Fast Estimation of Pairwise Biharmonic Distance on Graphs

CHANGAN LIU, College of Computer Science and Artificial Intelligence, Fudan University, China
XINNA ZHOU, College of Computer Science and Artificial Intelligence, Fudan University, China
BO ZHANG, College of Computer Science and Artificial Intelligence, Fudan University, China
AHAD N. ZEHMAKAN, School of Computing, Australian National University, Australia
ZHONGZHI ZHANG*, College of Computer Science and Artificial Intelligence, Fudan University, China

Biharmonic distance (BD) is a fundamental metric that measures the dissimilarity of a node pair s, t in a graph, encapsulating rich local and global structural information. It has found applications across various fields, including network science, graph machine learning, and computational graphics. Existing algorithms for pairwise BD queries, including the state-of-the-art (SOTA) solution that relies on sampling a significant number of long random walks, are often computationally infeasible on large graphs. In this work, we propose two novel formulations of BD based on a pivot node, derived through novel proof techniques. These formulations enhance our theoretical understanding of BD and enable efficient approximation algorithms. Our first algorithm, BackPush, is a deterministic method based on a newly established local *pushback* operation designed for BD. The second algorithm, FastWalk, is a randomized approach that estimates BD using ℓ-truncated absorbing random walks and counting their collisions. Lastly, we introduce FastTree, an algorithm inspired by a novel connection between BD and spanning trees. These algorithms, tailored to leverage more prior structural knowledge (particularly the relative connectivity to the pivot node), significantly improve efficiency and versatility compared to existing methods. Extensive empirical evaluations on benchmark networks with hundreds of millions of edges demonstrate that our algorithms can produce more accurate solutions than the SOTA methods over 100 times faster.

CCS Concepts: • **Theory of computation → Random walks and Markov chains**; • **Mathematics of computing → Graph algorithms**.

Additional Key Words and Phrases: Biharmonic Distance, Random Walk, Spanning Tree

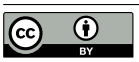
ACM Reference Format:
Changan Liu, Xinna Zhou, Bo Zhang, Ahad N. Zehmakan, and Zhongzhi Zhang. 2026. Fast Estimation of Pairwise Biharmonic Distance on Graphs. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 181 (June 2026), 27 pages. https://doi.org/10.1145/3802058

1 Introduction

What is the distance between a pair of nodes in a graph $\mathcal{G}$? To address this fundamental network science question, various measures of distance have been proposed, suitable for different applications [28, 59, 64, 74, 80]. One popular measure is the length of the shortest path between a node pair, known as geodesic distance [52]. While such measures are simple to define and compute, they

*Corresponding author: Zhongzhi Zhang.

Authors' Contact Information: Changan Liu, College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China, 19110240031@fudan.edu.cn; Xinna Zhou, College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China, 23210240097@m.fudan.edu.cn; Bo Zhang, College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China, 20300290009@fudan.edu.cn; Ahad N. Zehmakan, School of Computing, Australian National University, Canberra, Australia, ahadn.zehmakan@anu.edu.au; Zhongzhi Zhang, College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China, zhangzz@fudan.edu.cn.



ACM 2836-6573/2026/6-ART181
https://doi.org/10.1145/3802058

fall short in capturing complex local and global graph structural properties [42], which are essential for many applications.

The *biharmonic distance* (BD), originally introduced by Lipman et al. [39] to better capture local and global structural properties, has triggered extensive research. Its applications range from analyzing the robustness of dynamical networks in physics [65] and studying coherence in distributed systems [6, 17, 18, 78], to examining geometric relationships with the Kirchhoff index [31] in theoretical graph analysis [70]. Beyond these domains, BD serves as a critical metric in data management pipelines. Specifically, it has been linked to edge centrality [76], which allows for identifying critical connections. For instance, in a road network, analysts can evaluate BD to identify critical road segments essential for connectivity. Furthermore, graph learning pipelines [10, 32] leverage BD to mitigate the over-squashing problem [62], where batch jobs evaluate BD scores to identify optimal rewiring candidates. These scenarios inevitably require the system to handle pairwise BD evaluations over large-scale graphs efficiently.

