
# Theoretically and Practically Efficient Resistance Distance Computation on Large Graphs

Yichun Yang
Beijing Institute of Technology
Beijing, China
yc.yang@bit.edu.cn

Longlong Lin
Southwest University
Chongqing, China
longlonglin@swu.edu.cn

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

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

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

## ABSTRACT

The computation of resistance distance is pivotal in a wide range of graph analysis applications, including graph clustering, link prediction, and graph neural networks. Despite its foundational importance, efficient algorithms for computing resistance distances on large graphs are still lacking. Existing state-of-the-art (SOTA) methods, including power iteration-based algorithms and random walk-based local approaches, often struggle with slow convergence rates, particularly when the condition number of the graph Laplacian matrix, denoted by $\kappa$, is large. To tackle this challenge, we propose two novel and efficient algorithms inspired by the classic Lanczos method: Lanczos Iteration and Lanczos Push, both designed to reduce dependence on $\kappa$. Among them, Lanczos Iteration is a near-linear time global algorithm, whereas Lanczos Push is a local algorithm with a time complexity independent of the size of the graph. More specifically, we prove that the time complexity of Lanczos Iteration is $\tilde{O}(\sqrt{\kappa}m)$ ($m$ is the number of edges of the graph and $\tilde{O}$ means the complexity omitting the log terms) which achieves a speedup of $\sqrt{\kappa}$ compared to previous power iteration-based global methods. For Lanczos Push, we demonstrate that its time complexity is $\tilde{O}(\kappa^{2.75})$ under certain mild and frequently established assumptions, which represents a significant improvement of $\kappa^{0.25}$ over the SOTA random walk-based local algorithms. We validate our algorithms through extensive experiments on eight real-world datasets of varying sizes and statistical properties, demonstrating that Lanczos Iteration and Lanczos Push significantly outperform SOTA methods in terms of both efficiency and accuracy.

**PVLDB Reference Format:**
Yichun Yang, Longlong Lin, Rong-Hua Li, Meihao Liao, and Guoren Wang. Theoretically and Practically Efficient Resistance Distance Computation on Large Graphs. PVLDB, 19(8): 1688 - 1700, 2026.
doi:10.14778/3811243.3811244

**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://github.com/Ychun-yang/LanczosPush.


Proceedings of the VLDB Endowment, Vol. 19, No. 8 ISSN 2150-8097.
doi:10.14778/3811243.3811244

## 1 INTRODUCTION

Resistance distance (RD) computation is a fundamental task in graph data management, offering a powerful means to quantify the node similarities of a graph. Given an undirected graph $\mathcal{G}$, the resistance distance $r_{\mathcal{G}}(s,t)$ of two vertices $s$ and $t$ is proportional to the expected number of steps taken by a random walk starting at $s$, visits $t$ and then comes back to $s$. Consequently, a small $r_{\mathcal{G}}(s,t)$ indicates a high similarity between $s$ and $t$ [20, 49]. By effectively quantifying node similarities, RD emerges as a highly versatile and potent tool, finding numerous real-world applications, including graph clustering [37], link prediction [36], graph sparsification [40], maximum flow computation [26, 42], and graph neural networks [4, 53]. Therefore, designing fast algorithms for RD computation is a natural problem that has attracted extensive attention in recent years [14, 18–22, 33, 49, 51].

However, computing exact $r_{\mathcal{G}}(s,t)$ is equivalent to solving a linear system, which is only known to be solvable in $O(n^{2.37})$ time by the complex matrix multiplication [47], which is inefficient for massive graphs. Thus, many approximate algorithms have been developed recently for estimating $r_{\mathcal{G}}(s,t)$ with absolute error guarantee [20, 22, 33, 49] (details in Definition 2.2) to balance the efficiency and quality. Existing approximate methods can be roughly classified into two categories: (i) algorithms with nearly linear time complexity; (ii) local algorithms that do not depend on graph size. For example, Cohen et al. [11] developed a global Laplacian solver capable of achieving a $\epsilon$-absolute error approximation within a time complexity of $O(m\log^{1/2} n \log\frac{1}{\epsilon})$, where $n$ (resp., $m$) is the number of vertices (resp., edges) of the graph and $\epsilon$ is the absolute error parameter. However, though the Laplacian solver is theoretically very powerful, it remains practically inefficient for real-world datasets due to the complex data structures and large constant and log terms hidden in the complexity. Thus, the simpler Power Method [33, 34, 49] with an $\epsilon$-absolute error approximation is proposed, which can be implemented easier than the Laplacian solver. More specifically, the time complexity of Power Method [33, 34, 49] is $O(\kappa m\log\frac{\kappa}{\epsilon})$, where $\kappa$ is the condition number of the normalized Laplacian matrix $\mathcal{L}$ (details in Sec. 2). However, the Power Method remains computationally prohibitive for graphs with large condition number $\kappa$ due to its inherent linear dependency on $\kappa$ in time

complexity. Even in the optimal case of $\kappa = O(1)$ (i.e., expander graphs [14]), the time complexity of Power Method reduces to $O(m \log \frac{1}{\epsilon})$. However, this reduction still has to perform $O(\log \frac{1}{\epsilon})$ matrix-vector multiplication operations, rendering it inefficient for massive graphs since such each operation needs to traverse the entire graph.

To further boost the efficiency, researchers have recently turned to local algorithms[1] for estimating RD [33]. These algorithms operate by examining only localized subgraphs rather than requiring full graph access, thereby significantly reducing computational overhead. Informally, we consider the well-known adjacency model [34], which supports the following three types of queries on graph $\mathcal{G}$ in constant time: (i) degree query; (ii) neighbor query; (iii) jump query. The local algorithms aim to approximate $r_{\mathcal{G}}(s,t)$ by making as few queries as possible without searching the whole graph. For example, Peng et al. [33] proposed the random walk algorithm for local computation of $r_{\mathcal{G}}(s,t)$. Yang et al. [49] further improved this result by combining the Power Method with the random walk. However, the time complexity of these local algorithms are $\tilde{O}(\kappa^3/\epsilon^2)$[2] for estimating the $\epsilon$-absolute error approximation of $r_{\mathcal{G}}(s,t)$. These results show that although the complexity of local algorithms does not depend on $n$ and $m$, it comes at the cost of higher dependence on the condition number $\kappa$ and parameter $\epsilon$. Unfortunately, the condition number $\kappa$ can be very large in real-world graphs (Table 2). This makes the $\kappa^3$ bound very high. As shown in our experiments, we also find that existing algorithms for RD computation can only be efficiently implemented on datasets with a relatively small $\kappa$ (e.g., social networks), but their performance on datasets with a relatively large $\kappa$ (e.g., road networks) is unsatisfactory.

Therefore, a natural and important question is proposed: Can we design efficient linear time and sublinear time (local) algorithms for resistance distance computation, while the dependence on the condition number $\kappa$ is weaker? In this paper, we provide a positive answer to this question. First, we utilize the classic Lanczos Iteration algorithm to provide a global RD algorithm. Specifically, the Lanczos Iteration algorithm (details in Sec. 3) search a sequence of orthogonal vectors $\mathbf{v}_1, ..., \mathbf{v}_k$ lie in the $k$-dimension Krylov subspace $\mathcal{K}_k(\mathcal{A}, \mathbf{v}_1) = span\langle \mathbf{v}_1, \mathcal{A}\mathbf{v}_1, ..., \mathcal{A}^k\mathbf{v}_1 \rangle$ with $k = \sqrt{\kappa} \log \frac{\kappa}{\epsilon}$, where $\kappa$ denotes the condition number of the normalized Laplacian matrix $\mathcal{L}$. We prove that one can then approximate $r_{\mathcal{G}}(s,t)$ with $\epsilon$-absolute error guarantee by these orthogonal vectors $\mathbf{v}_1, ..., \mathbf{v}_k$ when setting $\mathbf{v}_1 = \left( \frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}} \right) / \sqrt{\frac{1}{d_s} + \frac{1}{d_t}}$, where $d_s, d_t$ denote the degree of the two specific nodes $s, t$. Next, by the implementation of Lanczos Iteration, $\mathbf{v}_1, ..., \mathbf{v}_k$ can be computed in $k$ matrix-vector queries. Therefore, this yields an $O(\sqrt{\kappa} m \log \frac{\kappa}{\epsilon})$ time algorithm for RD estimation, improves upon the Power Method by reducing a $\sqrt{\kappa}$ dependency. Subsequently, to obtain a local algorithm with less than $\kappa^3$ complexity, we introduce a novel concept of the local Lanczos recurrence (details in Eq. (3)). Using our novel techniques, we design the Lanczos Push algorithm (details in Sec. 4), which computes a sequence of sparse vectors $\hat{\mathbf{v}}_1, ..., \hat{\mathbf{v}}_k$ that is close to the orthogonal vectors $\mathbf{v}_1, ..., \mathbf{v}_k$ computed by Lanczos Iteration. As a consequence, our Lanczos Push algorithm estimates $r_{\mathcal{G}}(s,t)$ by only search only a small portion of the graph but still maintaining the acceleration effect by the Lanczos iteration. Under several mild and frequently-established assumptions (also have been verified in our experiments), we theoretically prove that our Lanczos Push algorithm has a time complexity $\tilde{O}(\kappa^{2.75} C_1 C_2 / (\epsilon \sqrt{d}))$ to obtain an absolute error approximation of the RD value, which is subcubic in $\kappa$. For the two numbers $C_1$ and $C_2$, we refer to Table 1 and Sec. 4 for details. On the other hand, we prove the lower bound of the time complexity is $\Omega(\kappa)$ for any algorithms to approximate $r_{\mathcal{G}}(s,t)$ with absolute error $\epsilon \leq 0.01$. This indicates that the time complexity of the local algorithms is indeed relevant to the condition number $\kappa$, which has not been emphasized in previous works for local RD computation[3].

We conduct extensive experiments to evaluate the effectiveness and scalability of the proposed solutions. The empirical results show that the performance of the proposed Lanczos Iteration and Lanczos Push algorithm substantially outperform all baselines. In particular, Lanczos Iteration performs 5× faster than the Power Method in social networks and 100× faster than Power Method in road networks. Lanczos Push is 5× to 10× faster than all the state-of-the-art algorithms in social networks and 50× faster than all other algorithms in road networks (including Lanczos Iteration). In short, our algorithms are both theoretically and practically efficient.

## 2 PRELIMINARIES

### 2.1 Notations and Concepts

Consider an undirected and connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = n$ vertices and $|\mathcal{E}| = m$ edges. In this paper, we primarily focus on unweighted graphs. However, all the algorithms and analysis can be directly generalized to weighed graphs. For any node $u \in \mathcal{V}$, its neighborhood is defined as $\mathcal{N}(u) = \{v | (u,v) \in \mathcal{E}\}$ and its degree is $d_u = |\mathcal{N}(u)|$. The degree matrix $\mathbf{D}$ is a $n \times n$ diagonal matrix with entries $\mathbf{D}_{i,i} = d_i$, while the adjacency matrix $\mathbf{A}$ satisfies $\mathbf{A}_{i,j} = 1$ if $(i,j) \in \mathcal{E}$ and $\mathbf{A}_{i,j} = 0$ otherwise. The Laplacian matrix is given by $\mathbf{L} = \mathbf{D} - \mathbf{A}$ with eigenvalues $0 = \lambda_1 < \lambda_2 \leq \lambda_3 \leq ... \leq \lambda_n$ and the corresponding eigenvectors $\mathbf{u}_1, ..., \mathbf{u}_n$. The Laplacian pseudo inverse is defined as $\mathbf{L}^\dagger = \sum_{i=2}^{n} \frac{1}{\lambda_i} \mathbf{u}_i \mathbf{u}_i^T$. Based on the definition of $\mathbf{L}^\dagger$, Resistance Distance (RD) is formulated as follows.

*Definition 2.1.* Given a pair of vertices $s$ and $t$ of a graph $\mathcal{G}$, the resistance distance of $s$ and $t$ is defined as $r_{\mathcal{G}}(s,t) = (\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{L}^\dagger (\mathbf{e}_s - \mathbf{e}_t)$. Where $\mathbf{e}_s$ (resp., $\mathbf{e}_t$) is the one-hot vector that takes value 1 at $s$ (resp., $t$) and 0 elsewhere.

Since computing exact RD requires $O(n^{2.37})$ time via complex matrix multiplication, prohibitively expensive for large graphs. Consequently, existing works have primarily focused on developing approximation algorithms with provable error guarantees to balance efficiency and quality. In this paper, following [20, 33, 49], we also focus on resistance distance with absolute error guarantee.

*Definition 2.2.* Given graph $\mathcal{G}$ and a pair of vertices $s$ and $t$, given any small constant $\epsilon > 0$, $\hat{r}_{\mathcal{G}}(s,t)$ is called $\epsilon$-absolute error approximation of $r_{\mathcal{G}}(s,t)$ iff $|r_{\mathcal{G}}(s,t) - \hat{r}_{\mathcal{G}}(s,t)| \leq \epsilon$.

[1] In this paper, the term "local" means the runtime of the algorithm independent or sublinear dependent on graph size $m$.

