $r(s,t)$, compared to the AbWalk algorithm. When $s = v$ or $s, t \ne v$, similar approach can be applied (see Lines 8-18).

It is easy to show that worst-case time complexity of Bipush is bounded by $O((T + 1/r_{\max})(h(s,v) + h(t,v)))$. In the experiments, we will show that Bipush is fast and extremely accurate to answer the single-pair resistance distance query on real-life graphs.

Algorithm 5: Push

```
Input: A graph $G$, a source node $s$, a target node $t$, a
       landmark node $v$, a threshold $r_{\max}$
Output: $\hat{r}(s,t)$
1 if $t = v$ (resp., $s = v$) then
2     $\hat{\tau}_v[t,u], r_t[u] \leftarrow v$-absorbed-push$(G, t, v, r_{\max})$
3     (resp., $\hat{\tau}_v[s,u], r_s[u] \leftarrow v$-absorbed-push$(G, s, v, r_{\max})$);
4     $\hat{r}(s,t) \leftarrow \frac{\hat{\tau}_v[t,t]}{d_t}$ (resp., $\hat{r}(s,t) \leftarrow \frac{\hat{\tau}_v[s,s]}{d_s}$);
5 else
6     $\hat{\tau}_v[s,u], r_s[u] \leftarrow v$-absorbed-push$(G, s, v, r_{\max})$;
7     $\hat{\tau}_v[t,u], r_t[u] \leftarrow v$-absorbed-push$(G, t, v, r_{\max})$;
8     $\hat{r}(s,t) \leftarrow \frac{\hat{\tau}_v[s,s]}{d_s} - \frac{\hat{\tau}_v[s,t]}{d_t} - \frac{\hat{\tau}_v[t,s]}{d_s} + \frac{\hat{\tau}_v[t,t]}{d_t}$;
9 return $\hat{r}(s,t)$;
```

4.5 Heuristic choices of the landmark node v

Recall that all the proposed algorithms need to select a landmark node v. Intuitively, a node that is easy-to-hit by a random walk is a good landmark node. This is because the time complexity of all the proposed algorithms are closely related to the hitting time $h(s,v)$ and $h(t,v)$. To achieve a good performance, we suggest three heuristic choices of the landmark node. Specifically, we select the landmark node as (i) the highest-degree node, or (ii) the highest PageRank node, or (iii) the node with the maximum core number (a k-core is the maximal subgraph where each node has a degree no less than k; and the core number of a node u is the largest k such that there is a k-core containing u). This is because all these three heuristic strategies select the high-centrality node as the landmark which is intuitively easy to hit by a random walk. Moreover, all these strategies can be efficiently implemented. Note that there may not exist one landmark node that optimizes all queries, but a careful choice of the landmark node can lead to a very well average performance. In our experiments, we will study the effect of those landmark selection strategies.

5 SINGLE-SOURCE QUERY COMPUTATION

In this section, we develop several novel and efficient algorithms to process the single-source resistance distance query. Note that all the algorithms proposed in Section 4 can be easily extended

Algorithm 6: Bipush

```
Input: A graph $G$, a source node $s$, a target node $t$, a
       landmark node $v$, a threshold $r_{\max}$, sample size $T$
Output: $\hat{r}(s,t)$
if $t = v$ (resp., $s = v$) then
    $\hat{\tau}_v[s,u], r_s[u] \leftarrow t$-absorbed-push$(G, s, v, r_{\max})$;
    (resp., $\hat{\tau}_v[t,u], r_t[u] \leftarrow s$-absorbed-push$(G, t, v, r_{\max})$);
    $\hat{r}(s,t) \leftarrow \frac{\hat{\tau}_v[s,s]}{d_s}$ (resp., $\hat{r}(s,t) \leftarrow \frac{\hat{\tau}_v[t,t]}{d_t}$);
    for $i = 1 : T$ do
        Run a $t$-absorbed random walk from $s$;
        $\hat{r}(s,t) \leftarrow \hat{r}(s,t) + \frac{r_s[u]}{d_u T}$ for each node $u$ that the
          random walk visits;
        (resp., run a $s$-absorbed random walk from $t$;
        $\hat{r}(s,t) \leftarrow \hat{r}(s,t) + \frac{r_t[u]}{d_u T}$ for each node $u$ that the
          random walk visits;)
else
    $\hat{\tau}_v[s,u], r_s[u] \leftarrow v$-absorbed-push$(G, s, v, r_{\max})$;
    $\hat{\tau}_v[t,u], r_t[u] \leftarrow v$-absorbed-push$(G, t, v, r_{\max})$;
    $\hat{r}(s,t) \leftarrow \frac{\hat{\tau}_v[s,s]}{d_s} - \frac{\hat{\tau}_v[s,t]}{d_t} - \frac{\hat{\tau}_v[t,s]}{d_s} + \frac{\hat{\tau}_v[t,t]}{d_t}$;
    for $i = 1 : T$ do
        Run a $v$-absorbed random walk from $s$;
        $\hat{r}(s,t) \leftarrow \hat{r}(s,t) + \frac{r_s[u]}{d_u T} - \frac{r_t[u]}{d_u T}$ for each node $u$ that
          the random walk visits;
        Run a $v$-absorbed random walk from $t$;
        $\hat{r}(s,t) \leftarrow \hat{r}(s,t) + \frac{r_t[u]}{d_u T} - \frac{r_s[u]}{d_u T}$ for each node $u$ that
          the random walk visits;
return $\hat{r}(s,t)$;
```

to handle the single-source query problem by processing $n-1$ queries $r(s,u)$ for all $u \in V$. However, the time complexity of such straightforward baselines is around $O(n)$ times of the time costs taken by processing a single-pair query, which is clearly costly for large graphs. To tackle this problem, we first propose a new loop-erased random walk sampling algorithm which is shown to be more efficient than the baselines. Then, to further improve the efficiency, we develop two novel index-based algorithms which can answer a single-source query in almost the same time as answering a single-pair query. Below, we first briefly analyze the challenges of the single-source query problem, followed by the loop-erased random walk algorithm and the index-based algorithms.

5.1 Challenges of single-source query

According to Eq. (5) and Eq. (6), we can see that to compute $r(s,u)$ for all $u \in V$, it is sufficient to (i) calculate $(L_v^{-1})_{su}$ for all $u \in V$ (because L_v^{-1} is a symmetric matrix), and (ii) compute all diagonal elements of the matrix L_v^{-1} (i.e., $(L_v^{-1})_{uu}$ for all $u \in V$). Note that we can run a v-absorbed push procedure (Algorithm 1) with a source node s to compute $(L_v^{-1})_{su}$ for all $u \in V$, which corresponds to a row of the matrix L_v^{-1}. The most challenging part of the single-source resistance distance query problem is how to efficiently compute all the diagonal elements of L_v^{-1}. To our knowledge, there is no efficient algorithm that can compute the diagonal elements of L_v^{-1}. A straightforward algorithm is to compute the matrix inverse of L_v and then get the diagonal elements, which is very costly and clearly does not work for large graphs.

Remark. It is worth remarking that a single-source personalized PageRank query only requires computing a row of the personalized PageRank matrix (no need to compute the diagonal elements), thus the problem can be easily solved by using a push algorithm with a source node s. Therefore, existing techniques for processing single-source personalized PageRank query cannot be extended to solve our problem. We also note that a recent work on approximating the diagonal items of the pseudo-inverse of Laplacian $L^\dagger$ [6] can be modified to compute the single-source resistance distance query. However, the method proposed in [6] is mainly tailored for estimating the diagonal elements of $L^\dagger$ (not L_v^{-1}). Moreover, the technique used in [6] requires to match a sampled spanning tree with a pre-computed tree, which is also expensive for large graphs (as evidenced in our experiments).