Despite all BD's applications and numerous research works conducted, computing BD at scale remains a challenging task. Yi et al. [76–78] proposed a promising algorithm for approximating all-pairs BD, but the algorithm's need to construct a large, dense matrix during preprocessing makes it unsuitable for modern, rapidly growing networks. Recently, Liu et al. [42] developed several efficient algorithms to estimate pairwise BD based on random walk sampling and one of them, `SWF`, is the current state-of-the-art (SOTA) algorithm. Specifically, `SWF` works by estimating the degree-normalized (K_1, K_2)-step collision probability of the random walks, which is the probability that two random walks collide (i.e., pass through the same node) after K_1 and K_2 steps, respectively. However, the limitation of this algorithm is that, on large graphs, it requires large choices of K_1 and K_2 to achieve a satisfactory estimation accuracy, thus rendering the algorithm with high computational overhead.

We attribute `SWF`'s limitation to underusing prior structural knowledge. In heterogeneous graphs the signal relevant to a pairwise distance is not uniform: random walks often spend many steps shuttling within the same highly connected region [7, 44, 69], so fixed-length long-walk samplers like `SWF` keep advancing even when additional steps add little beyond what that region already reveals, inflating cost without a proportional gain in accuracy.

A direct idea is to make this structural prior explicit by designating a small set of highly connected pivot (index) nodes, stop random walks as soon as they hit any of them, and then estimate the distance between the query nodes indirectly from their *relative proximity* to these index nodes rather than letting the walks keep roaming. This strategy has successfully accelerated shortest-path [2, 23, 53] and for effective resistance queries [36, 37]. To our knowledge this idea has not been applied to computing BD. We adopt the same high-level idea, but adapting it is non-trivial as BD is defined via the square of the grounded Laplacian [51] and couples first-order with second-order interactions between nodes [42].

Recognizing the aforementioned gap, we aim to design algorithms that can accurately estimate BD for very large networks. We first fix a pivot node v and propose two novel formulations for BD based on this node. To derive them, we begin by proving that the submatrix of the Laplacian square $\widehat{\boldsymbol{L}}_v := (\boldsymbol{L}^2)_v$ is *invertible*, where $\widehat{\boldsymbol{L}}_v$ denotes squaring the graph Laplacian $\boldsymbol{L}$ and then removing the v-th row and v-th column. Then, through novel proofs, we present its inverse based on the inverse of the Laplacian submatrix. We subsequently establish the relationship between $\widehat{\boldsymbol{L}}^{\dagger}$, the Moore-Penrose pseudoinverse of the Laplacian square $\widehat{\boldsymbol{L}}$, and $\widehat{\boldsymbol{L}}_v^{-1}$, leading to the new formulations that decompose BD into terms built from $\boldsymbol{L}_v^{-2}$.

The new formulations inspire three novel combinatorial views of BD that motivate our algorithm designs. The first view is a backward relation for absorbing random walks with a trap node,

indicating that the expected visits to a node can be rebuilt from neighbor contributions, turning the computation into a purely local propagation. The second view interprets the key terms of the new formulations as collisions between absorbing walks. The third view connects the new formulations to path statistics on spanning trees.

These interpretations lead to three algorithms for BD estimation. First, based on the connection from $\boldsymbol{L}_v^{-1}$ to the absorbing random walks, we propose BackPush that works in a deterministic manner by backward *pushing* the transition probability hop-by-hop to the neighboring nodes. Second, we make use of the novel absorbing random walk-based interpretation of BD and propose FastWalk that simulates a number of random walks and counts their collisions. To avoid the random walks being too long, we further adopt the l-truncated absorbing random walk strategy, which permits us to theoretically establish a trade-off between accuracy and computational complexity. Third, FastTree relies on the spanning-tree interpretation of BD. The algorithm approximates BD by sampling spanning trees and aggregates lightweight path statistics to approximate the terms in the new formulations. We additionally propose several optimization techniques to accelerate the computation.

In practice the three methods complement one another. BackPush is most effective when absorption is fast and the affected region stays local, which is common in small-world [69] and scale-free [7] networks. FastWalk is robust on graphs with larger diameters where a purely local push would expand too widely, because it concentrates effort on informative collisions rather than exploring the whole neighborhood. FastTree benefits from near-linear tree sampling and compact path summaries and becomes the fastest option on several large networks in our experiments.