[2] $\tilde{O}(.)$ means the complexity omitting the log term of $n$, $m$ and $\epsilon$.

[3] There is still a large gap between the upper and lower bounds of the dependence of $\kappa$, so we leave how to match the upper and lower bounds as future work.

**Table 1: A comparison of different algorithms for computing pairwise RD with $\epsilon$-absolute error guarantee, where for Lanczos Push, $C_1 = \max_{u=s,t;i\le k} \|\mathbf{D}^{1/2}T_i(\mathbf{P})\mathbf{e}_u\|_1$, and $C_2 = \max_{i\le k}(\|\mathbf{v}_i\|_1 + \|\mathcal{A}\mathbf{v}_i^+\|_1 + \|\mathcal{A}\mathbf{v}_i^-\|_1)$, see Section 4 for details.**

| Methods | Algorithm | Time Complexity |
|---|---|---|
| Linear | Power Method [49] | $O(\kappa m \log \frac{\kappa}{\epsilon})$ |
| | LapSolver [35] | $O(m \log^{1/2} n \log \frac{1}{\epsilon})$ |
| | FastRD [25] | $\tilde{O}(m/\epsilon^2)$ |
| | Lanczos (Section 3) | $O(\sqrt{\kappa} m \log \frac{\kappa}{\epsilon})$ |
| Sublinear (Local) | TP [33] | $\tilde{O}(\kappa^4/\epsilon^2)$ |
| | TPC [33] | $\tilde{O}(\kappa^3/\epsilon^2)$ |
| | GEER [49] | $\tilde{O}(\kappa^3/(\epsilon^2 d^2))$ |
| | Algorithm 1 in [51] | $\tilde{O}(\kappa^3/(\epsilon\sqrt{d}))$ |
| | BiSPER [13] | $\tilde{O}(\kappa^{7/3}/\epsilon^{2/3})$ |
| | Lanczos Push (Section 4) | $\tilde{O}(\kappa^{2.75} C_1 C_2/(\epsilon\sqrt{d}))$ |
| Others | Push [20] | $\times$ |
| | BiPush [20] | $\times$ |
| | Lower Bound (Section 5) | $\Omega(\kappa)$ |

## 2.2 Existing Methods and Their Defects

For our analysis, we define the normalized adjacency matrix as $\mathcal{A} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ and define the normalized Laplacian matrix as $\mathcal{L} = \mathbf{I} - \mathcal{A}$. The condition number of $\mathcal{L}$ is defined as follows [14].

*Definition 2.3.* The condition number $\kappa$ of the normalized Laplacian matrix $\mathcal{L}$ is defined as $\kappa \triangleq \frac{2}{\mu_2}$, where $\mu_2$ is the second eigenvalue of $\mathcal{L}$, i.e., the smallest non-zero eigenvalue of $\mathcal{L}$.

Most existing RD algorithms have time complexity relevant to $\kappa$. Specifically, we let $\mathbf{P} \triangleq \mathbf{A}\mathbf{D}^{-1}$ be the probability transition matrix, and these existing algorithms are based on the following Taylor expansion formula [33, 51].

$$r_{\mathcal{G}}(s,t) = (\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{L}^{\dagger}(\mathbf{e}_s - \mathbf{e}_t) = \frac{1}{2}(\mathbf{e}_s - \mathbf{e}_t)^T \sum_{k=0}^{+\infty} \mathbf{D}^{-1}\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^k (\mathbf{e}_s - \mathbf{e}_t). \quad (1)$$

Thus, we can set a sufficiently large truncation step $l$, and define the approximation $\overline{r}_{\mathcal{G}}(s,t) = \frac{1}{2}(\mathbf{e}_s - \mathbf{e}_t)^T \sum_{k=0}^{l} \mathbf{D}^{-1}\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^k (\mathbf{e}_s - \mathbf{e}_t)$. To make $\overline{r}_{\mathcal{G}}(s,t)$ be the $\epsilon$-absolute error approximation of $r_{\mathcal{G}}(s,t)$, the truncation step $l = O(\kappa \log \frac{\kappa}{\epsilon})$ [33, 49, 51]. Based on this interpretation, the basic Power Method and random walk-based methods can be easily derived. Table 1 summarizes the state-of-the-art methods and our solutions.

**Power Method (PM)** is one of the most classic and fundamental algorithms in numerical linear algebra. PM is also widely used in the computation of PageRank [31], Heat Kernal PageRank [50], SimRank [43], and graph propagation [43]. In our pairwise RD computation problem, we slightly modify the implementation of PM and use it to compute the groundtruth RD value. See Algorithm 1 for the pseudocode illustation. Specifically, the PM algorithm is implemented as follows: (i) Initially, we define the approximation $\overline{r}_{\mathcal{G}}(s,t) = 0$ and a vector $\mathbf{r} = \mathbf{e}_s - \mathbf{e}_t$ with iteration $k = 0$; (ii) for the $k^{th}$ iteration of the algorithm with $k \le l$, we perform $\overline{r}_{\mathcal{G}}(s,t) \leftarrow \overline{r}_{\mathcal{G}}(s,t) + \frac{\mathbf{r}(s)}{2d_s} - \frac{\mathbf{r}(t)}{2d_t}$ and $\mathbf{r} \leftarrow \left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)\mathbf{r}$, then turn to the $(k+1)^{th}$ iteration; (iii) after $l$ iterations, we stop the algorithm and output $\overline{r}_{\mathcal{G}}(s,t)$. By Equ. (1), we can easily see that $\overline{r}_{\mathcal{G}}(s,t) = \frac{1}{2}(\mathbf{e}_s - \mathbf{e}_t)^T \sum_{k=0}^{l} \mathbf{D}^{-1}\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^k (\mathbf{e}_s - \mathbf{e}_t)$ is the output after this process. Next, we prove that by setting $l = O(\kappa \log \frac{\kappa}{\epsilon})$, the approximation $\overline{r}_{\mathcal{G}}(s,t)$ is the $\epsilon$-absolute error approximation of $r_{\mathcal{G}}(s,t)$. Due to space limits, the missing proofs of this paper can be found in the full-version [52].

**Algorithm 1:** Power Method (PM) for RD computation

**Input:** $\mathcal{G}, l, s, t$
$\overline{r}_{\mathcal{G}}(s,t) = 0$, $\mathbf{r} = \mathbf{e}_s - \mathbf{e}_t$;
**for** $i = 0, 1, 2, ..., l$ **do**
  $\overline{r}_{\mathcal{G}}(s,t) \leftarrow \overline{r}_{\mathcal{G}}(s,t) + \frac{\mathbf{r}(s)}{2d_s} - \frac{\mathbf{r}(t)}{2d_t}$;
  $\mathbf{r} \leftarrow \left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)\mathbf{r}$;
**end**
**Output:** $\overline{r}_{\mathcal{G}}(s,t)$ as the approximation of $r_{\mathcal{G}}(s,t)$

Theorem 2.4. *When setting $l = 2\kappa \log \frac{\kappa}{\epsilon}$, the approximation error holds that $|\overline{r}_{\mathcal{G}}(s,t) - r_{\mathcal{G}}(s,t)| \le \epsilon$. In addition, the time complexity of* PM *is $O(lm) = O(\kappa m \log \frac{\kappa}{\epsilon})$.*

**Random Walk (RW)** is also a fundamental operator in various graph algorithms, including PageRank [23, 24, 46, 48], Heat Kernel PageRank [10, 50] and local Laplacian Solver [2]. For pairwise RD estimation, the random walk operator was first introduced in [33]. The general idea of the algorithm is as follows: based on Eq. (1), to approximate $r_{\mathcal{G}}(s,t)$, we only need to approximate $\mathbf{e}_s\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^i \mathbf{e}_t/d_s$, $\mathbf{e}_s\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^i \mathbf{e}_s/d_s$, $\mathbf{e}_t\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^i \mathbf{e}_s/d_t$, and $\mathbf{e}_t\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^i \mathbf{e}_t/d_t$ for each $i \le l$, which is exactly the probability of a length $i$ lazy random walk starts at $t$ and ends at $s$. Specifically, the length $i$ lazy random walk denotes a sequence $(v_0, v_1, ..., v_i)$ such that for each $0 \le j < i$, $v_{j+1}$ is a random neighbour of $v_j$ with probability $1/2$ and $v_{j+1} = v_j$ with probability $1/2$. The details of the pseudocode are illustrated in algorithm 2. By setting the number of random walks $n_r = O(l^2/\epsilon^2)$ with $l = O(\kappa \log \frac{\kappa}{\epsilon})$ [33], we can obtain $\hat{r}_{\mathcal{G}}(s,t)$ with the $\epsilon$-absolute error approximation. Thus the total time complexity of this algorithm is $O(n_r l^2) = \tilde{O}(\kappa^4/\epsilon^2)$ since $l = O(\kappa \log \frac{\kappa}{\epsilon})$ and $n_r = O(l^2/\epsilon^2)$. Peng et al. also proposed an advanced algorithm for the estimation of $\mathbf{e}_s\left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}\right)^i \mathbf{e}_t/d_s$ (details in Algorithm 2 in [33]). The time complexity of the advanced algorithm is $\tilde{O}(\kappa^3/\epsilon^2)$, resulting in a reduced dependency on $\kappa$. Later, Yang et al. [49] further improved this algorithm by providing a tighter analysis of the variance. They derived a refined time complexity $\tilde{O}(\kappa^3/(\epsilon^2 d^2))$ under the same error guarantee, where $d = \min\{d_s, d_t\}$. However, none of the RW based algorithms break the $\kappa^3$ bottleneck.

**Other Methods.** Despite the above two classical methods, there are also related works based on different techniques and data structures. For example, the nearly linear time Laplacian Solver [15, 17] allows us to compute the $\epsilon$-approximation of RD in $O(m \log^{1/2} n \log \frac{1}{\epsilon})$ time, independent of condition number $\kappa$. Using the Laplacian solver, one can also design $\tilde{O}(m/\epsilon^2)$ resistance distance sketches

**Algorithm 2:** Random Walk (RW) for RD computation [33]

**Input:** $\mathcal{G}, \epsilon, l, n_r, s, t$

1 **for** $i = 0, 1, 2, ..., l$ **do**

2 perform $n_r$ independent lazy random walks of length $i$ starting at $s$, let $X_{i,s}$ (resp., $X_{i,t}$) be the number of walks ends at $s$ (resp., $t$) ;

3 perform $n_r$ independent lazy random walks of length $i$ starting at $t$, let $Y_{i,s}$ (resp., $Y_{i,t}$) be the number of walks ends at $s$ (resp., $t$) ;

4 $\hat{r}_{\mathcal{G}}(s,t) += \frac{X_{i,s}}{2n_r d_s} - \frac{X_{i,t}}{2n_r d_t} + \frac{Y_{i,t}}{2n_r d_t} - \frac{Y_{i,s}}{2n_r d_s}$ ;

5 **end**

**Output:** $\hat{r}_{\mathcal{G}}(s,t)$ as the approximation of $r_{\mathcal{G}}(s,t)$

and query any single-pair RD in $\tilde{O}(\epsilon^{-2})$ time [25, 40]. However, Laplacian Solver is hard to implement in practice and its time complexity is linearly dependent on $m$. Besides, Liao et al. [20] introduced a novel landmark interpretation of RD and design several efficient RW-based algorithms for the computation of RD based on their interpretation. However, their algorithms are heuristic and lack theoretical guarantees based on $n, m, \kappa$. Recently, [13, 51] utilize bidirectional techniques to improve the efficiency of RW for single pair RD computation. Specifically, Yang et al. [51] derive an $\tilde{O}(\kappa^3/(\epsilon\sqrt{d}))$ algorithm under $\epsilon$-absolute error guarantee and an $\tilde{O}(\kappa^3\sqrt{d}/\epsilon)$ algorithm under relative error guarantee. Recently, Cui et. al [13] derive an $\tilde{O}(\kappa^{7/3}/\epsilon^{2/3})$ algorithm under $\epsilon$-absolute error guarantee. As an additional remark, while BiSPER [13] holds a theoretical advantage in asymptotic complexity, our Lanczos Push algorithm delivers superior practical performance on graphs with large condition numbers, such as road networks. The reasons could be that the basic framework of the bidirectional methods [13, 51] is PowerMethod, which typically require $O(\kappa)$ iterations to converge, whereas the basic framework of the Lanczos Push algorithm is the Lanczos iteration, which often requires less than $O(\sqrt{\kappa})$ iterations to converge in practice. The stronger convergence of the Lanczos iteration leads to better practical performance of the proposed Lanczos Push algorithm (as shown in experiments), though slightly weaker theoretical worst case guarantee.