5.2 The loop-erased random walk based algorithm

In this subsection, we develop a novel algorithm based on the loop-erased random walk sampling [61] to answer the single-source query. We show that sampling a spanning tree by the loop-erased random walk can obtain an unbiased estimator for $r(s,u)$ for all $u \in V$ $(u \neq v)$. As a consequence, we can sample T spanning trees to obtain a good approximation for the single-source resistance distance query problem. We will show that such a new algorithm is much faster than the approach based on processing $O(n)$ single-pair queries.

Our technique is based on an interesting connection between the v-absorbed random walk and the loop-erased random walk. As introduced in Section 4.2, the loop-erased random walk algorithm (i.e., the Wilson algorithm) first sets a node v as root. Then, the algorithm fixes an arbitrary node ordering $u_1, \cdots, u_{n-1}$, and follows this order to simulate loop-erased random walk until hitting the former trajectories. The algorithm terminates when all nodes are covered. Through a complete execution of the Wilson algorithm, each node except v can be visited many times. Surprisingly, we show that the expected number of visits on each node u equals $\tau_v[u,u]$, which is the expected number of visits on u for a v-absorbed random walk starting from u.

Lemma 5.1. *Suppose that we simulate a loop-erased random walk with a root v. Let X_u be the random variable of the number of visits on a node u. Then, we have*

$$E[X_u] = \tau_v[u,u] = (I - P_v)^{-1}_{uu}, \text{ for all } u \in V, u \neq v. \qquad (12)$$

Proof. Note that in the first step, there is only one absorbed node v, thus the loop-erased random walk starting from u_1 visits u_1 with $\tau_v[u_1,u_1]$ times in expectation. By Lemma 3.3, the expected number of visits on u itself in a v-absorbed random walk starting from u is $(I - P_v)^{-1}_{uu}$, thus Eq. (12) holds for the first node u_1 in the fixed ordering $u_1, \cdots, u_{n-1}$ in the Wilson algorithm. A key property of the Wilson algorithm is that the resulting random walk trajectory distribution is independent of the node ordering [61]. Therefore, every node can be served as the first node, and the resulting distribution remains the same. As a result, the expected number of visits on a node u equals $\tau_v[u,u]$ for all $u \in V$ with $u \neq v$, thus the lemma is established. □

According to Eq. (6) and Lemma 3.4, we have $r(v,u) = \tilde{\tau}_v[u,u] = \frac{\tau_v[u,u]}{d_u}$ for all $u \in V$ with $u \neq v$. Thus, $\frac{X_u}{d_u}$ is an unbiased estimator of $r(v,u)$. As a result, we can simulate loop-erased random walks to estimate all $r(v,u)$. The algorithm is outlined in Algorithm 7 which is a slightly modification of the classic Wilson algorithm. Compared to the Wilson algorithm, the main difference is that we additionally

Algorithm 7: LEwalk

```
Input: A graph $G$, a source node $s$, sample size $T$
Output: $\hat{r}(s,u)$ for all $u \in V, u \neq s$
1  $\hat{r}(s,u) \leftarrow 0$ for all $u \in V$;
2  Fix an arbitrary ordering $(v_1, \cdots, v_{n-1})$ of $V \setminus \{s\}$;
3  for $i = 1 : T$ do
4      $InTree[u] \leftarrow false, Next[u] \leftarrow -1$ for $u \in V$;
5      $InTree[s] \leftarrow true$;
6      for $j = 1 : n-1$ do
7          $u \leftarrow v_j, \hat{r}(s,u) \leftarrow \hat{r}(s,u) + \frac{1}{d_u T}$;
8          while $!InTree[u]$ do
9              $Next[u] \leftarrow RandomNeighbor(u)$;
10             $u \leftarrow Next[u], \hat{r}(s,u) \leftarrow \hat{r}(s,u) + \frac{1}{d_u T}$;
11         $\hat{r}(s,u) \leftarrow \hat{r}(s,u) - \frac{1}{d_u T}$;
12         $u \leftarrow v_i$;
13         while $!InTree[u]$ do
14             $InTree[u] \leftarrow true, u \leftarrow Next[u]$;
15 return $\hat{r}(s,u)$ for all $u \in V, u \neq s$;
```

record the number of visits on each node and use it to construct an unbiased estimator (see Lines 7, 10, and 11).

The time complexity of Algorithm 7 is closely related to the root node v (i.e., the landmark node) as shown in the following theorem.

Theorem 5.2. *The time complexity of Algorithm 7 is $O(T \times Tr((I - P_v)^{-1}))$, where $Tr((I - P_v)^{-1}))$ denotes the trace of the matrix $(I - P_v)^{-1})$ and T is the sample size.*

Proof. By Lemma 5.1, the expected number of visits on a node u by the loop-erased random walk equals $\tau_v[u,u]$. Clearly, the time complexity of Algorithm 7 is dominated by the total number of visits of all nodes by the loop-erased random walk. Therefore, the time complexity for sampling T samples is $O(T \times \sum_{u \in V}(I - P_v)^{-1}_{uu}) = O(T \times Tr((I - P_v)^{-1}))$. □

Note that the time complexity of Algorithm 7 is lower than that of the baseline algorithm based on processing $O(n)$ single-pair queries. This is because answering a single-pair query ($r(s,u)$) takes at least $O(T \times h(s,u)) = O(T \times \sum_{u \neq v} \tau_v[s,u]) = O(T \times \sum_{u \neq v}[I - P_v]^{-1}_{su})$ time by the algorithms proposed in Section 4. As a result, for a single-pair query, such an algorithm takes $O(T \times \sum_{s \neq v} \sum_{u \neq v}[I - P_v]^{-1}_{su})$, which is clearly much higher than $O(T \times Tr((I - P_v)^{-1}))$.

5.3 The index-based algorithms

Although Algorithm 7 is much faster than the baseline algorithm, sampling a spanning tree is a *global* procedure which may take a long time on large graphs. Hence, a natural question is that can we have a *local* algorithm (like the algorithms for single-pair query) to handle the single-source query? We answer this question affirmatively by developing two index-based algorithms.

Index construction. The key observation is that if can pre-computed the diagonal elements of the matrix L_v^{-1}, then the single-source query problem can be solved efficiently, as we analyzed in Section 5.1. By Eq. (6), we can see that the diagonal element $(L_v^{-1})_{ss} = r(v,s)$ is exactly the resistance distance between s and the landmark node v. As a result, we can first pre-compute all the resistance distances from the landmark node v to all the other nodes (just computing one single-source query from the landmark node v), and then use these pre-computed distance as an index $R[u]$ for $u \in V$ and $u \neq v$. With such a resistance distance index, it is sufficient to answer any single-source query from a node $s \neq v$ by computing a row of (L_v^{-1}). Note that we can use Algorithm 7 to build the index array R which takes $O(T \times Tr((I - P_v)^{-1}))$ time. Clearly, the space overhead of the index array R is $O(n)$.