We evaluate the three algorithms on benchmark graphs with up to over 12 million nodes and 160 million edges. Across datasets, our estimators consistently outperform the best existing method for pairwise BD queries while matching or improving accuracy, and on a single commodity server the fastest variant often reduces query time by a large margin.

Our main contributions can be summarized as follows.

- We develop novel formulations for BD using the inverse of the v-grounded Laplacian matrix [51], which inspires three novel combinatorial interpretations of BD.
- Leveraging the new formulations and interpretations, we present three novel approaches for computing pairwise BD. Among which, BackPush works by performing the novel deterministic *pushback* operation that propagates the transition probability hop-by-hop to the neighbors of the current node. FastWalk works by simulating l-truncated absorbing random walks and then counting their collisions. FastTree works by sampling spanning trees and then performing path statistics in the tree.
- We present complexity analyses for all three algorithms. Additionally, for FastWalk, we derive a principled rule for choosing the truncation length l. For FastTree, we introduce four non-trivial optimizations that yield substantial practical speedups.
- Extensive experiments using real datasets demonstrate that on a single commodity server, our proposed algorithms provide consistent speedups over the state of the art together with strong accuracy across regimes.

Outline. The remainder of the paper is organized as follows. In Section 2, we provide preliminaries for our study. We propose novel BD formulations in Section 3 and then propose our first solution BackPush in Section 4. We further propose a Monte Carlo algorithm FastWalk in Section 5. We propose our third algorithm FastTree in Section 6. Our solutions are evaluated in Section 7. Related work is reviewed in Section 8. Finally, Section 9 provides the conclusion.

Table 1. Summary of notation.

Symbol	Description
$\boldsymbol{x}_{-i}$	Vector $\boldsymbol{x}$ without its i-th element.
$\boldsymbol{M}_i$	Matrix $\boldsymbol{M}$ without its i-th row and column
$\boldsymbol{e}_i$ and $\boldsymbol{b}_{st}$	i-th standard basis and $\boldsymbol{b}_{st}=\boldsymbol{e}_s-\boldsymbol{e}_t$ for nodes s,t
$\mathcal{G}=(V,E)$	Graph with node set V and edge set $E\subseteq V\times V$
$\mathcal{N}(i)$	Neighbor set of node i
$\boldsymbol{A}\in\{0,1\}^{n\times n}$	Adjacency matrix; $\boldsymbol{A}_{ij}=1$ if $(i,j)\in E$
$\boldsymbol{P}$	Transition matrix
$\boldsymbol{L}$ and $\widehat{\boldsymbol{L}}$	Graph Laplacian and its square $\boldsymbol{L}^2$
$\boldsymbol{L}^\dagger$	Moore–Penrose pseudoinverse of Laplacian
$\boldsymbol{L}_v$	Ground Laplacian
$\beta(s,t)$ ($\beta'(s,t)$)	BD between nodes s and t (its estimation)
$h_{s,t}$	Hitting time from s to t
$\pi_v(s,t)$	Expected visits from s to t for an absorbing random walk with a trap node v
$r_{\max}$	Push threshold in BackPush
ω_w and l	Sampling size and truncation length in FastWalk
ω_t	Sampling size in FastTree

2 Preliminary

This section sets up the stage for our study by introducing basic notations, the formal problem definition of pairwise BD query, and the existing algorithms for approximating BD.