## 3 A NOVEL LANCZOS ITERATION METHOD

We begin by introducing the classical Lanczos algorithm for computing the quadratic form $\mathbf{x}^T f(\mathbf{A})\mathbf{x}$, where $\mathbf{x} \in \mathbb{R}^n$ is a vector, $f$ is a matrix function, $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a positive semi-definite (PSD) matrix. In our case for RD computation, we choose $\mathbf{x} = \frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}$, $\mathbf{A} = \mathcal{A}$ to be the normalized adjacency matrix, and $f(\mathcal{A}) = (\mathbf{I} - \mathcal{A})^\dagger$. Now by the definition of RD, we have that:

$$\begin{aligned} r_{\mathcal{G}}(s,t) &= (\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{L}^\dagger (\mathbf{e}_s - \mathbf{e}_t) \\ &= (\mathbf{e}_s - \mathbf{e}_t)^T \mathbf{D}^{-1/2} (\mathbf{I} - \mathcal{A})^\dagger \mathbf{D}^{-1/2} (\mathbf{e}_s - \mathbf{e}_t) \\ &= (\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}})^T (\mathbf{I} - \mathcal{A})^\dagger (\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}) \\ &= \mathbf{x}^T f(\mathcal{A}) \mathbf{x}. \end{aligned} \tag{2}$$

The second equality holds beacuse $\mathbf{L}^\dagger = \mathbf{D}^{-1/2}\mathcal{L}^\dagger\mathbf{D}^{-1/2} = \mathbf{D}^{-1/2}(\mathbf{I} - \mathcal{A})^\dagger\mathbf{D}^{-1/2}$. This immediately allows us to use Lanczos iteration to compute the RD value. However, we notice that $f(\mathcal{A}) = (\mathbf{I} - \mathcal{A})^\dagger$ is the pseudo inverse of the matrix $\mathbf{I} - \mathcal{A}$, which is not a common matrix function (eg., matrix inverse, matrix exponential). So we include one of the implementation of Lanczos algorithm [29] with slight modification in Algorithm 3. The main idea of the Lanczos iteration is to project the matrix $\mathcal{A}$ onto the tridiagonal matrix $\mathbf{T}$ such that $\mathbf{T} = \mathbf{V}^T\mathcal{A}\mathbf{V}$, where $\mathbf{V} = [\mathbf{v}_1, ..., \mathbf{v}_k]$ be the $n \times k$ dimension column orthogonal matrix with $k \ll n$ that spans the $k$-dimension Krylov subspace: $\mathcal{K}_k(\mathbf{v}_1, \mathcal{A}) = span\langle \mathbf{v}_1, \mathcal{A}\mathbf{v}_1, ..., \mathcal{A}^k\mathbf{v}_1\rangle$ with $\mathbf{v}_1 = \mathbf{x}/\|\mathbf{x}\|_2 = \left(\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}\right)/\sqrt{\frac{1}{d_s} + \frac{1}{d_t}}$. First, if we have already compute the column orthogonal matrix $\mathbf{V}$, we prove that for the vectors $\mathcal{A}^i\mathbf{v}_1$ with $i \le k$, we have $\mathcal{A}^i\mathbf{v}_1 = \mathbf{V}\mathbf{T}^i\mathbf{V}^T\mathbf{v}_1$. Formally we state the following Lemma.

Lemma 3.1. *Given $\mathbf{V} = [\mathbf{v}_1, ..., \mathbf{v}_k]$ is the $n \times k$ dimension orthogonal matrix spanning the subspace $\mathcal{K}_k(\mathbf{v}_1, \mathcal{A}) = span\langle \mathbf{v}_1, \mathcal{A}\mathbf{v}_1, ..., \mathcal{A}^k\mathbf{v}_1\rangle$ and $\mathbf{T} = \mathbf{V}^T\mathcal{A}\mathbf{V}$. Then for any polynomial $p_k$ with degree at most k, we have $p_k(\mathcal{A})\mathbf{v}_1 = \mathbf{V}p_k(\mathbf{T})\mathbf{V}^T\mathbf{v}_1$.*

Next, based on Lemma 3.1 we show that $\mathbf{v}_1^T\mathbf{V}(\mathbf{I} - \mathbf{T})^{-1}\mathbf{V}^T\mathbf{v}_1$ is a good estimator of the quadratic form $\mathbf{v}_1^T(\mathbf{I} - \mathcal{A})^\dagger\mathbf{v}_1$. The proof is based on the classical analysis but we slightly modify the proof.

Lemma 3.2. *Let $\mathbf{V} \in \mathbb{R}^{n\times k}$ and $\mathbf{T} \in \mathbb{R}^{k\times k}$ defined in Lemma 3.1. Let $\mathcal{P}_k$ be the set of all polynomials with degree $\le k$. Then, we have:*

$$\begin{aligned} &|\mathbf{v}_1^T\mathbf{V}(\mathbf{I} - \mathbf{T})^{-1}\mathbf{V}^T\mathbf{v}_1 - \mathbf{v}_1^T(\mathbf{I} - \mathcal{A})^\dagger\mathbf{v}_1| \\ &\quad \le 2 \min_{p\in\mathcal{P}_k} \max_{x\in[\lambda_{min}(\mathcal{A}), \lambda_2(\mathcal{A})]} |1/(1-x) - p(x)|, \end{aligned}$$

*where $\lambda_{min}(\mathcal{A})$ and $\lambda_2(\mathcal{A})$ denotes the smallest and second largest eigenvalue of $\mathcal{A}$, respectively.*

Based on Lemma 3.2, we can now approximate the RD value by the following formula:

$$\begin{aligned} r_{\mathcal{G}}(s,t) &= (\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}})^T(\mathbf{I} - \mathcal{A})^\dagger(\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}) \\ &= \left(\frac{1}{d_s} + \frac{1}{d_t}\right)\mathbf{v}_1^T(\mathbf{I} - \mathcal{A})^\dagger\mathbf{v}_1 \\ &\approx \left(\frac{1}{d_s} + \frac{1}{d_t}\right)\mathbf{v}_1^T\mathbf{V}(\mathbf{I} - \mathbf{T})^{-1}\mathbf{V}^T\mathbf{v}_1 \\ &= \left(\frac{1}{d_s} + \frac{1}{d_t}\right)\mathbf{e}_1^T(\mathbf{I} - \mathbf{T})^{-1}\mathbf{e}_1. \end{aligned} \tag{3}$$

The second equality holds because $\mathbf{v}_1 = \left(\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}\right)/\left\|\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}\right\|_2$ and $\left\|\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}\right\|_2^2 = \frac{1}{d_s} + \frac{1}{d_t}$. The last equality holds because $\mathbf{v}_1^T\mathbf{V} = \mathbf{e}_1^T$, by $\mathbf{v}_1$ is orthogonal to $\mathbf{v}_2, ..., \mathbf{v}_k$. Now since $\mathbf{I} - \mathbf{T}$ is only a $k \times k$ dimension matrix with $k \ll n$, we can efficiently compute $\left(\frac{1}{d_s} + \frac{1}{d_t}\right)\mathbf{e}_1^T(\mathbf{I} - \mathbf{T})^{-1}\mathbf{e}_1$ in only $O(k^{2.37})$ time with fast matrix multiplication [47]. Finally, all the things remain is how to efficiently compute these orthogonal vectors $\mathbf{v}_1, ..., \mathbf{v}_k$. To reach this, in each iteration we perform the Lanczos recurrence (i.e. Line 3-7 in Algorithm 3):

$$\beta_{i+1}\mathbf{v}_{i+1} = (\mathcal{A} - \alpha_i)\mathbf{v}_i - \beta_i\mathbf{v}_{i-1}, \tag{4}$$

where $\alpha_i$ and $\beta_{i+1}$ are scalars computed in Line 4 and Line 6 in Algorithm 3. For our analysis, we present a classical result of the

**Algorithm 3:** Lanczos iteration for RD computation

**Input:** $\mathcal{G}, s, t, k$

1 $\mathbf{v}_0 = 0, \mathbf{v}_1 = \left(\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}\right) / \sqrt{\frac{1}{d_s} + \frac{1}{d_t}}, \beta_1 = 0;$

2 **for** $i = 1, 2, ..., k$ **do**

3 $\quad \mathbf{v_{i+1}} = \mathcal{A}\mathbf{v}_i - \beta_i \mathbf{v}_{i-1};$

4 $\quad \alpha_i = \langle \mathbf{v}_{i+1}, \mathbf{v}_i \rangle;$

5 $\quad \mathbf{v_{i+1}} = \mathbf{v_{i+1}} - \alpha_i \mathbf{v}_i;$

6 $\quad \beta_{i+1} = \|\mathbf{v}_{i+1}\|_2;$

7 $\quad \mathbf{v}_{i+1} = \mathbf{v}_{i+1} / \beta_{i+1};$

8 **end**

9 $\mathbf{T} = \begin{bmatrix} \alpha_1 & \beta_2 & & 0 \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_k \\ 0 & & \beta_k & \alpha_k \end{bmatrix}, \mathbf{V} = [\mathbf{v}_1, ..., \mathbf{v}_k];$

**Output:** $\hat{r}_{\mathcal{G}}(s,t) = \left(\frac{1}{d_s} + \frac{1}{d_t}\right) \mathbf{e}_1^T (\mathbf{I} - \mathbf{T})^{-1} \mathbf{e}_1$ as the approximation of $r_{\mathcal{G}}(s,t)$

output of Lanczos iteration (see eg. Claim 4.1 in [29]). This will be used in our analysis later.

Proposition 3.3. *(Output guarantee) The Lanczos iteration (Algorithm 3) computes* $\mathbf{V} \in \mathbb{R}^{n \times k}$, $\mathbf{T} \in \mathbb{R}^{k \times k}$ *and an additional vector* $\mathbf{v}_{k+1}$, *a scalar* $\beta_{k+1}$ *such that:*

$$\mathbf{AV} = \mathbf{VT} + \beta_{k+1} \mathbf{v}_{k+1} \mathbf{e}_k^T.$$

*Moreover, the column space of* $\mathbf{V}$ *spans the Krylov subspace* $\mathcal{K}_k(\mathbf{v}_1, \mathcal{A}) = span\langle \mathbf{v}_1, \mathcal{A}\mathbf{v}_1, ..., \mathcal{A}^k \mathbf{v}_1 \rangle$ *and* $\mathbf{T} = \mathbf{V}^T \mathcal{A} \mathbf{V}$.

Finally, based on Lemma 3.2 and Proposition 3.3 we prove the following Theorem which gives the error bound and time complexity of the Lanczos iteration (Algorithm 3).

Theorem 3.4. *(Error guarantee) Let* $\hat{r}_{\mathcal{G}}(s,t)$ *output by Lanczos iteration (Algorithm 3). Then the error satisfies*

$$|\hat{r}_{\mathcal{G}}(s,t) - r_{\mathcal{G}}(s,t)| \leq \epsilon.$$

*When setting the iteration number* $k = O(\sqrt{\kappa} \log \frac{\kappa}{\epsilon})$. *Furthermore, the time complexity of Algorithm 3 is* $O(km + k^{2.37}) = O(\sqrt{\kappa} m \log \frac{\kappa}{\epsilon})$ *when* $k^{1.37} \leq m$.

Therefore, according to Theorem 3.4 we provided that the approximation $\hat{r}_{\mathcal{G}}(s,t)$ satisfies the $\epsilon$-absolute error guarantee after $k = \sqrt{\kappa} \log \frac{\kappa}{\epsilon}$ iterations of Lanczos recurrence. Recall that the iteration number of PM is required to be $l = \kappa \log \frac{\kappa}{\epsilon}$, so Algorithm 3 is $\sqrt{\kappa}$-times faster than PM.

# 4 A NOVEL LANCZOS PUSH METHOD

## 4.1 The Lanczos Push Algorithm

Next, we introduce our novel push-style local algorithm, called Lanczos Push, based on a newly-developed "purning" technique. Before introducing our techniques, we first explain the high level idea of our method. Recall that the Lanczos iteration search a sequence of orthogonal vectors $\mathbf{v}_1, ..., \mathbf{v}_k$ to accelerate RD computation (i.e., only $\sqrt{\kappa}$ dependency on condition number $\kappa$, which improves upon PM by a $\sqrt{\kappa}$ factor). Now, to design a local algorithm for RD computation while maintaining the rapid convergence rate by Lanczos iteration, our hope is to search a sequence of **sparse** vectors $\hat{\mathbf{v}}_1, ..., \hat{\mathbf{v}}_k$ that is similar to $\mathbf{v}_1, ..., \mathbf{v}_k$ (though they can be no longer orthogonal). See Fig. 1 as an illustration.

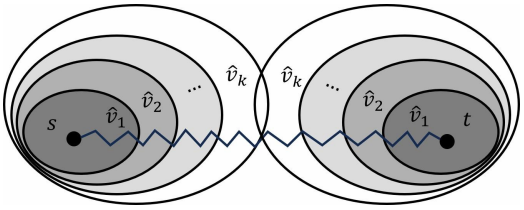



**Figure 1: High level idea of our Lanczos Push algorithm for computing $r_{\mathcal{G}}(s,t)$; $\hat{\mathbf{v}}_1, ..., \hat{\mathbf{v}}_k$ are sparse vectors.**

To specify our method, let's closely examine the time complexity of Lanczos iteration (Algorithm 3). Note that in Line 3 of Algorithm 3, we perform the matrix-vector multiplication $\mathcal{A}\mathbf{v}_i$, which requires $O(m)$ operations. In Line 4-7 of Algorithm 3, we invoke the inner product and addition/subtraction operations on $\mathbf{v}_i$, which requires $O(n)$ operations. Therefore, to transform Algorithm 3 to a local algorithm, there are two main challenges: (i) "locally" implement the matrix-vector multiplication $\mathcal{A}\mathbf{v}_i$ in tach iteration. That is, we should approximate $\mathcal{A}\mathbf{v}_i$ by only invoking small portion of graph. (ii) maintaining $\mathbf{v}_i$ a sparse vector in each iteration. To address these challenges, we propose the Lanczos Push algorithm to locally approximate these vectors. To reach this end, first we define our sparse approximation of $\mathbf{v}_i$ and $\mathcal{A}\mathbf{v}_i$.