Algorithm 8: AbWalk*

```
Input: A graph $G$, a source node $s$, a landmark node $v$,
       sample size $T$, a resistance distance index array $R[u]$
       for $u \in V, u \neq v$
Output: $\hat{r}(s,u)$ for all $u \in V$ and $u \neq s$
1  $\hat{\tau}_v[s,u] \leftarrow 0$ for all $u \in V$ and $u \neq v$;
2  for $i = 1 : T$ do
3      Run a $v$-absorbed random walk from $s$, for each step the
         random walk passes $u$, $\hat{\tau}_v[s,u] \leftarrow \hat{\tau}_v[s,u] + \frac{1}{T}$;
4  for $i = 1 : n$ do
5      $\hat{r}(s,u) \leftarrow R[s] + R[u] - 2\frac{\hat{\tau}_v[s,u]}{d_u}$;
6  return $\hat{r}(s,u)$ for all $u \in V$ and $u \neq s$;
```

Algorithm 9: Push*

```
Input: A graph $G$, a source node $s$, a landmark node $v$, a
       threshold $r_{max}$, a resistance distance index array
       $R[u]$ for $u \in V, u \neq v$
Output: $\hat{r}(s,u)$ for all $u \in V$ and $u \neq s$
1  $\hat{\tau}_v[s,u], r[u] \leftarrow v$-absorbed-push$(s, v, r_{max})$
2  for $i = 1 : n$ do
3      $\hat{r}(s,u) \leftarrow R[s] + R[u] - 2\frac{\hat{\tau}_v[s,u]}{d_u}$;
4  return $\hat{r}(s,u)$ for all $u \in V$ and $u \neq s$;
```

Query processing. We propose two approaches to compute the s-th row of (L_v^{-1}). The first approach is based on Lemma 3.4. We can repetitively simulate v-absorbed random walks from a node s; the normalized expected number of visits on a node u is $(L_v^{-1})_{su}$. The resulting algorithm AbWalk* is shown in Algorithm 8. An index array R is taken as an input, where $R[u] = r(v,u)$ and is pre-computed using Algorithm 7. The time complexity of AbWalk* is almost the same as that of Algorithm 2, with only an $O(n)$ additional term to compute and output $O(n)$ answers (Lines 4-6).

The second approach is to apply a v-absorbed push (Algorithm 1) to estimate the s-th row of L_v^{-1} which results in our Push* algorithm. The pseudo-code of Push* is shown in Algorithm 9. Again, the time complexity of Push* is almost the same as that of Algorithm 1, with only an $O(n)$ additional term to compute and output $O(n)$ answers (Lines 2-4). Note that similar to AbWalk and Push, the time complexity of AbWalk* and Push* is closely related to the landmark node v. Likewise, a good landmark should be an easy-to-hit node. Thus, the same landmark selection strategies proposed in Section 4.5 can also be applied for our index-based algorithms.

6 EXPERIMENTS

6.1 Experimental setup

Datasets and query sets. We use 5 real-life datasets, including 2 small graphs and 3 large graphs. All datasets can be downloaded

Table 2: Datasets

Type	Dataset	n	m	$\bar{d}$	$\bar{h}$	Δ_G
Small graphs	Facebook	4,039	88,233	43.69	767	8
	Hep-th	8,638	24,806	5.74	1189	18
	CAIDA	26,475	53,381	4.03	85	17
	Astro-ph	17,903	196,972	22	835	14
	Email-enron	33,696	180,811	10.73	4783	13
Large graphs	Amazon	334,863	925,872	5.53	14230	47
	DBLP	317,080	1,049,866	6.62	7564	23
	Youtube	1,134,890	2,987,624	5.27	269	24
	Pokec	1,632,803	22,301,964	27.32	3169	14
	Orkut	3,072,441	117,184,899	76.28	7336	10

from [29]. Note that for a non-connected graph, since the resistance distance between nodes in different connected components are infinite, we can process each connected component separately. As a result, we precompute the largest connected component of all datasets, the detailed statistics of the largest connected components are summarized in Table 2. Specifically, $\bar{d} = \frac{2m}{n}$ denotes the average degree, Δ_G is the diameter of the graph, $\bar{h}$ is the average hitting time from all nodes to the landmark node (the highest degree node). Suppose that v is the highest-degree node, $\bar{h} = \sum_{s \in V} \frac{1}{n} h(s, v)$. We estimate $\bar{h}$ by randomly generating 10^6 nodes uniformly and simulating v-absorbed random walks from these nodes. The smaller $\bar{h}$ is, the smaller $h(s, v) + h(t, v)$ will be for any two nodes s and t. For small graphs, we are able to compute the exact resistance distance between any pair of nodes via computing the L_v^{-1} matrix. The exact resistance distance is used as the ground truth to evaluate the estimation errors of different approximation algorithms. For large graphs, it is very difficult to obtain the exact resistance distances by existing approaches. Thus, for single-pair query, we use Bipush with a large sample size T ($T = 10^6$) and a small $r_{\max}$ ($r_{max} = 10^{-6}$) to compute a high-precision resistance distance estimation, and use such a high-precision estimation as the ground truth. Similarly, for single-source query, we first use a relatively-large sample size T ($T = 10^5$) to build the index, and then invoke Push* with a small $r_{\max}$ ($r_{\max} = 10^{-7}$) to derive high-precision single-source query estimations as the ground-truth results.

For single-pair queries, we uniformly sample 1000 node pairs and use them as the query set. The error of different algorithms is measured by $|\hat{r}(s, u) - r(s, u)|$, where $\hat{r}(s, u)$ and $r(s, u)$ are the estimated resistance distance and the ground truth respectively. For single-source queries, we uniformly sample 50 source nodes and use them as the query set. We use the L_1-error to evaluate the error of various single-source query processing algorithms, which is defined as $\sum_{u \in V} |\tilde{r}(s, u) - r(s, u)|$.

Different algorithms. For single-pair query, we compare our algorithms with two state-of-the-art algorithms Commute and Akp proposed in [44]. Commute is based on estimating the commute time of the random walk, while Akp is based on estimating the truncated transition probability. We do not include other previous algorithms because all of them are outperformed by Commute and Akp [44]. For Commute and Akp, we use their original implementations in [44]. For our solutions, we implement four different algorithms which are AbWalk (Algorithm 2), LocalTree (Algorithm 3), Push (Algorithm 5) and Bipush (Algorithm 6).

For single-source query, we compare our algorithms with two baselines: Base and Ust. Here Base is a baseline algorithm that invokes AbWalk $n - 1$ times to compute $n - 1$ single-pair queries. Ust is the algorithm proposed in [6] for computing the diagonal elements of $L^{\dagger}$. As we discussed in Section 5.1, such an algorithm can be modified to compute the single-source resistance distance query. For our solutions, we implement three different algorithms: an online algorithm LEwalk (Algorithm 7), and two index-based algorithm AbWalk* (Algorithm 8) and Push* (Algorithm 9).

Parameters. For Commute, there is a parameter ϵ, we set it as 0.1 following [44]. For Akp, there are two parameters the truncated parameter K and the sample size T; we set $K = 100$ and $T = 10^4$. The other methods can be categorized into two different types: i) sampling-based methods (including AbWalk, LocalTree, LEwalk, AbWalk*, Base, Ust); ii) push-based methods (including Push and Push*). For sampling-based methods, we set the sample size T to 10^4. For push-based methods, there is a parameter r_{max}, we set it to 10^{-4}. Bipush has both parameters T and r_{max}, we set $T = 10^4$ and $r_{max} = 10^{-4}$ respectively. We will evaluate the proposed algorithms by varying parameters in Section 6.4. All our algorithms requires to select a landmark node. We choose the highest-degree node as the landmark node by default. We will also study the effect of other landmark choices proposed in Section 6.6.

Experimental environment. All the experiments are conducted on a Linux 20.04 server with Intel 2.0 GHz CPU and 128GB memory. All the proposed algorithms are implemented in C++. For the baselines Akp and Commute, we use the open-source C++ implementations provided by their original authors [44]. All algorithms used in our experiments are complied using GCC9.3.0 with -O3 optimization.

6.2 Results of single-pair query

In this experiment, we compare the query time and estimation error of different algorithms for answering single-pair query. We use box-plot to illustrate the query time and error, so that we can clearly observe the distributions of the performance of different algorithms. Fig. 3 shows the running time of various algorithms. As can be seen, all the proposed algorithms AbWalk, LocalTree, Push and Bipush achieve significantly less query time than Commute on most datasets, which confirms our analysis in Section 2.2. We can also observe that the overall query time of AbWalk, LocalTree and Bipush are comparable with that of Akp. Among all competitors, the proposed Push algorithm is extremely fast on all datasets. The average query time of Push is at least two orders of magnitude lower than the state-of-the-art (SOTA) algorithms on large graphs. For example, on Pokec, the average query time of Push is 0.02 seconds, while the state-of-the-art Akp algorithm consumes around 17.5 seconds. These results demonstrate the high efficiency of the proposed algorithms.

Fig. 4 shows the estimation error of various algorithms. From Fig. 4, we can see that all the proposed algorithms AbWalk, LocalTree, Push and Bipush are much more accurate than two SOTA algorithms Commute and Akp. Compared to the other algorithms, Akp is much less accurate, which confirms our analysis in Section 2.2. The smaller $\bar{h}$ is, the faster AbWalk and LocalTree will be. LocalTree performs worse than AbWalk on Youtube which has a relatively large diameter. We also observe that our Bipush algorithm is extremely accurate whose average estimation error can be up to four orders of magnitude lower than that of Commute. For example, on CAIDA, the average estimation error of Bipush is round 4×10^{-6}, while the average error of the SOTA algorithm Commute is round 4×10^{-2}. These results indicate that the proposed algorithms can achieve a very high precision in answering single-pair resistance distance query.

In summary, for real-life datasets with a relatively small $\bar{h}$, our AbWalk and LocalTree algorithms are better than the state-of-the-art algorithms. LocalTree is more suitable for graphs with small diameters. Bipush has the best overall performance on all datasets, as it can achieve a very high accuracy and consumes comparable time as the other competitors. As a result, we recommend to apply

Bipush to approximate single-pair resistance distance computations.

6.3 Results of single-source query

In this experiment, we study the query time and estimation error of different algorithms for answering single-source queries. We use box-plot to show the performance of various algorithms. We also use a triangle to denote the index-building time as well as the estimation error using LEwalk. We compare three online methods, Base, Ust and LEwalk, and two index-based methods, AbWalk* and Push*. The results of query time of various algorithms is shown in Fig. 5. As can be seen, both LEwalk and Ust are much faster than Base which is consistent with our analysis in Section 5. On large datasets, Base cannot terminate within 10 hours. In general, LEwalk is faster than Ust because it avoids the operations of matching two trees as we discussed in Section 5.1. For the index-based methods, the index-building time is generally lower than an online single-source query time, because the time overhead $O(Tr((I-P_v)^{-1}))$ is often lower for the landmark node v, compared to the other query nodes. Both AbWalk* and Push* can answer a single-source query in very short time even on the largest graph; and they are $3 \sim 4$ orders of magnitude faster than the online algorithms. Especially, Push* is extremely fast, and it can answer single-source queries within around 1 second on all large graphs. These results demonstrate the high-efficiency of the proposed algorithms. Fig. 6 depicts the estimation errors of different algorithms. We can see that the estimation errors of the index-based methods are comparable to online algorithms. Also, we can see that LEwalk achieves slightly lower errors than Ust. These results demonstrate that our solutions are very efficient and effective to handle single-source queries. For the index size, it only requires an $O(n)$ array to store the resistance distance values, thus it is much smaller than the graph size. We omit the results for evaluating the index size due to the space limit.

In summary, for online algorithms, we recommend to use LEwalk to approximately compute the single-source resistance distance; for index-based algorithms, we recommend to apply LEwalk to construct an index for the landmark node, and use Push* to answer the single-source resistance distance query.

6.4 The effect of different parameters

First, we evaluate the performance of the proposed algorithms for answering single-pair queries when varying T and $r_{\max}$. For AbWalk, LocalTree and Bipush, we vary the parameter T (i.e., the sample size) from 10^2 to 10^6; and for Push and Bipush, we vary the parameter $r_{\max}$ from 10^{-3} to 10^{-7}. The results on Facebook are shown in Fig. 7; and similar results can also be observed on the other datasets. As expected, with T increasing, the query time of AbWalk, LocalTree and Bipush increases, and their estimation errors decrease. Similarly, with $r_{\max}$ decreasing, the query time of Push and Bipush increases and the errors getting smaller. Again, we find that Bipush is much more accurate than the other algorithms; and its estimation error can even be less than 10^{-9} when $r_{\max} = 10^{-5}$. These results further confirm the efficiency and effectiveness of our algorithms.

Second, for the single-source query, we vary the parameter T in Ust, LEwalk and AbWalk*, and vary the parameter r_{max} in Push*. Also, the index-building algorithm has a parameter T, we refer to it as LEwalk-v. The results on Facebook are shown in Fig. 8. For the other datasets, the results are consistent. As can be seen from Fig. 8, the query time becomes larger and the L_1-error becomes smaller as T getting larger for Ust, LEwalk and AbWalk*. Likewise, as r_{max} decreases, the query time of Push* increases and the error decreases. Again, from Fig. 8(a), we can observe that LEwalk is slightly faster than Ust; both LEwalk and Ust are much slower than the index-based method AbWalk*. From Fig. 8(c), our index-building algorithm has a lower error than the single-source query processing algorithms. The reason may be that for the selected landmark node is often easier to hit than the other source nodes, thus the loop-erased random walk trajectories generated by LEwalk is often not too long, which reduces the variance of the estimator. These results further conform that our index-based solutions are very accurate.

6.5 The impact of the landmark node

In this experiment, we evaluate the impact of the landmark node v on all proposed algorithms. Three heuristic landmark selection strategies including Degree, Core and PageRank are evaluated. Here Degree chooses the node with the largest degree, Core picks the node with the largest core number [39], and PageRank selects the node with the highest PageRank value. The results on Facebook are shown in Fig. 9. Similar results can also be observed on the other datasets.

As shown in Figs. 9(a-b), the query time of all the algorithms is the lowest when using the highest-degree node as the landmark for both single-pair and single-source queries, followed by Core and PageRank. The reason may be that the highest-degree node is intuitively the most easy-to-hit node compared to the other nodes. For the estimation errors (Figs. 9(c-d)), Degree and Core can achieve comparable performance, and both of them are generally better than PageRank. These results suggest that the highest-degree node is a very good landmark node in practice which is also used in our previous experiments.