*Definition 4.1.* We define $AMV(\mathcal{A}, \hat{\mathbf{v}}_i) \in \mathbb{R}^n$ as the vector that approximates $\mathcal{A}\hat{\mathbf{v}}_i$. Specifically, for each node $v \in \mathcal{V}$, we define $AMV(\mathcal{A}, \hat{\mathbf{v}}_i)(v) = \sum_{u \in \mathcal{N}(v)} \frac{\hat{\mathbf{v}}_i(u)}{\sqrt{d_u d_v}} \mathbb{I}_{|\hat{\mathbf{v}}_i(u)| > \epsilon \sqrt{d_u d_v}}$. Where we denote $\mathbb{I}_{|\hat{\mathbf{v}}_i(u)| > \epsilon \sqrt{d_u d_v}}$ as the indicator that takes value 1 if and only if $|\hat{\mathbf{v}}_i(u)| > \epsilon \sqrt{d_u d_v}$.

*Definition 4.2.* We define $\hat{\mathbf{v}}_i|_{S_i} \in \mathbb{R}^n$ as the vector $\hat{\mathbf{v}}_i$ constraint on the subset $S_i \subset \mathcal{V}$. Specifically, $\hat{\mathbf{v}}_i|_{S_i}(u) = \hat{\mathbf{v}}_i(u)$ for $u \in S_i$, and $\hat{\mathbf{v}}_i|_{S_i}(u) = 0$ otherwise. Where $S_i = \{u \in \mathcal{V} : |\hat{\mathbf{v}}_i(u)| > \epsilon d_u\}$.

Based on Definition 4.1 and 4.2, we propose the following subset Lanczos recurrence:

$$\hat{\beta}_{i+1} \hat{\mathbf{v}}_{i+1} = AMV(\mathcal{A}, \hat{\mathbf{v}}_i) - \hat{\alpha}_i \hat{\mathbf{v}}_i|_{S_i} - \hat{\beta}_i \hat{\mathbf{v}}_{i-1}|_{S_{i-1}}. \tag{5}$$

In Eq. (5), $AMV(\mathcal{A}, \hat{\mathbf{v}}_i)$ is the sparse approximation of $\mathcal{A}\hat{\mathbf{v}}_i$ by Definition 4.1, $\hat{\mathbf{v}}_i|_{S_i}$ is the sparse approximation of $\hat{\mathbf{v}}_i$ by Definition 4.2, $\hat{\alpha}_i$ and $\hat{\beta}_{i+1}$ are scalars computed in the same way as Line 4 and Line 6 in Algorithm 3. Putting these things together, we provide the pseudocode in Algorithm 4. The idea of Algorithm 4 is simple: to make Lanczos iteration implemented locally on graphs, we perform the subset Lanczos iteration (Eq. (5)). In each iteration, we perform the following two key operations: (i) For the approximation of the matrix-vector multiplication $\mathcal{A}\hat{\mathbf{v}}_i$, we only search on the node $u$ such that $\hat{\mathbf{v}}_i(u)$ non-zero. Next, for the neighbors $v \in \mathcal{N}(u)$, we only add the scores along the edge $(u,v) \in \mathcal{E}$ with $|\hat{\mathbf{v}}_i(u)| > \epsilon \sqrt{d_u d_v}$, see Line 4-8 in Algorithm 4. We denote $AMV(\mathcal{A}, \hat{\mathbf{v}}_i)$ as the approximation of $\mathcal{A}\hat{\mathbf{v}}_i$ after this operation. (ii) We select a subset of important nodes $S_i = \{u \in \mathcal{V} : |\hat{\mathbf{v}}_i(u)| > \epsilon d_u\}$, and we perform addition/subtraction only constraint on subset $S_i$, see Line 9-11, Line 13-15 of Algorithm 4. These operations maintain the sparsity of the computation.

**Algorithm 4:** Lanczos Push for RD computation

**Input:** $\mathcal{G}, s, t, k, \epsilon$

1 $\hat{\mathbf{v}}_0 = 0, \hat{\mathbf{v}}_1 = \left(\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}\right) / \sqrt{\frac{1}{d_s} + \frac{1}{d_t}}, \hat{\beta}_1 = 0;$

2 **for** $i = 1, 2, ..., k$ **do**

3 $\quad S_i = \{u \in \mathcal{V} : |\hat{\mathbf{v}}_i(u)| > \epsilon d_u\};$

4 $\quad$ **for** *node u such that* $\hat{\mathbf{v}}_i(u) \neq 0$ **do**

5 $\quad\quad$ **for** $v \in \mathcal{N}(u)$ *with* $|\hat{\mathbf{v}}_i(u)| > \epsilon\sqrt{d_u d_v}$ **do**

6 $\quad\quad\quad \hat{\mathbf{v}}_{i+1}(v) = \hat{\mathbf{v}}_{i+1}(v) + \frac{1}{\sqrt{d_u d_v}}\hat{\mathbf{v}}_i(u);$

7 $\quad\quad$ **end**

8 $\quad$ **end**

9 $\quad$ **for** $u \in S_{i-1}$ **do**

10 $\quad\quad \hat{\mathbf{v}}_{i+1}(u) = \hat{\mathbf{v}}_{i+1}(u) - \hat{\beta}_i \hat{\mathbf{v}}_{i-1}(u);$

11 $\quad$ **end**

12 $\quad \hat{\alpha}_i = \langle \hat{\mathbf{v}}_{i+1}, \hat{\mathbf{v}}_i \rangle;$

13 $\quad$ **for** $u \in S_i$ **do**

14 $\quad\quad \hat{\mathbf{v}}_{i+1}(u) = \hat{\mathbf{v}}_{i+1}(u) - \hat{\alpha}_i \hat{\mathbf{v}}_i(u);$

15 $\quad$ **end**

16 $\quad \hat{\beta}_{i+1} = \|\hat{\mathbf{v}}_{i+1}\|_2;$

17 $\quad \hat{\mathbf{v}}_{i+1} = \hat{\mathbf{v}}_{i+1} / \hat{\beta}_{i+1};$

18 **end**

19 $\hat{\mathbf{T}} = \begin{bmatrix} \hat{\alpha}_1 & \hat{\beta}_2 & & 0 \\ \hat{\beta}_2 & \hat{\alpha}_2 & \ddots & \\ & \ddots & \ddots & \hat{\beta}_k \\ 0 & & \hat{\beta}_k & \hat{\alpha}_k \end{bmatrix}, \hat{\mathbf{V}} = [\hat{\mathbf{v}}_1, ..., \hat{\mathbf{v}}_k];$

**Output:** $r'_{\mathcal{G}}(s,t) = \left(\frac{1}{d_s} + \frac{1}{d_t}\right) \hat{\mathbf{v}}_1^T \hat{\mathbf{V}} (\mathbf{I} - \hat{\mathbf{T}})^{-1} \mathbf{e}_1$ as the approximation of $r_{\mathcal{G}}(s,t)$

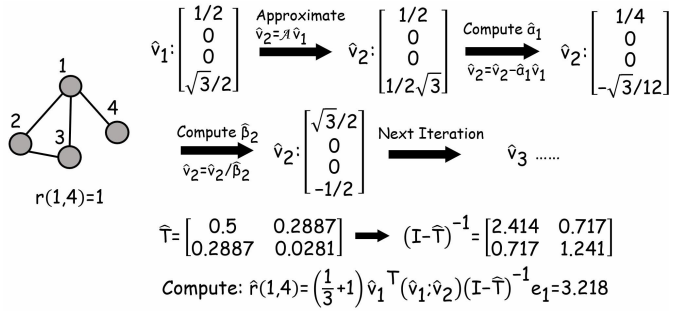



**Figure 2: A running example of our Lanczos Push algorithm**

**A running example.** To illustrate the proposed Lanczos Push algorithm, we provide a running example here. We consider a toy graph with vertex set $\mathcal{V} = \{1, 2, 3, 4\}$ and edge set $\mathcal{E} = \{(1,2), (1,3), (1,4), (2,3)\}$, as illustrated in Fig. 2. We focus on computing the resistance distance (RD) between nodes index 1 and 4, where the exact value is known to be $r(1,4) = 1$ by the definition of resistance. For demonstration purposes, we set the algorithm parameters to $k = 2$ and $\epsilon = 0.25$ for Lanczos Push algorithm. We first initialize $\hat{\mathbf{v}}_1 = \left(\frac{\mathbf{e}_s}{\sqrt{d_s}} - \frac{\mathbf{e}_t}{\sqrt{d_t}}\right) / \sqrt{\frac{1}{d_s} + \frac{1}{d_t}} = [1/2, 0, 0, \sqrt{3}/2]^T$ and set $\hat{\beta}_1 = 0$ by Line 1 of Algorithm 4. We set $S_1 = \{1, 4\}$ by Line 3. Next, we invoke Lines 4-8 to approximate the matrix-vector multiplication $\mathcal{A}\hat{\mathbf{v}}_1$. Notably, the computations along the edge $(1,2)$ and $(1,3)$ are prevented because $|\hat{\mathbf{v}}_1(u)|/\sqrt{d_u d_v} = 0.5/\sqrt{6} < 0.25 = \epsilon$ for $(u,v) = (1,2)$ and $(1,3)$. After the computation of Line 4-8, we obtain $\hat{\mathbf{v}}_2 = [1/2, 0, 0, 1/2\sqrt{3}]^T$. Then, we skip the computation from Lines 9-11 because we set $\hat{\beta}_1 = 0$ at initial step. Next, we compute $\hat{\alpha}_1 = \langle \hat{\mathbf{v}}_2, \hat{\mathbf{v}}_1 \rangle = 1/2$ by Line 12. We update $\hat{\mathbf{v}}_2 = \hat{\mathbf{v}}_2 - \hat{\alpha}_1 \hat{\mathbf{v}}_1|_{S_1} = [1/4, 0, 0, -\sqrt{3}/12]^T$ by Line 13-15. We compute $\hat{\beta}_2 = \|\hat{\mathbf{v}}_2\|_2 = 1/2\sqrt{3}$ by Line 16, and we finally update $\hat{\mathbf{v}}_2 = \hat{\mathbf{v}}_2/\beta_2 = [\sqrt{3}/2, 0, 0, -1/2]^T$ by Line 17. Subsequently, we turn to the next iteration. For iteration $i = 2$, we can similarly compute $\hat{\alpha}_2 = 0.0281$; we omit the details here because all the routines maintain the same. Finally, we compute the approximate RD value by $\hat{r}(1,4) = \left(\frac{1}{3} + 1\right) \hat{\mathbf{v}}_1^T [\hat{\mathbf{v}}_1; \hat{\mathbf{v}}_2] (\mathbf{I} - \hat{\mathbf{T}})^{-1} \mathbf{e}_1 = 3.218$. Note that this result seems inaccurate mainly because we set a large threshold $\epsilon = 0.25$ for better illustration. In real-world graphs, when setting the threshold $\epsilon$ smaller, our Lanczos Push algorithm successfully reduce computation costs while does not produce too much additional error. The theoretical guarantees are provided in the next subsection.

## 4.2 Analysis of the Lanczos Push Algorithm

Here, we prove that our Lanczos Push algorithm (Algorithm 4) does not introduce significant errors compared to the Lanczos iteration, while drastically reducing the runtime. However, due to the complexity of the proofs, we provide only an outline here, the details are left in the full version [52]. Overall, the theoretical analysis is devided into two steps: (i) the error bound of Algorithm 4, (ii) the local time complexity of Algorithm 4. For simplicity, we denote $C_1 = \max_{u=s,t; i \leq k} \|T_i(\mathbf{P})\mathbf{e}_u\|_1$, where $T_i(x)$ is the first kind of Chebyshev polynomials (definition see Eq. 13 in Appendix). We denote $C_2 = \max_{i \leq k} (\|\hat{\mathbf{v}}_i\|_1 + \|\mathcal{A}\hat{\mathbf{v}}_i^+\|_1 + \|\mathcal{A}\hat{\mathbf{v}}_i^-\|_1)$, where $\hat{\mathbf{v}}_i^+, \hat{\mathbf{v}}_i^-$ are the positive parts and negative parts of $\hat{\mathbf{v}}_i$, respectively.

Theorem 4.3. *Given $\epsilon' \in (0,1)$ , setting iteration number $k = \sqrt{\kappa} \log \frac{\kappa}{\epsilon'}$, parameter $\epsilon = \tilde{\Omega}(\frac{\epsilon'\sqrt{d}}{\kappa^{2.25} C_1})$, the error between the approximation $r'_{\mathcal{G}}(s,t)$ output by Algorithm 4 and accurate RD value $r_{\mathcal{G}}(s,t)$ can be bounded by:*

$$|r_{\mathcal{G}}(s,t) - r'_{\mathcal{G}}(s,t)| \leq \epsilon'.$$

Theorem 4.4. *The total operations performed by Algorithm 4 can be bounded by $O(\frac{1}{\epsilon} \sum_{i \leq k} (\|\hat{\mathbf{v}}_i\|_1 + \|\mathcal{A}\hat{\mathbf{v}}_i^+\|_1 + \|\mathcal{A}\hat{\mathbf{v}}_i^-\|_1) + k^{2.37})$. When setting $k = \sqrt{\kappa} \log \frac{\kappa}{\epsilon'}, \epsilon = \tilde{\Omega}(\frac{\epsilon'\sqrt{d}}{\kappa^{2.25} C_1})$, this is $\tilde{O}(\kappa^{2.75} C_1 C_2 / (\epsilon'\sqrt{d}))$.*

The above theoretical analysis of the Lanczos Push algorithm is based on a simple assumption.

Assumption 1. *We assume $\lambda(\hat{\mathbf{T}}) \subset [\lambda_n(\mathcal{A}), \lambda_2(\mathcal{A})]$.*

We notice that the requirement of Assumption 1 is mild. In fact, from the proof of Lemma 3.2, we always have $\lambda(\mathbf{T}) \subset [\lambda_n(\mathcal{A}), \lambda_2(\mathcal{A})]$ for the exact Lanczos iteration (i.e., Algorithm 3 , also Lanczos Push algorithm when setting $\epsilon = 0$). We also show that this assumption holds in various datasets in our experiments (details in Sec. 6).

**Understanding the theoretical result.** At a first glance, the result of Theorem 4.3 and Theorem 4.4 maybe not clear enough due to the existence of $C_1$ and $C_2$, we hereby provide some further discussions. First, we provide a simpler version of Theorem 4.4 by proving the upper bound of $C_1$ and $C_2$.

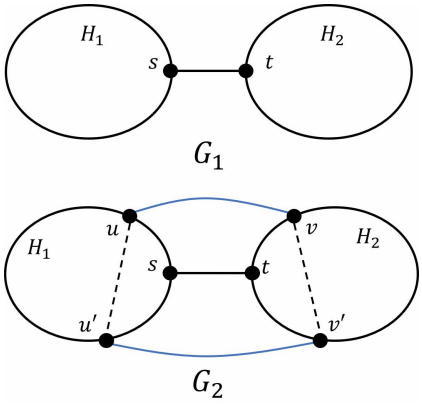



**Figure 3: The construction of the lower bound**

Corollary 4.5. *The runtime of Algorithm 4 is* $\tilde{O}(\kappa^{2.75}\sqrt{nm}/(\epsilon\sqrt{d}))$ *to compute the* $\epsilon$*-absolute error approximation of* $r_{\mathcal{G}}(s,t)$*. Specifically,* $C_1 \leq \sqrt{m}$ *and* $C_2 \leq 3\sqrt{n}$*.*

Corollary 4.5 establishes that Lanczos Push achieves sublinear runtime dependence on graph size $m$ while maintaining sub-cubic dependence on the condition number $\kappa$. This result confirms our key intuition: the algorithm preserves faster $\kappa$-dependent convergence rate while operating in sublinear time in terms of $m$. Furthermore, our analysis provides the first theoretical justification for implementing Lanczos in a local manner. However, our experimental results consistently outperform the worst-case bounds in Corollary 4.5, suggesting these theoretical guarantees may not be tight. In particular, the empirical behavior of $C_1$ and $C_2$ demonstrates better scaling than simply using the Cauchy-Schwarz inequality in our proof. Moreover, our experiments (Exp-9, Section 6) reveal an intriguing pattern: $C_1$ and $C_2$ tend to grow slower on poor expansion graphs (e.g., road networks) and faster on good expansion graphs (e.g., social networks). This finding provides further evidence that Lanczos Push is particularly well-suited for poor expansion graphs, and we hope that there will be sharper theoretical bounds or further findings to explain this phenomenon in future studies.

## 5 LOWER BOUNDS

In this section, we establish the lower bound of $\kappa$ dependency for the approximation of $r_{\mathcal{G}}(s,t)$. Our proof is based on the former result proposed by Cai et al. [6]. Below, we briefly introduce the result established in [6]. First, we consider two $d$-regular expander graphs $H_1, H_2$ (that is, $H_1, H_2$ has condition number $\kappa = O(1)$). Specifically, we choose $H_1$, $H_2$ to be two copies of Ramanujan graph [28] on $n/2$ vertices (three regular graph with $\kappa = O(1)$). Next, we connect a node $s \in H_1$ and $t \in H_2$ by an edge $(s,t)$, generating the graph $\mathcal{G}_1$. Clearly, $r_{\mathcal{G}_1}(s,t) = 1$. For the construction of $\mathcal{G}_2$, we select a random edge $(u,u') \in H_1$ and $(v,v') \in H_2$. we remove the edges $(u,u')$ and $(v,v')$, and add two new edges $(u,v)$ and $(u',v')$. Then, [6] prove that after this operation, the resulting graph $\mathcal{G}_2$ has the effective resistance $r_{\mathcal{G}_2}(s,t) \leq 0.99$. However, any algorithm that distinguishes $\mathcal{G}_1$ and $\mathcal{G}_2$ with positive probability requires at least $\Omega(n)$ queries. For better understanding, we summarize this result into the following theorem.

Theorem 5.1. *(Theorem 1.1 in [6]) If* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *are constructed in the manner described above, then we have* $|r_{\mathcal{G}_1}(s,t) - r_{\mathcal{G}_2}(s,t)| \geq 0.01$*. But, any (randomized) algorithm that distinguish* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *with probability* $\geq 0.6$ *requires at least* $\Omega(n)$ *queries.*

**Table 2: Statistics of datasets**

| Datasets | $n$ | $m$ | $m/n$ | condition number $\kappa$ |
|---|---|---|---|---|
| Dblp | 317,080 | 1,049,866 | 3.31 | 717.62 |
| Youtube | 1,134,890 | 2,987,624 | 2.63 | 907.85 |
| LiveJournal | 4,846,609 | 42,851,237 | 8.84 | 2785.51 |
| Orkut | 3,072,626 | 11,718,5083 | 38.13 | 381.46 |
| powergrid | 4,941 | 6,594 | 1.33 | 7299.27 |
| RoadNet-CA | 1,965,206 | 2,766,607 | 1.41 | 64516.12 |
| RoadNet-PA | 1,088,092 | 1,541,898 | 1.42 | 24390.24 |
| RoadNet-TX | 1,379,917 | 1,921,660 | 1.39 | 28985.5 |
| ER | $10^2 \sim 10^6$ | $n \log n$ | $\log n$ | $O(1)$ |
| BA | $10^2 \sim 10^6$ | $n \log n$ | $\log n$ | $O(1)$ |

Based on Theorem 5.1, we can prove that the condition number of $\mathcal{G}_1$ is $\kappa = \Theta(n)$, which is stated in the following lemma.

Lemma 5.2. *If* $\mathcal{G}_1$ *is constructed in the manner described above, we have* $\kappa(\mathcal{G}_1) = \Theta(n)$*.*

Combining with Theorem 5.1 and Lemma 5.2, we can prove that given any $\epsilon < 0.01$, any algorithm computes the $\epsilon$-approximate RD value $r_{\mathcal{G}}(s,t)$ requires at least $\Omega(\kappa)$ queries.

Theorem 5.3. *Any algorithm that approximates* $r_{\mathcal{G}}(s,t)$ *with absolute error* $\epsilon \leq 0.01$ *with success probability* $\geq 0.6$ *requires* $\Omega(\kappa)$ *queries.*

This indicates that the time complexity of the local algorithms is indeed relevant to the condition number $\kappa$, which has not been emphasized in previous works for local RD computation. We anticipate this $\Omega(\kappa)$ is not tight. Unfortunately, to the best of our knowledge, there are no better constructions, thus we leave how to match the upper and lower bounds as an interesting future direction.

## 6 EXPERIMENTAL EVALUATION

### 6.1 Experimental Setup

We use 8 publicly available datasets[4] of varying sizes (Table 2), including 4 social networks (Dblp, Youtube, LiveJournal, Orkut) which serve as standard benchmarks for various graph algorithms [20, 43, 44, 48, 49] and 4 datasets that represent hard cases (due to large $\kappa$) for previous RD algorithms [22]. Among them, RoadNet-CA, RoadNet-PA, and RoadNet-TX are road networks, and powergrid is an infrastructure network, statistically similar to road networks. Additionally, we artificially generate the classic Erdős-Rényi (ER) and Barabási-Albert (BA) graphs with size $n$ varying from $10^2 \sim 10^6$ and $m = n \log n$ for the scalability testing. To compute the condition number of these datasets, we use the standard Power Method to compute the second largest eigenvalue $\mu_2$ of the normalized adjacency matrix, and by definition, the condition number is $\kappa = \frac{2}{1-\mu_2}$. We set the threshold of the Power Method to $10^{-9}$ to make the approximation of $\mu_2$ sufficiently accurate. To evaluate the approximation errors of different algorithms, we compute *ground-truth* RD value using Power Method (PM) with a sufficiently large truncation

[4]All the 8 datasets can be downloaded from http://snap.stanford.edu/ and http://konect.cc/networks/

step $N = 20000$. We compare our proposed algorithms, Lanczos iteration (Lz) and Lanczos Push (LzPush) with 6 state-of-the-art (SOTA) baselines: PM [49], GEER [49], Push [20], BiPush [20], also two very recent studies BiSPER [13] and FastRD [25]. Since the algorithms proposed in [20, 49] outperform the algorithms proposed in [33] in all datasets, we no longer compare our algorithms with the algorithms proposed in [33]. We randomly generate 50 source nodes and 50 sink nodes as query sets and report the average performance across them for different algorithms. By Definition 2.2, we use absolute error (denoted as Absolute Err) to evaluate the estimation error of different algorithms. We conduct all experiments on a Linux 20.04 server with an Intel 2.0 GHz CPU and 128GB memory. All algorithms are implemented in C++ and compiled using GCC 9.3.0 with -O3 optimization.

## 6.2 Overall Empirical Results

**Exp-1: Query time of different algorithms on unweighted graphs.** In this experiment, we compare the proposed algorithms, Lz and LzPush, with 6 SOTA algorithms proposed in the previous studies [13, 20, 25, 49]: PM, GEER, Push, BiPush, BiSPER and FastRD. For PM, GEER, Push, BiPush, following [20, 49] we vary the parameter $\epsilon$ from $10^{-1}$ to $10^{-5}$. For BiSPER and FastRD, following [13, 25] we vary $\epsilon$ from 0.9 to $10^{-3}$. For FastRD, since its storage is unacceptable for million-size graphs under high precision computation, so we only compare FastRD under large $\epsilon$ for large datasets, e.g., $\epsilon = 0.9, 0.5, 0.1$, following the setting as [25]. For the datasets with small condition number (i.e. the four social networks), we set the parameters for our algorithms as follows for fair comparison: for Lz we vary the iteration step $k = [5, 10, 15, 20, 30]$; for LzPush we vary $\epsilon = [0.1, 0.05, 0.01, 0.005, 0.001]$ and fix $k = 20$. Fig. 4 reports the query time of different algorithms for RD computation, under the Absolute Err metric. We make the following observations: (i) The proposed algorithm LzPush is the most efficient and is 5× to 10× faster than all the other competitors across all datasets. (ii) Since the time complexity of LzPush is independent of the number of vertices and edges, for the datasets of varying sizes, we observe that the performance of LzPush remains almost the same. This is consistent with our theoretical analysis (LzPush is a local algorithm). On the other hand, the performance of Lz and PM is dependent on $m$, with their query time increasing on larger datasets (e.g., LiveJournal, Orkut).