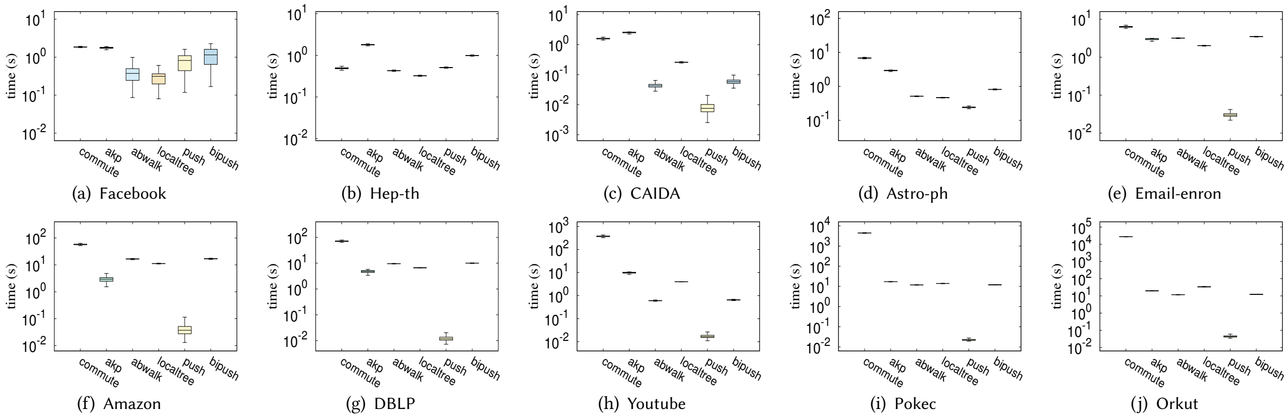
6.6 The impact of the landmark node

In this experiment, we evaluate the impact of the landmark node v on all proposed algorithms. Three heuristic landmark selection strategies including Degree, Core and PageRank are evaluated. Here Degree chooses the node with the largest degree, Core picks the node with the largest core number [39], and PageRank selects the node with the highest PageRank value. The results on Facebook are shown in Fig. 9. Similar results can also be observed on the other datasets.

As shown in Figs. 9(a-b), the query time of all the algorithms is the lowest when using the highest-degree node as the landmark for both single-pair and single-source queries, followed by Core and PageRank. The reason may be that the highest-degree node is intuitively the most easy-to-hit node compared to the other nodes. For the estimation errors (Figs. 9(c-d)), Degree and Core can achieve comparable performance, and both of them are generally better than PageRank. These results suggest that the highest-degree node is a very good landmark node in practice which is also used in our previous experiments.

6.7 The impact of the node ordering

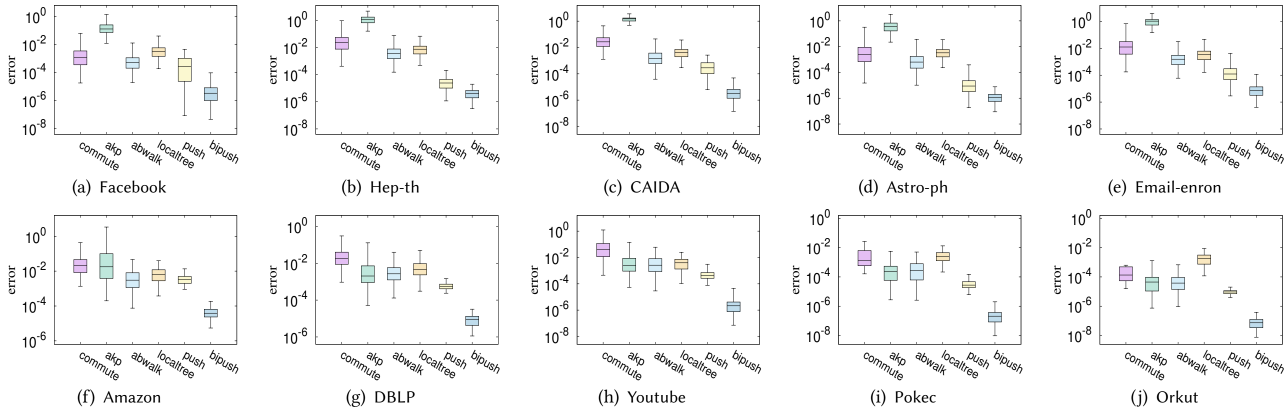
Recall that the v-absorbed push procedure (Algorithm 1) iteratively updates nodes with residual larger than a threshold. The performance of the algorithm may depend on the node-update ordering. In this experiment, we study the impact of different orderings. Specifically, we compare two ordering techniques: i) first-in-first-out (FIFO) ordering, and ii) priority ordering. For the priority ordering, we use the residual as the priority for each node, and the algorithm always processes the node with the largest residual. We apply Algorithm 1 to estimate $\tau_v[s,u]$ for all $u \in V$ and $u \neq v$ w.r.t. a query node s. For the query set, we generate 50 nodes uniformly and report the average result. We vary the parameter r_{max} from 10^{-3} to 10^{-7}. Fig. 10 shows the results on Youtube, and the results on the other datasets are consistent. As can be seen, the query time of Algorithm 1 with priority ordering is significantly higher than that of Algorithm 1 with FIFO ordering. However, the $L1$-errors of Algorithm 1 with these two orderings are almost the same. This result



(a) Facebook (b) Hep-th (c) CAIDA (d) Astro-ph (e) Email-enron

(f) Amazon (g) DBLP (h) Youtube (i) Pokec (j) Orkut

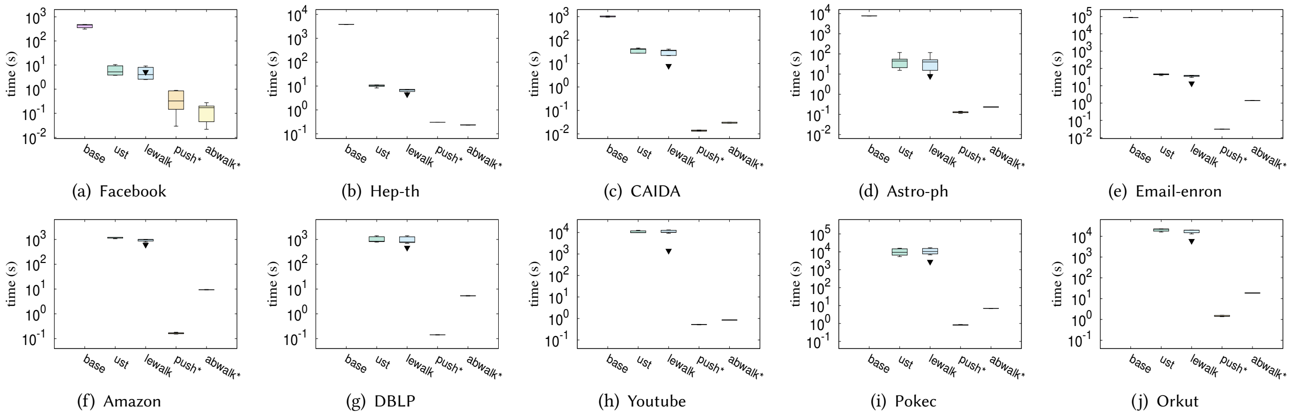
Figure 3: Runtime of different algorithms for answering single-pair queries



(a) Facebook (b) Hep-th (c) CAIDA (d) Astro-ph (e) Email-enron

(f) Amazon (g) DBLP (h) Youtube (i) Pokec (j) Orkut

Figure 4: Estimation errors of different algorithms for answering single-pair queries