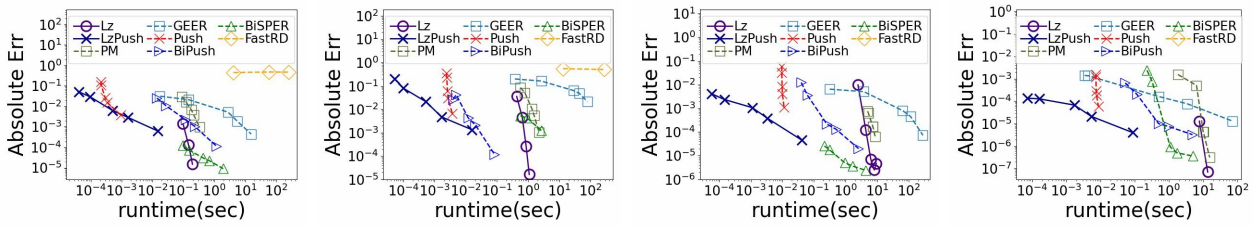
For datasets with large condition numbers (i.e. the three road networks and powergrid), we set the parameters as follows for fair comparison: for Lz we vary the iteration step $k = [20, 40, 60, 80, 100]$; for LzPush we vary $\epsilon = [10^{-3}, 5 * 10^{-4}, 10^{-4}, 5 * 10^{-5}, 10^{-5}]$ and fix $k = 100$. Fig. 5 reports the query time of different algorithms for RD computation under the Absolute Err metric. We make the following observations: (i) Our proposed algorithm LzPush, is still the most efficient, while all the previous local algorithms (GEER, Push, BiPush, BiSPER) fail to quickly output high-quality results (e.g., Absolute Err $< 10^{-1}$) on three road networks. The reasons are as follows: Since the condition number $\kappa$ of the road networks is large, the random walk sampling based method requires setting a very large truncation step $l$ and very large sampling times $n_r = \tilde{O}(l^2/\epsilon^2)$. This makes GEER inefficient. Similar observation also holds for BiSPER, though the bidirectional techniques improve the efficiency of random walks. For Push and BiPush, these algorithms are heuristic, and their time complexity is related to hitting time $h(s, v) + h(t, v)$, where $v$ is a pre-selected landmark node. However, on road networks $h(s, v)$ can be $O(n)$, this makes Push and BiPush very inefficient, especially for high precision computation. Most notably, on three road networks, our LzPush algorithm is over 1000× faster than PM and 50× faster than Lz. (ii) The performance of LzPush is similar across the four datasets, while the performance of Lz and PM is related to $m$. Specifically, Lz and PM are faster on small graph powergrid, but slower on three large road networks. This is consistent with the results for the social networks.

**Exp-2: Query time of different algorithms on weighted graphs.** In this experiment, we evaluate our proposed algorithms against existing baselines on weighted graphs. Following the common practice in prior work [45], we generate weighted versions of unweighted graphs by assigning each edge $e \in \mathcal{E}$ a weight equal to the number of triangles containing $e$[5]. Fig. 6 shows the results on Dblp and LiveJournal datasets. Similar results can also be observed on the other datasets. As shown in Fig. 6, the results demonstrate consistent behavior with our unweighted graph experiments: our LzPush algorithm remains the most efficient algorithm among all compared methods.

**Exp-3: Scalability testing on synthetic graphs.** In this experiment, we compare the scalability between our proposed Lz and LzPush algorithms. We use the classic ER and BA graphs with varying $n$ from $10^2$ to $10^6$ and $m = n \log n$. We set the iteration number $k = 20$, threshold $\epsilon = 0.005$, and compare the runtime between Lz and LzPush. Fig. 7 shows the result. As can be seen, the runtime of Lz grows almost linear with respect to the graph size $n$, while the runtime of LzPush is sublinear to the growth of $n$. These results further indicate that Lz is a nearly linear global algorithm and LzPush is a local algorithm, which is consistent with our theoretical analysis. Thus, these findings provide strong evidence that the proposed Lz and LzPush have excellent scalability over massive graphs.

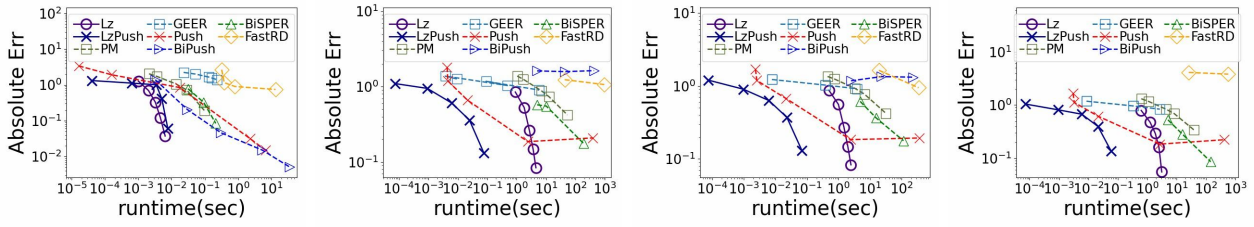
**Exp-4: Memory usage testing on various datasets.** In this experiment, we evaluate the memory usage of the proposed algorithms Lz and LzPush with PM under both low-precision case (Fig. 8 (a)) and high-precision case (Fig. 8 (b)). Specifically, for the low-precision case, we set $k = 5, \epsilon = 10^{-1}$ for social networks and $k = 20, \epsilon = 10^{-3}$ for road networks; for the high-precision case, we set $k = 30, \epsilon = 10^{-3}$ for social networks and $k = 100, \epsilon = 10^{-5}$ for road networks. From Fig. 8, we have the following observations: (1) Both Lz and LzPush exhibit linear memory dependence on graph size $n$, comparable to PM's $3n$ requirement (storing three vectors), with LzPush showing marginally higher usage than Lz; The memory usage of LzPush is slightly higher than Lz. (2) Memory consumption remains nearly identical across different precision settings, as our Lz implementation only maintains three vectors $(\mathbf{v}_{i-1}, \mathbf{v}_i, \mathbf{v}_{i+1})$ per iteration rather than the full matrix $\mathbf{V} = [\mathbf{v}_0, ..., \mathbf{v}_k]$. The modest additional memory requirement of LzPush stems from storing the candidate set $S_i$ and non-zero entries of $\mathbf{v}_i$ during iterations, though the overall space complexity remains $O(n)$. Consequently, our algorithms maintain memory efficiency even for large-scale

[5]If $e$ is not contained in any triangle, we set $w(e) = 1$ to ensure the network is connected.



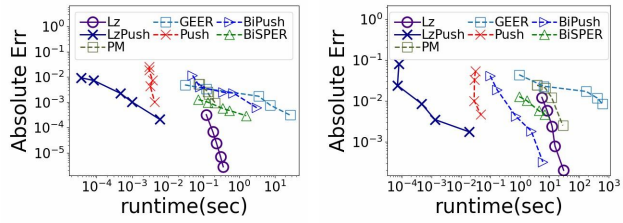

(a) Dblp (b) Youtube (c) LiveJournal (d) Orkut

**Figure 4: Query time of different algorithms for graphs with small condition number $\kappa$**



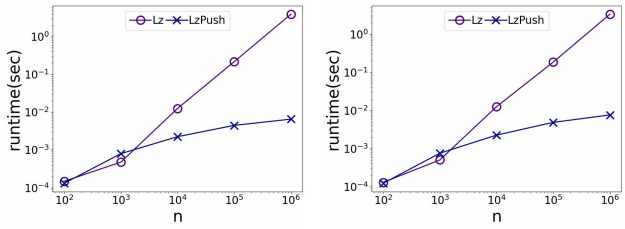

(a) powergrid (b) RoadNet-CA (c) RoadNet-PA (d) RoadNet-TX

**Figure 5: Query time of different algorithms for graphs with large condition number $\kappa$**




(a) weight-Dblp (b) weight-LiveJournal

**Figure 6: Query time of different algorithms for weighted graphs**

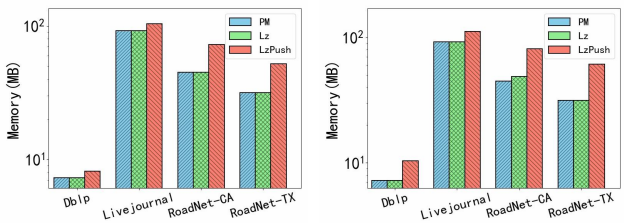



(a) ER (b) BA

**Figure 7: Scalability testing on synthetic graphs**

datasets, as their $O(n)$ storage requirement is dominated by the $O(m)$ space needed for the graph data itself.

**Exp-5: Stability testing of Lz and LzPush under high precision.** In this experiment, we examines the error stability of our proposed Lz and LzPush algorithms. Specifically, we vary the iteration number $k$ of Lz and LzPush from 5 to 150, and set $\epsilon_1 = 10^{-3}$, $\epsilon_2 = 10^{-5}$,

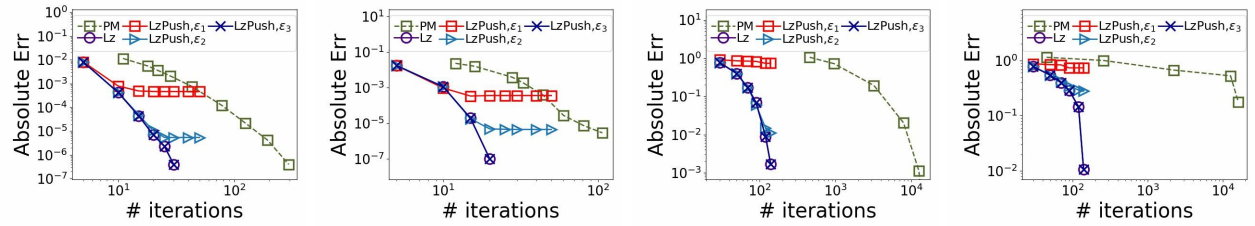



(a) low precision (b) high precision

**Figure 8: Memory usage testing on different datasets**

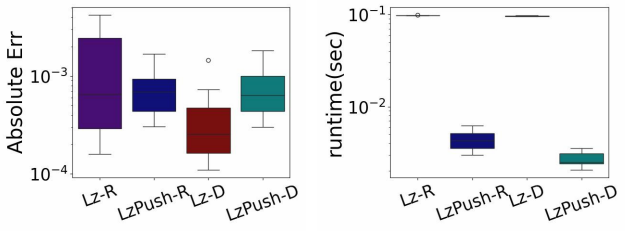
$\epsilon_3 = 10^{-7}$ for LzPush to match different absolute errors (See Fig. 9). The results demonstrate that with sufficiently large $k$ and small $\epsilon$, both Lanczos algorithms achieve high-precision solutions while converging $\sqrt{\kappa}$ times faster than PM. [6] As a result, Lz and LzPush are both stable under high precision computation. Our findings align with previous studies [27, 29]: Lanczos methods remain numerically stable for matrix function computation. In our case, the stability for RD computation stems from this phenomenon, since $\mathbf{L}^{\dagger}(\mathbf{e}_s - \mathbf{e}_t)$ is matrix function. As a result, our algorithms do not suffer unstable problem for high precision computation. Moreover, there is an additional observation that when $k$ increases, the absolute error produced by Lz drops exponentially, but the absolute error produced by LzPush does not change drastically under $\epsilon = 10^{-3}$ and $10^{-5}$. These results suggest that the efficiency of Lz mainly

[6] Comparing with previous studies [13, 20, 22, 33, 49], the $10^{-7}$ absolute error for social networks and $10^{-3}$ absolute error for road networks is already very high precision for RD computation. This is beacuse PM requires over $10^3$ iterations for social networks and $10^4$ iterations for road networks to reach this error guarantee for the ground-truth value computation.



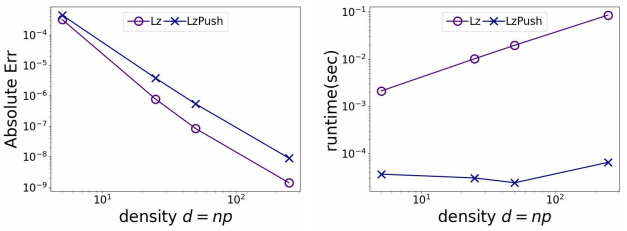

(a) Dblp (b) LiveJournal (c) powergrid (d) RoadNet-CA

**Figure 9: Stability testing of Lz and LzPush with varying iteration number $k$ and $\epsilon$**




(a) Dblp, Absolute Err (b) Dblp, runtime

**Figure 10: The performance of Lz and LzPush with different query node select strategies. Lz − R and LzPush − R denote Lz and LzPush selecting source node $s$ and sink node $t$ uniformly, respectively. Lz − D and LzPush − D represent Lz and LzPush selecting source node $s$ and sink node $t$ with the highest degree, respectively.**




(a) ER, Absolute Err (b) ER, runtime

**Figure 11: The performance of Lz and LzPush on $\mathbb{G}(n, p)$ with varying density, where the density is defined as $d = np$.**

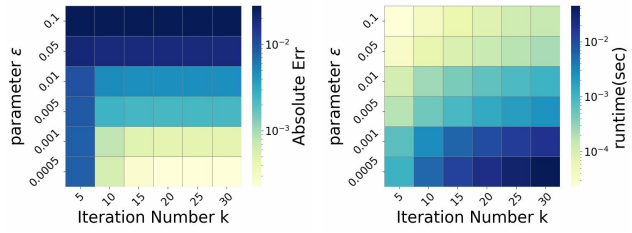
depends on $k$, but the error produced by LzPush mainly depends on $\epsilon$ for larger $k$, which is consistent with our theoretical analysis.