(a) Facebook (b) Hep-th (c) CAIDA (d) Astro-ph (e) Email-enron

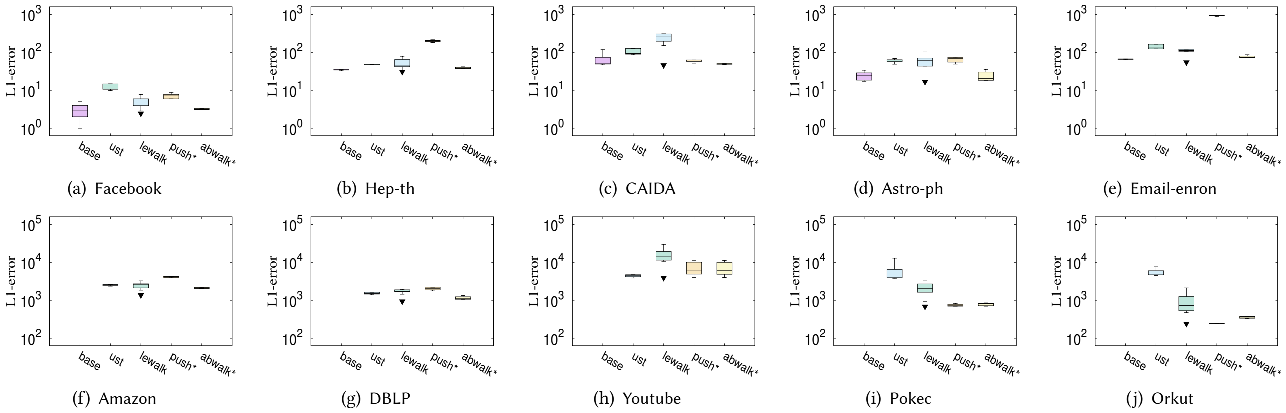
(f) Amazon (g) DBLP (h) Youtube (i) Pokec (j) Orkut

Figure 5: Runtime of different algorithms for single-source query (the triangle in each figure denotes the index building time)

indicates that it is sufficient to use a FIFO ordering in our v-absorbed push algorithm, which is also used in our previous experiments.

6.8 Case Studies

In this experiment, we conduct two case studies to study the effectiveness of the resistance distance related metrics. Recall that the resistance distance $r(s,t)$ measures the similarity between t and s; the smaller $r(s,t)$ is, the more similar s and t are. However, as shown in [54], $r(s,t) \approx 2m(\frac{1}{d_s}+\frac{1}{d_t})$ on geometric graphs. That is, the resistance distance between two nodes are mainly determined by their degrees. To fix this issue, the authors in [54] proposed a corrected resistance distance defined as $\tilde{r}(s,t)=\sqrt{r(s,t)-\frac{1}{d_s}-\frac{1}{d_t}-(\frac{1}{d_s}-\frac{1}{d_t})^2}$. Note that the key to compute such a corrected resistance distance is to calculate the resistance distance, thus our techniques can be directly used.



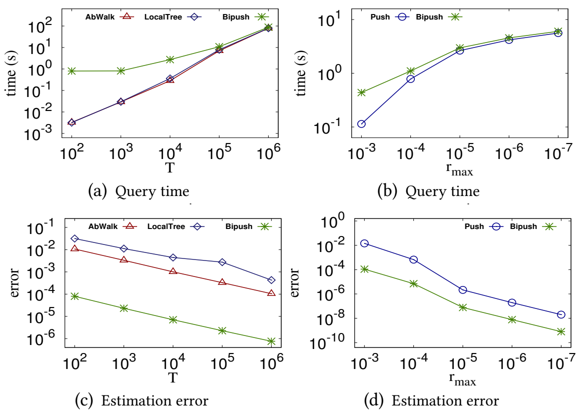
(a) Facebook (b) Hep-th (c) CAIDA (d) Astro-ph (e) Email-enron

(f) Amazon (g) DBLP (h) Youtube (i) Pokec (j) Orkut

Figure 6: Estimation error of different algorithms for single-source query (the triangle in each figure denotes the L_1-errors of the index)

Table 3: Top-10 similar results w.r.t. a query "Leman Akoglu" on DBLP and a query "decrease" on WordNet dataset

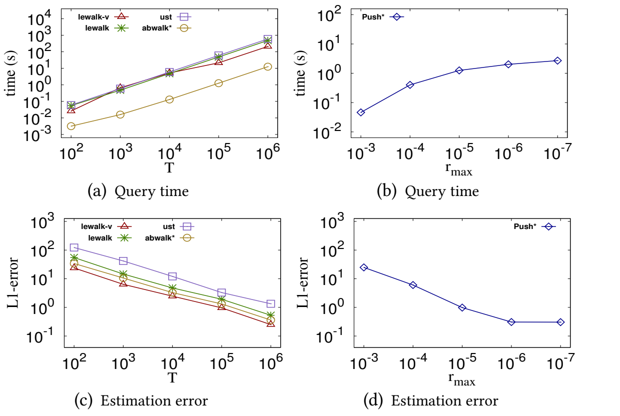
	DBLP					WordNet				
Rank	PageRank	SimRank	Res	Degree	Corrected-Res	PageRank	SimRank	Res	Degree	Corrected-Res
1	Christos Faloutsos	Sadegh M. Milajerdi	Christos Faloutsos	Jiawei Han	Disha Makhija	small	diminish	money	food	budget
2	Hanghang Tong	Emaad A. Manzoor	Jiawei Han	Philip S. Yu	Keith Henderson	lower	lower	food	money	diminish
3	Stephan Gunnemann	Hau Chan	Philip S. Yu	Christos Faloutsos	Shebuti Rayana	down	reduce	water	water	shrink
4	Neil Shah	Shuchu Han	Michael J. Franklin	Michael J. Franklin	Alex Beutel	reduce	minimum	car	car	descend
5	Emmanuel Muller	Tianmin Zou	Raghu Ramakrishnan	Jian Pei	Bryan Hooi	diminish	shrink	good	good	increase
6	B. Aditya Prakash	Roni Rosenfeld	Beng Chin Ooi	Gerhard Weikum	Brian Gallagher	decline	descent	bad	bad	reduce
7	Sadegh M. Milajerdi	Kailash Budhathoki	Jian Pei	Raghu Ramakrishnan	Christos Faloutsos	increase	decline	work	work	decline
8	Emaad A. Manzoor	Apratim Bhattacharyya	Gerhard Weikum	Michael Stonebraker	Hanghang Tong	less	minus	house	school	lower
9	Tina Eliassi-Rad	Koen Smets	Jeffrey F. Naughton	Beng Chin Ooi	Roger Magoulas	shrink	descend	school	house	depletion
10	Lei Li	Disha Makhija	Hector Garcia-Molina	Jeffrey F. Naughton	Tim O'Reilly	descend	increase	love	love	temperature



(a) Query time (b) Query time

(c) Estimation error (d) Estimation error

Figure 7: Single-pair query with varying T and r_{max} (Facebook)

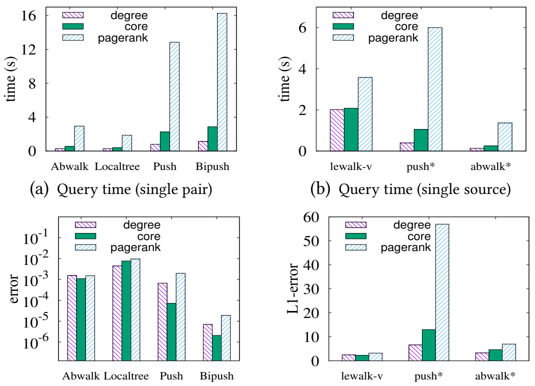


(a) Query time (b) Query time

(c) Estimation error (d) Estimation error

Figure 8: Single-source query with varying T and r_{max} (Facebook)