**Exp-6: Stability testing of Lz and LzPush under various density and node selection strategies.** This experiment evaluates the stability of Lz and LzPush in terms of: (i) various node selection strategies; (ii) density of the datasets. Specifically, for the node selection strategies, we explores two strategies for selecting source node $s$ and sink node $t$: (i) choosing the 10 nodes with the highest degrees, and (ii) randomly selecting 10 nodes from the graph. For Lz and LzPush, we set the parameter $\epsilon = 10^{-3}$ and $k = 15$. Fig. 10 presents the results using box plots to depict the distribution of query qualities and query times on Dblp. As can be seen, we have the following observations: (i) The error produced by Lz and LzPush is insensitive to the node selection strategies. (ii) For different source/sink node selection strategies, the query time and Absolute Err do not differ significantly. The overall observation is that the performance of both Lz and LzPush do not heavily depend on the strategy of the $s, t$ node selection. For the density testing, we consider ER random graph $\mathbb{G}(n, p)$ with $n = 1000$ nodes. We define the density $d = np$ and we vary $d = [5, 25, 50, 250]$ to generate datasets with different density. We set $k = 5$ and $\epsilon = 0.1$ for Lz and LzPush. Fig. 11 depicts the results. We have the following observations: the runtime of Lz grows linearly with the density of the graph, while LzPush is insensitive to the density. Moreover, the absolute value drops when the density $d$ becomes larger. This is mainly because the accurate RD value is small for dense graphs.

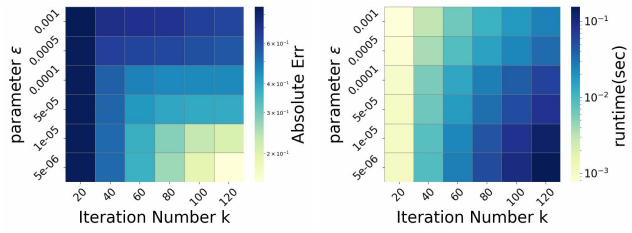
**Exp-7: The performance of LzPush with varying $k$ and $\epsilon$.** In this experiment, we explore the performance of LzPush under different setting of $k$ and $\epsilon$ on two typical datasets Dblp and RoadNet-CA. Other datasets are ignored due to they have similar trends. Fig. 12 shows the heatmap to illustrate this result. As can be seen, we have the following observations: (1) we observe each row from Fig. 12 (a) and 12 (c). The conclusion is that on both two datasets, for each fixed $\epsilon$, the absolute error produced by LzPush does not drastically drop when $k$ increases. However, for a specific pre-selected $k$ (e.g., $k = 20$ for Dblp and $k = 120$ for RoadNet-CA), the different setting of $\epsilon$ actually affect the performance of LzPush. Specifically, the absolute error drops when $\epsilon$ becomes smaller. (2) the runtime of LzPush (i.e., Fig. 12 (b) and 12 (d)) both influenced by $\epsilon$ and $k$. Specifically, larger $k$ and smaller $\epsilon$ resulting in longer runtime of LzPush. Thus, these results align with our theoretical analysis.

**Exp-8: Testing the reasonability of Assumption 1.** In this experiment, we test the we test whether Assumption 1 holds in numerical experiments, which requires that $\lambda(\hat{\mathbf{T}}) \subset [\lambda_n(\mathcal{A}), \lambda_2(\mathcal{A})]$. For clearer visualization, we examine the eigenvalues of $\mathbf{I} - \mathbf{T}$ and find that $\lambda(\mathbf{I} - \hat{\mathbf{T}}) \subset [\lambda_2(\mathcal{L}), \lambda_n(\mathcal{L})]$. For social networks we fix $k = 15$, for road networks we fix $k = 100$ while varying $\epsilon$ across different values. Fig. 13 show that shows that the eigenvalues of $\mathbf{I} - \hat{\mathbf{T}}$ always fall within the range of $\lambda(\mathcal{L})$ with the extremal eigenvalues approaching $\lambda_2(\mathcal{L})$ and $\lambda_n(\mathcal{L})$ as $\epsilon$ decreases. These results support the reasonability of Assumption 1 and suggest that the Lanczos Push algorithm may also be useful for approximating extremal eigenvalues.

**Exp-9: Testing the value of $C_1$ and $C_2$.** Recall that in our theoretical analysis, the runtime bound for LzPush is $\tilde{O}(\kappa^{2.75} C_1 C_2 / \epsilon)$ with two additional numbers $C_1 = \max_{u=s,t;i\le k} \|T_i(\mathbf{P})\mathbf{e}_u\|_1$ and

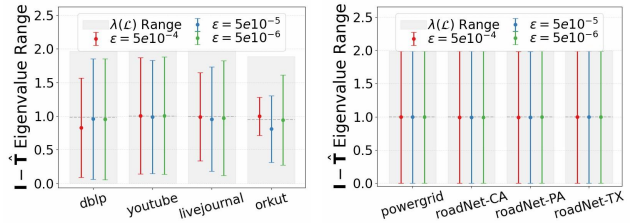



(a) Dblp,Absolute Err (b) Dblp,runtime




(c) RoadNet-CA,Absolute Err (d) RoadNet-CA,runtime

**Figure 12: The performance of LzPush with varying $\epsilon$ and $k$**



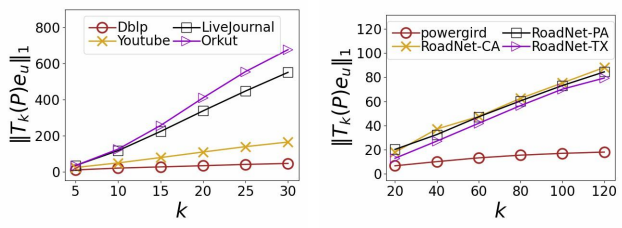

(a) Social Networks (b) Road Networks

**Figure 13: Testing the eigenvalue range of $\mathbf{I} - \hat{\mathbf{T}}$ for different datasets with varing $\epsilon$**

$C_2 = \max_{i \le k} (\|\hat{\mathbf{v}}_i\|_1 + \|\mathcal{A}\hat{\mathbf{v}}_i^+\|_1 + \|\mathcal{A}\hat{\mathbf{v}}_i^-\|_1)$. In this experiment we test the value of these two numbers with varying $k$ and $\epsilon$. Specifically, for $C_1$ we vary $k$ from 5 to 30 for social networks and 20 to 120 for road networks, with results shown in Fig. 14. We have the following observations: (i) $\|T_i(\mathbf{P})\mathbf{e}_u\|_1$ increases with both graph size and $k$, but experimentally scales better than its theoretical worst case $C_1 \le \sqrt{n}$ in Corollary 4.5. (ii) On graphs with poor expansion (road networks), the growth rate of $C_1$ is slower than the datasets with good expansion (social networks). For $C_2$, we fix $k = 15$ for social networks, $k = 100$ for road networks with varying $\epsilon$, with results shown in Fig. 15. The observation is similar to $C_1$: $\|\hat{\mathbf{v}}_i\|_1, \|\mathcal{A}\hat{\mathbf{v}}_i\|_1$ increase with larger graph size and smaller $\epsilon$, and the growth rate on road networks is slower. These observations further demonstrate that LzPush performs better on road networks comparing with previous algorithms.

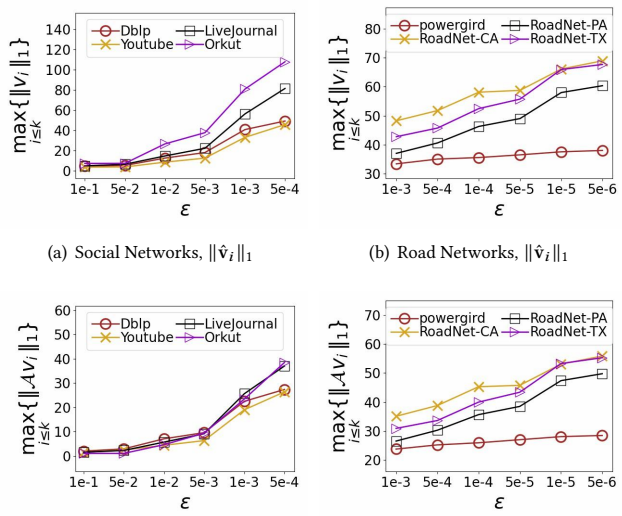
## 6.3 Case Study: Robust Routing on Road Networks

One interesting application of the proposed methods is the robust routing on road networks. One recent research [39] showing that generating a set of alternates routes via electric flows have the




(a) Social Networks (b) Road Networks

**Figure 14: Testing the value of $\max_{u=s,t;i\le k} \|T_i(\mathbf{P})\mathbf{e}_u\|_1$ with varying $k$**




(a) Social Networks, $\|\hat{\mathbf{v}}_i\|_1$ (b) Road Networks, $\|\hat{\mathbf{v}}_i\|_1$

(c) Social Networks, $\|\mathcal{A}\hat{\mathbf{v}}_i\|_1$ (d) Road Networks, $\|\mathcal{A}\hat{\mathbf{v}}_i\|_1$

**Figure 15: Testing the value of $\max_{i\le k} \|\hat{\mathbf{v}}_i\|_1$ and $\max_{i\le k} \|\mathcal{A}\hat{\mathbf{v}}_i\|_1$ with varing $\epsilon$**

advantage in terms of: (i) *Stretch.* Each alternate path does not too much worse than the shortest path. (ii) *Diversity.* The alternate paths sufficiently differs from each other. (iii) *Robustness.* Under random edge deletions, there will be a good route among the alternates with high probability. Specifically, their algorithms left the electric flow computation as a crucial subroutine, which is closely related to our RD computation problem.

*Definition 6.1.* Given two nodes $s, t$, the potential vector $\boldsymbol{\phi} \in \mathbb{R}^n$ is defined by $\boldsymbol{\phi} = \mathbf{L}^\dagger(\mathbf{e}_s - \mathbf{e}_t)$. The electric flow vector $\mathbf{f} \in \mathbb{R}^m$ is defined by $\mathbf{f}(e) = \boldsymbol{\phi}(u) - \boldsymbol{\phi}(v)$ for each edge $e = (u, v) \in \mathcal{E}$.

We show that the proposed Lanczos Algorithm can be directly generalized to compute $\boldsymbol{\phi}$ (and thus computing $\mathbf{f}$) by only a single line adjustment, see Algorithm 5. After this modification, we prove that the Lanczos Algorithm computes an accurate approximation of $\mathbf{f}$ [7].

Theorem 6.2. *Algorithm 5 computes $\hat{\mathbf{f}}$ such that $\|\mathbf{f} - \hat{\mathbf{f}}\|_1 \le \epsilon$ when setting the iteration number $k = O(\sqrt{\kappa} \log \frac{m}{\epsilon})$.*

[7] We just provide the error analysis by Lanczos Algorithm for simplicity, though the proposed Lanczos Push algorithm can also be modified to compute the electric flow.

**Algorithm 5:** Robust routing using Lanczos Iteration.

**Input:** $\mathcal{G}, s, t, k, l$

$\hat{\boldsymbol{\phi}} \leftarrow$ Lanczos Iteration $(\mathcal{G}, s, t, k)$ with final output modified by $\hat{\boldsymbol{\phi}} = \sqrt{\frac{1}{d_s} + \frac{1}{d_t}}\mathbf{D}^{-1/2}\mathbf{V}(\mathbf{I} - \mathbf{T})^{-1}\mathbf{e}_1$;

Compute $\hat{\mathbf{f}}$ by $\hat{\mathbf{f}}(e) = \hat{\boldsymbol{\phi}}(u) - \hat{\boldsymbol{\phi}}(v)$ for any $e = (u, v) \in \mathcal{E}$;

**for** $i \in [2l]$ **do**

  Find a path $\mathcal{P}$ that maximizes the minimum flow from $\hat{\mathbf{f}}$ along its edges, and remove it from $\hat{\mathbf{f}}$;

  If no $\mathcal{P}$ exists, break;

**end**

**Output:** $l$ paths with the smallest cost

**Table 3: A comparison over different methods for robust routing**

| Metric/Methods | Electric Flow | Penalty | Plateau |
|---|---|---|---|
| Stretch | 1.15 | 1.64 | 1.09 |
| Diversity | 0.93 | 0.98 | 0.51 |
| Robustness | 0.95 | 0.85 | 0.52 |
| Runtime | 0.039 | 0.036 | 0.099 |

For the next step, we use the same strategy as [39] to compute $l$ robust paths using electric flow $\hat{\mathbf{f}}$: we iteratively find a path that maximizes the minimum flow from $s$ to $t$, then remove the corresponding minimum flow value from the edges it pass through. We repeat this process $2l$ times and finally find $l$ paths with the smallest cost. We provide an running example on Boston extracted from OpenStreetMap [12]. We randomly choose two node index $s, t$. We choose $l = 10$ alternative routes and $k = 150$ for Lanczos algorithm, we compare the electric flow-based routing with SOTA shortest path routing methods: Penalty and Plateau in terms of three metrics following [39]: (i) Stretch: The average length of the alternative paths over the length of $s, t$-shortest path; (ii) Diversity: The average pairwise Jaccard similarity among all alternative paths; (iii) Robustness: The probability that there is an alternative path after random edge deletions. Specifically, we delete each edge independently with probability 0.01 in our experiments. The experimental results are reported in Table 6. As can be seen, the electric flow based routing output paths with better robustness and stretch comparing with Penalty and with better robustness and diversity comparing with Plateau. Moreover, the runtime using Lanczos algorithm is comparable with the runtime of the SOTA methods. We note that in previous studies, the main bottleneck for the electric flow-based routing is the computation of the potential $\boldsymbol{\phi}$ [39]. Our algorithm overcomes this bottleneck, enabling a fast and practical solution for alternative routing.