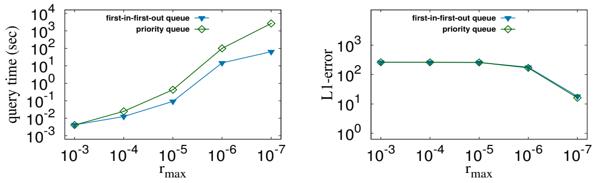
We compare the resistance distance (denoted by Res) and the corrected resistance (denoted by Corrected-Res) with two widely-used similarity measures PageRank [60] and SimRank [23]. In addition, we also include a degree-based metric (denoted by Degree) for comparison which is defined by $\frac{1}{d_s} + \frac{1}{d_t}$. We use two real-life datasets DBLP and WordNet for our case studies. Specifically, DBLP [1] is a co-authorship network collected from recent 10-year publications (2006-2016) in the database area. The DBLP dataset contains 37,177 nodes and 131,715 edges. WordNet [2] is an English word graph parsed from the text of English Wikipedia, where each node is a word and each edge is a dependency between words. WordNet contains 5,040 nodes and 55,258 edges. We aims to find the most similar nodes w.r.t. a source node s on these datasets. Specifically, we compute the top-10 nodes with the lowest Res (Corrected-Res and Degree) values as well as the highest PageRank (SimRank) values. The results are shown in Tabel 3. As expected, for PageRank and SimRank the top-10 similar users of "Leman Akoglu" are all authors that have a close relationship with "Leman Akoglu" (who has co-authored more than one paper with him). Most of the top-10 results of Res, however, are well-known researchers in the database area, indicating that the resistance distance metric tends to be a *global measure* of importance. Moreover, we can see that the results of Res and Degree are very similar, which is consistent with the theoretical analysis in [54]. Interestingly, for the corrected resistance distance,



(a) Query time (single pair) (b) Query time (single source)

(c) Estimation error (single pair) (d) Estimation error (single source)

Figure 9: The effect of the landmark node v (Facebook)



(a) time (b) error

Figure 10: Evaluation of the impact of node-update ordering in the v-absorbed push procedure (Algorithm 1)

the top-10 results are highly similar to "Leman Akoglu" and its performance is comparable with those of PageRank and SimRank. These results show the high effectiveness of such a corrected resistance distance for measuring similarity on graphs. We can observe similar results on WordNet. To answer a query “decrease", both PageRank and SimRank return words that have similar meanings w.r.t. “decrease". The results of Res are all commonly-used words, which further demonstrate the global property of the resistance distance metric. Likewise, for the corrected resistance distance, the results are very similar to the query word “decrease" and are also comparable with the results of both PageRank and SimRank. These results further confirm the effectiveness of the corrected resistance distance.

In summary, our case studies show that the original resistance distance may not be suitable for measuring the similarities of the nodes in graph. However, a slight correction of the resistance distance can be a very good similarity measure. These results confirm the theoretical analysis shown in [54]. In addition, the corrected resistance distance is a distance metric which can be directly used for many downstream machine learning applications [54], thus we believe that our techniques can be very useful for those applications. Both the PageRank and SimRank, however, are not a distance metric, thus may limit their use in machine learning applications.

7 RELATED WORK

Resistance distance computation. There is a large number of studies on the properties of the resistance distance [8, 11, 14, 53, 55]. Previous studies have already revealed the connection between resistance distance and the commute time of the random walk on graphs [11, 53]. The connection with random spanning trees has also revealed previously [36]. Our new theoretical results substantially generalize these connections and bring new insights into the research of resistance distance. The computation of resistance distance is also studied in theoretical points of view [37, 38, 52]. However, most of these algorithms perform poorly in practice [44]. There exist several practical algorithms that focus on computing the so-called spanning tree centrality [22, 40], which is a special case of resistance distance $r(s,t)$ when there is an edge between s and t. Note that those algorithms do not work for single-pair resistance distance computation. Recently, Peng et al. develop several efficient algorithms to compute the single-pair resistance distance based on random walk sampling. Compared to their algorithms, our landmark-based algorithms not only significantly reduce query time, but can also support single-source query.

Personalized PageRank computation. There exist many algorithms for computing personalized PageRank [4, 5, 7, 10, 16, 25, 32–34, 47, 50, 56, 58, 60, 62, 63]. Among them, [16, 25, 50, 63] are based on matrix operations which are often expensive for large graphs. Another set of methods are based on random walk sampling [7, 33, 47] which are also not very efficient to achieve a good estimation accuracy on large graphs. More efficient algorithms for personalized PageRank computation are push based deterministic algorithms [4, 5, 10]. Recently, such push based solutions are further optimized by a bidirectional framework that combines both push algorithms and random walk sampling [30, 32, 34, 58–60, 62]. Motivated by this bidirectional framework, we also propose a bidirectional algorithm to compute resistance distance. However, as well discussed in Section 4, it is quite nontrivial to extend the existing random walk sampling algorithms (push algorithms) to our new v-absorbed random walk algorithm (v-absorbed push algorithm).

8 CONCLUSION

In this paper, we study the problem of approximately computing the resistance distance on real-life network. Based on a novel formula for computing $r(s,t)$, we propose several novel explanations for resistance distance and also develop several new and efficient algorithms for both single-pair and single-source resistance distance query problems. Our results establish several interesting connections among resistance distance, random walk, random spanning trees, and deterministic push procedure; and bring new and deep insights on efficient resistance distance computations. We conduct extensive experiments on 5 real-life datasets to evaluate our algorithms. The results show that for single-pair query, our best algorithm Bipush outperforms the state-of-the-art algorithms by at least three orders of magnitude in terms of the average error using similar query time. For single-source query, the proposed online algorithm LEwalk is significantly faster than the baselines and the proposed index-based method Push* is at least three orders of magnitude faster than the baselines, with comparable approximation errors (i.e., $L1$-error).

REFERENCES

[1] 2016. DBLP: DBLP Collaboration Network. http://dblp.uni-trier.de/~ley/db.

[2] 2022. Project WordGraph. http://www.ims.uni-stuttgart.de/en/research/projects/wordgraph/.

[3] Vedat Levi Alev, Nima Anari, Lap Chi Lau, and Shayan Oveis Gharan. 2018. Graph Clustering using Effective Resistance. In *9th Innovations in Theoretical Computer Science Conference, ITCS*.

[4] Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. 2008. Local Computation of PageRank Contributions. *Internet Math.* (2008), 23–45.

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

[6] 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 *ESA*.

[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] Wayne Barrett, Emily J. Evans, Amanda E. Francis, Mark Kempton, and John Sinkovic. 2020. Spanning 2-forests and resistance distance in 2-connected graphs. *Discret. Appl. Math.* 284 (2020), 341–352.

[10] Pavel Berkhin. 2006. Bookmark-Coloring Approach to Personalized PageRank Computing. *Internet Math.* 3, 1 (2006), 41–62.

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

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

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

[14] Pavel Chebotarev and Elena Deza. 2020. Hitting time quasi-metric and its forest representation. *Optim. Lett.* (2020), 291–307.

[15] Paul F. Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. 2011. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *STOC*.