## 7 RELATED WORK

**RD estimation with applications.** There is a long line of research regarding the estimation of RD. The current algorithms for RD estimation can be broadly classified in two categories: (i) RD computation for multiple vertex pairs [9, 14, 16, 18, 40, 51]; (ii) RD computation for single source or single vertex pair [13, 20, 22, 33, 49]. More specifically, for the first problem, the SOTA theoretical result is the $\tilde{O}(m + (n + |S|)/\epsilon^{1.5})$ time algorithm for $\epsilon$-approximates $S$ pairs of RD value for general undirected graphs [9]. For expander graphs, the time complexity is recently reduced to $\tilde{O}(\sqrt{nm}/\epsilon + |S|)$ while maintaining the same error guarantee [51]. For the second problem, the SOTA algorithms [33, 49] estimate $r_{\mathcal{G}}(s, t)$ with $\epsilon$-absolute error guarantee in $\tilde{O}(1/\epsilon^2)$ time for a given pair of nodes $s, t$ for expander graphs. In this paper, we focus on improving the efficiency of the local algorithm on non-expander graphs (i.e., $\kappa$ is not $O(1)$), so our research has independent interest over these previous studies. The RD computation is widely used as a sub-procedure in graph sparsification [40], graph clustering [1], counting random spanning trees [9, 18], robust routing of road networks [15] and understanding oversquashing in GNNs [4].

**Lanczos iteration.** Lanczos iteration is a powerful algorithm both theoretically and experimentally. In numerical analysis, understanding the stability of Lanczos iteration in finite precision is a key issue. For example, [27, 29] analyze the error propagation of Lanczos iteration in finite precision for matrix function approximation, [32] studies the error bound for eigenvalue approximation by Lanczos iteration, [7] studies the stability of Lanczos algorithm under regular spectral distributions. However, in this paper, in contrast to just analyzing rounding errors produced by Lanczos iteration, we actively create errors by our subset Lanczos recurrence (Eq. 5) to make our algorithm implemented locally. As a result, our proposed techniques have independent interest over these previous researches. Recently, Lanczos iteration has also been applied in various graph algorithms. For example, [30] use Lanczos iteration to compute the Heat Kernel PageRank value, [5] use Lanczos iteration to compute the Katz scores and commute times, [41] use stochastic Lanczos quadrature to estimate the Von Neumann graph entropy, [3, 8] use stochastic Lanczos quadrature to design the linear time algorithm to approximate the spectral density of a symmetric matrix, and [38] applies Lanczos iteration on a subgraph to computes the extreme eigenvector for local clustering. However, all these previous researches directly apply the Lanczos algorithm as a sub-procedure, while we provide the first algorithm that modifies the Lanczos algorithm by implementing it locally. Finally, note that although our algorithm is primarily designed to compute the RD value, it is also possible to be generalized to more applications. These generalizations are left to future work.

## 8 CONCLUSION

In this paper, we propose two efficient algorithms for RD computation, i.e., the Lanczos iteration and Lanczos Push. Specifically, we first show that the classical Lanczos iteration can be used to compute the RD value. We prove that Lanczos iteration can be implemented in $\tilde{O}(\sqrt{\kappa}m)$ time to approximate the RD value with $\epsilon$-absolute error guarantee, which is $\sqrt{\kappa}$ times faster than the traditional Power Method. Then, we develop a novel subset Lanczos recurrence technique to achieve a local algorithm for RD computation. Extensive experimental results demonstrate that our algorithms significantly outperform existing SOTA methods for RD computation, especially for large graphs and graphs with larger condition number $\kappa$.

## ACKNOWLEDGMENTS

This work was partially supported by the NSFC Grants U24A20255, U2241211, and 62427808. Rong-Hua Li is the corresponding author of this paper.

# REFERENCES


[1] Vedat Levi Alev, Nima Anari, Lap Chi Lau, and Shayan Oveis Gharan. Graph clustering using effective resistance. *arXiv preprint arXiv:1711.06530*, 2017.

[2] Alexandr Andoni, Robert Krauthgamer, and Yosef Pogrow. On solving linear systems in sublinear time. *arXiv preprint arXiv:1809.02995*, 2018.

[3] Rajarshi Bhattacharjee, Rajesh Jayaram, Cameron Musco, Christopher Musco, and Archan Ray. Improved spectral density estimation via explicit and implicit deflation. *arXiv preprint arXiv:2410.21690*, 2024.

[4] Mitchell Black, Zhengchao Wan, Amir Nayyeri, and Yusu Wang. Understanding oversquashing in gnns through the lens of effective resistance. In *International Conference on Machine Learning*, pages 2528–2547. PMLR, 2023.

[5] Francesco Bonchi, Pooya Esfandiar, David F Gleich, Chen Greif, and Laks VS Lakshmanan. Fast matrix computations for pairwise and columnwise commute times and katz scores. *Internet Mathematics*, 8(1-2):73–112, 2012.

[6] Dongrun Cai, Xue Chen, and Pan Peng. Effective resistances in non-expander graphs. *arXiv preprint arXiv:2307.01218*, 2023.

[7] Tyler Chen and Thomas Trogdon. Stability of the lanczos algorithm on matrices with regular spectral distributions. *ArXiv*, abs/2302.14842, 2023.

[8] Tyler Chen, Thomas Trogdon, and Shashanka Ubaru. Analysis of stochastic lanczos quadrature for spectrum approximation. In *International Conference on Machine Learning*, pages 1728–1739. PMLR, 2021.

[9] Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation via short cycle decompositions. *FOCS*, 2020.

[10] Fan Chung. The heat kernel as the pagerank of a graph. *Proceedings of the National Academy of Sciences*, 104(50), 2007.

[11] Michael B Cohen, Rasmus Kyng, Gary L Miller, Jakub W Pachocki, Richard Peng, Anup B Rao, and Shen Chen Xu. Solving sdd linear systems in nearly m log1/2 n time. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 343–352, 2014.

[12] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org. https://www.openstreetmap.org., 2017.

[13] Guanyu Cui, Hanzhi Wang, and Zhewei Wei. Mixing time matters: Accelerating effective resistance estimation via bidirectional method. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '25, 2025.

[14] Rajat Vadiraj Dwaraknath, Ishani Karmarkar, and Aaron Sidford. Towards optimal effective resistance estimation. *Advances in Neural Information Processing Systems*, 36, 2024.

[15] Yuan Gao, Rasmus Kyng, and Daniel A Spielman. Robust and practical solution of laplacian equations by approximate elimination. *arXiv preprint arXiv:2303.00709*, 2023.

[16] Arun Jambulapati and Aaron Sidford. Efficient $\tilde{O}(n/\epsilon)$ spectral sketches for the laplacian and its pseudoinverse. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2018.

[17] Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582. IEEE, 2016.

[18] Lawrence Li and Sushant Sachdeva. A new approach to estimating effective resistances and counting spanning trees in expander graphs. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2728–2745. SIAM, 2023.

[19] Meihao Liao, Cheng Li, Rong-Hua Li, and Guoren Wang. Efficient index maintenance for effective resistance computation on evolving graphs. *Proc. ACM Manag. Data*, 2025.

[20] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, Hongyang Chen, Hongchao Qin, and Guoren Wang. Efficient resistance distance computation: The power of landmark-based approaches. *Proc. ACM Manag. Data*, 2023.

[21] Meihao Liao, Yueyang Pan, Rong-Hua Li, and Guoren Wang. Efficient exact resistance distance computation on small-treewidth graphs: A labelling approach. *Proc. ACM Manag. Data*, 2025.

[22] Meihao Liao, Junjie Zhou, Rong-Hua Li, Qiangqiang Dai, Hongyang Chen, and Guoren Wang. Efficient and provable effective resistance computation on large graphs: an index-based approach. *Proceedings of the ACM on Management of Data*, 2(3):1–27, 2024.

[23] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. Personalized pagerank estimation and search: A bidirectional approach. In *WSDM*, 2016.

[24] Peter Lofgren and Ashish Goel. Personalized pagerank to a target node. *arXiv preprint arXiv:1304.4658*, 2013.

[25] Zenan Lu, Xiaotian Zhou, and Zhongzhi Zhang. Fast estimation and optimization of resistance diameter on graphs. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, page 2391–2401, 2025.

[26] Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 593–602. IEEE, 2016.

[27] Gérard Meurant and Zdeněk Strakoš. The lanczos and conjugate gradient algorithms in finite precision arithmetic. *Acta Numerica*, 15:471–542, 2006.

[28] Moshe Morgenstern. Existence and explicit constructions of q+ 1 regular ramanujan graphs for every prime power q. *Journal of Combinatorial Theory, Series B*, 62(1):44–62, 1994.

[29] Cameron Musco, Christopher Musco, and Aaron Sidford. Stability of the lanczos method for matrix function approximation. In *SODA*. SIAM, 2018.

[30] Lorenzo Orecchia, Sushant Sachdeva, and Nisheeth K Vishnoi. Approximating the exponential, the lanczos method and an o (m)-time spectral algorithm for balanced separator. In *STOC*, 2012.

[31] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford infolab, 1999.

[32] Chris C Paige. Accuracy and effectiveness of the lanczos algorithm for the symmetric eigenproblem. *Linear algebra and its applications*, 34:235–258, 1980.

[33] Pan Peng, Daniel Lopatta, Yuichi Yoshida, and Gramoz Goranci. Local algorithms for estimating effective resistance. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1329–1338, 2021.

[34] Dana Ron. Sublinear-time algorithms for approximating graph parameters. In *Computing and Software Science: State of the Art and Perspectives*, pages 105–122. Springer, 2019.

[35] Sushant Sachdeva, Nisheeth K Vishnoi, et al. Faster algorithms via approximation theory. *Foundations and Trends® in Theoretical Computer Science*, 9(2), 2014.

[36] Purnamrita Sarkar, Andrew W. Moore, and Amit Prakash. Fast incremental proximity search in large graphs. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, 2008.

[37] Jieming Shi, Nikos Mamoulis, Dingming Wu, and David W. Cheung. Density-based place clustering in geo-social networks. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2014.

[38] Pan Shi, Kun He, David Bindel, and John E Hopcroft. Local lanczos spectral approximation for community detection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 651–667. Springer, 2017.

[39] Ali Kemal Sinop, Lisa Fawcett, Sreenivas Gollapudi, and Kostas Kollias. Robust routing using electrical flows. *ACM Trans. Spatial Algorithms Syst.*, 2023.

[40] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 563–568, 2008.

[41] Anton Tsitsulin, Marina Munkhoeva, and Bryan Perozzi. Just slaq when you approximate: Accurate spectral distances for web-scale graphs. In *Proceedings of the Web Conference 2020*, pages 2697–2703, 2020.

[42] Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P Liu, Richard Peng, and Aaron Sidford. Faster maxflow via improved dynamic spectral vertex sparsifiers. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 543–556, 2022.

[43] Hanzhi Wang, Mingguo He, Zhewei Wei, Sibo Wang, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Approximate graph propagation. In *KDD*, 2021.

[44] Hanzhi Wang and Zhewei Wei. Estimating single-node pagerank in $\tilde{O}(min\{d_t, \sqrt{m}\})$ time. *Proc. VLDB Endow.*, 2023.

[45] Hanzhi Wang, Zhewei Wei, Junhao Gan, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Edge-based local push for personalized pagerank. *Proc. VLDB Endow.*, 2022.

[46] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. Fora: simple and effective approximate single-source personalized pagerank. In *KDD*, pages 505–514, 2017.

[47] Virginia Williams. Multiplying matrices faster than coppersmith-winograd. pages 887–898, 05 2012.

[48] Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. Unifying the global and local approaches: an efficient power iteration with forward push. In *SIGMOD*, 2021.

[49] Renchi Yang and Jing Tang. Efficient estimation of pairwise effective resistance. *Proceedings of the ACM on Management of Data*, 1(1):1–27, 2023.

[50] Renchi Yang, Xiaokui Xiao, Zhewei Wei, Sourav S Bhowmick, Jun Zhao, and Rong-Hua Li. Efficient estimation of heat kernel pagerank for local clustering. In *SIGMOD*, 2019.

[51] Yichun Yang, Rong-Hua Li, Meihao Liao, and Guoren Wang. Improved algorithms for effective resistance computation on graphs. COLT, 2025.

[52] Yichun Yang, Longlong Lin, Rong-Hua Li, Meihao Liao, and Guoren Wang. Theoretically and practically efficient resistance distance computation on large graphs. https://arxiv.org/abs/2601.11159, 2026.

[53] Yunfeng Yu, Longlong Lin, Qiyu Liu, Zeli Wang, Xi Ou, and Tao Jia. GSD-GNN: generalizable and scalable algorithms for decoupled graph neural networks. In *ICMR*, 2024.