[16] Mustafa Coskun, Ananth Grama, and Mehmet Koyutürk. 2016. Efficient Processing of Network Proximity Queries via Chebyshev Acceleration. In *KDD*. 1515–1524.

[17] Mustafa Coskun, Ananth Grama, and Mehmet Koyutürk. 2018. Indexed Fast Network Proximity Querying. *VLDB* 11, 8 (2018), 840–852.

[18] Peter G Doyle and J Laurie Snell. 1984. *Random walks and electric networks*. Vol. 22. American Mathematical Soc.

[19] Peter G. Doyle and J. Laurie Snell. 1984. Random Walks and Electrical Networks. *Mathematical Association of America, Washington* (1984).

[20] 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.

[21] Massimo Franceschet and Enrico Bozzo. 2017. Approximations of the Generalized Inverse of the Graph Laplacian Matrix. *Internet Math.* (2017).

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

[23] Glen Jeh and Jennifer Widom. 2002. SimRank: a measure of structural-context similarity. In *KDD*.

[24] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *WWW*. 271–279.

[25] Jinhong Jung, Namyong Park, Lee Sael, and U Kang. 2017. BePI: Fast and Memory-Efficient Method for Billion-Scale Random Walk with Restart. In *SIGMOD*. 789–804.

[26] Heung-Nam Kim and Abdulmotaleb El-Saddik. 2011. Personalized PageRank vectors for tag recommendations: inside FolkRank. In *ACM Conference on Recommender Systems*.

[27] Jérôme Kunegis and Stephan Schmidt. 2007. Collaborative Filtering Using Electrical Resistance Network Models. In *Industrial Conference on Data Mining*.

[28] Katz Leo. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18, 1 (1953), 39–43.

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

[30] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, and Guoren Wang. 2022. Efficient Personalized PageRank Computation: A Spanning Forest Sampling based Approach. In *SIGMOD*. 1996–2008.

[31] David Liben-Nowell and Jon M. Kleinberg. 2003. The link prediction problem for social networks. In *CIKM*.

[32] Dandan Lin, Raymond Chi-Wing Wong, Min Xie, and Victor Junqiu Wei. 2020. Index-Free Approach with Theoretical Guarantee for Efficient Random Walk with Restart Query. In *ICDE*. 913–924.

[33] Qin Liu, Zhenguo Li, John C. S. Lui, and Jiefeng Cheng. 2016. PowerWalk: Scalable Personalized PageRank via Random Walks with Vertex-Centric Decomposition. In *CIKM*. 195–204.

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

[35] Peter Lofgren and Ashish Goel. 2013. Personalized PageRank to a Target Node. *CoRR* abs/1304.4658 (2013). arXiv:1304.4658 http://arxiv.org/abs/1304.4658

[36] László Lovász. 1993. Random walks on graphs. *Combinatorics, Paul erdos is eighty* 2, 1-46 (1993), 4.

[37] Russell Lyons and Shayan Oveis Gharan. 2018. Sharp bounds on random walk eigenvalues via spectral embedding. *International Mathematics Research Notices* 2018, 24 (2018), 7555–7605.

[38] Aleksander Madry, Damian Straszak, and Jakub Tarnawski. 2015. Fast Generation of Random Spanning Trees and the Effective Resistance Metric. In *SODA*. 2019–2036.

[39] 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.

[40] Charalampos Mavroforakis, Richard Garcia-Lebron, Ioannis Koutis, and Evimaria Terzi. 2015. Spanning Edge Centrality: Large-scale Computation and Applications. In *WWW*. 732–742.

[41] Qiaozhu Mei, Dengyong Zhou, and Kenneth Ward Church. 2008. Query suggestion using hitting time. In *CIKM*.

[42] Eisha Nathan and David A. Bader. 2018. Incrementally updating Katz centrality in dynamic graphs. *Soc. Netw. Anal. Min.* 8, 1 (2018), 26.

[43] Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. 2021. Graph Kernels: A Survey. *J. Artif. Intell. Res.* 72 (2021), 943–1027.

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

[45] James Gary Propp and David Bruce Wilson. 1998. How to get a perfectly random sample from a generic Markov chain and generate a random spanning tree of a directed graph. *Journal of Algorithms* 27, 2 (1998), 170–217.

[46] Purnamrita Sarkar, Andrew W. Moore, and Amit Prakash. 2008. Fast incremental proximity search in large graphs. In *ICML*.

[47] Tamás Sarlós, András A. Benczúr, Károly Csalogány, Dániel Fogaras, and Balázs Rácz. 2006. To randomize or not to randomize: space optimal summaries for hyperlink analysis. In *WWW*. 297–306.

[48] Aaron Schild, Satish Rao, and Nikhil Srivastava. 2018. Localization of Electrical Flows. In *SODA*, Artur Czumaj (Ed.).

[49] Jieming Shi, Tianyuan Jin, Renchi Yang, Xiaokui Xiao, and Yin Yang. 2020. Realtime Index-Free Single Source SimRank Processing on Web-Scale Graphs. *Proc. VLDB Endow.* 13, 7 (2020), 966–978.

[50] Kijung Shin, Jinhong Jung, Lee Sael, and U Kang. 2015. BEAR: Block Elimination Approach for Random Walk with Restart on Large Graphs. In *SIGMOD*, Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives (Eds.). 1571–1585.

[51] Ali Kemal Sinop, Lisa Fawcett, Sreenivas Gollapudi, and Kostas Kollias. 2021. Robust Routing Using Electrical Flows. In *SIGSPATIAL '21: 29th International Conference on Advances in Geographic Information Systems*.

[52] Daniel A. Spielman and Nikhil Srivastava. 2008. Graph sparsification by effective resistances. In *STOC*.

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

[54] Ulrike von Luxburg, Agnes Radl, and Matthias Hein. 2010. Getting lost in space: Large sample analysis of the resistance distance. In *NIPS*. 2622–2630.

[55] Ulrike Von Luxburg, Agnes Radl, and Matthias Hein. 2010. Hitting and commute times in large graphs are often misleading. *arXiv:1003.1266* (2010).

[56] Hanzhi Wang, Zhewei Wei, Junhao Gan, Sibo Wang, and Zengfeng Huang. 2020. Personalized PageRank to a Target Node, Revisited. In *KDD*. 657–667.

[57] Shuguang Wang and Milos Hauskrecht. 2010. Effective query expansion with the resistance distance based term similarity metric. In *SIGIR*.

[58] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. HubPPR: Effective Indexing for Approximate Personalized PageRank. *VLDB* 10, 3 (2016), 205–216.

[59] Sibo Wang, Renchi Yang, Runhui Wang, Xiaokui Xiao, Zhewei Wei, Wenqing Lin, Yin Yang, and Nan Tang. 2019. Efficient Algorithms for Approximate Single-Source Personalized PageRank Queries. *TODS* (2019), 18:1–18:37.

[60] 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.

[61] David Bruce Wilson. 1996. Generating Random Spanning Trees More Quickly than the Cover Time. In *STOC*.

[62] 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.

[63] Minji Yoon, Jinhong Jung, and U Kang. 2018. TPA: Fast, Scalable, and Accurate Method for Approximate Random Walk with Restart on Billion Scale Graphs. In *ICDE*. 1132–1143.

[64] Zhen Zhang, Mianzhi Wang, Yijian Xiang, Yan Huang, and Arye Nehorai. 2018. RetGK: Graph Kernels based on Return Probabilities of Random Walks. In *NeurIPS*.