2.1 Notations

Throughout this paper, we utilize bold lowercase (e.g., $\boldsymbol{x}$) for (column) vectors and uppercase (e.g., $\boldsymbol{M}$) for matrices. Elements are indicated by subscripts, e.g., $\boldsymbol{x}_i$ or $\boldsymbol{M}_{ij}$. For matrices, $\boldsymbol{M}[i,:]$ and $\boldsymbol{M}[:,j]$ represent the i-th row and j-th column, respectively. We use $\boldsymbol{e}_i$ to denote the i-th standard basis vector of appropriate dimension, with the i-th element being 1 and other elements being 0. For any pair of distinct nodes $s,t\in V$, we define $\boldsymbol{b}_{st}=\boldsymbol{e}_s-\boldsymbol{e}_t$. We use $\mathbf{1}$ (resp. $\mathcal{J}$) to denote the all-ones (column) vector (resp. matrix) of proper dimension. Let $\boldsymbol{I}$ be the identity matrix. Furthermore, let $\mathbb{I}_{x=y}$ be an indicator function, which equals 1 if $x=y$ and 0 otherwise. Let $\boldsymbol{M}_i$ stands for the submatrix of $\boldsymbol{M}$ obtained by removing its i-th row and i-th column. The precedence of matrix subscripts is the lowest, that is, $\boldsymbol{M}_i^{-1}$ denotes the inverse of $\boldsymbol{M}_i$ instead of a submatrix of $\boldsymbol{M}^{-1}$. For a vector $\boldsymbol{x}$, $\boldsymbol{x}_{-i}$ denotes its subvector obtained by removing its i-th element and $\boldsymbol{x}^\top$ denotes its transpose. Table 1 summarizes the frequently used notations.

2.2 Graph Related Concepts

Consider a connected undirected graph $\mathcal{G}=(V,E)$ with node set V and edge set $E\subseteq V\times V$. Let $n:=|V|$ and $m:=|E|$ denote the number of nodes and edges respectively. A spanning tree of $\mathcal{G}$ is a connected subgraph with n nodes and $n-1$ edges. Let $\mathcal{N}(i)$ denote the neighbor set of i, and $\boldsymbol{d}_i=|\mathcal{N}(i)|$ its degree. The Laplacian matrix of $\mathcal{G}$ is a symmetric matrix $\boldsymbol{L}=\boldsymbol{D}-\boldsymbol{A}$, where $\boldsymbol{A}\in\{0,1\}^{n\times n}$ is the adjacency matrix whose entry $\boldsymbol{A}_{ij}=1$ if $(i,j)\in E$, and $\boldsymbol{A}_{ij}=0$ otherwise, and $\boldsymbol{D}$ is a diagonal matrix $\boldsymbol{D}=\mathtt{diag}(\boldsymbol{d}_1,\cdots,\boldsymbol{d}_n)$. Matrix $\boldsymbol{L}$ is positive semi-definite. Its Moore-Penrose pseudoinverse is $\boldsymbol{L}^\dagger=\left(\boldsymbol{L}+\frac{1}{n}\mathcal{J}\right)^{-1}-\frac{1}{n}\mathcal{J}$. We denote by $\widehat{\boldsymbol{L}}:=\boldsymbol{L}^2$ the square of the Laplacian matrix. We denote by $\boldsymbol{L}_v$ the v-grounded Laplacian matrix [51], obtained from $\boldsymbol{L}$ by deleting its v-th row and column.

The classical random walk on $\mathcal{G}$ can be defined by the transition matrix $\boldsymbol{P}=\boldsymbol{D}^{-1}\boldsymbol{A}$, in which $\boldsymbol{P}_{ij}=\frac{1}{\boldsymbol{d}_i}$ if $(i,j)\in E$ and $\boldsymbol{P}_{ij}=0$ otherwise. If we set a node $v\in V$ as the trap node, which means

that a classical random walk terminates upon reaching v, then we obtain an absorbing random walk on graph $\mathcal{G}$.

2.3 Problem Definition

Definition 2.1. *(Pairwise BD [42, 76]) For a connected undirected graph $\mathcal{G} = (V, E)$ with the square of the Laplacian matrix $\widehat{\boldsymbol{L}}$, the BD $\beta(s,t)$ between any two distinct nodes $s, t \in V$ is defined as*

$$\beta(s,t) := \boldsymbol{b}_{st}^\top \widehat{\boldsymbol{L}}^\dagger \boldsymbol{b}_{st} = \widehat{\boldsymbol{L}}_{ss}^\dagger + \widehat{\boldsymbol{L}}_{tt}^\dagger - 2\widehat{\boldsymbol{L}}_{st}^\dagger. \tag{1}$$

We study the *pairwise BD computation* problem, where the goal is to find $\beta(s,t)$ for given a connected undirected graph $\mathcal{G} = (V, E)$ and two nodes s, t with $s \neq t$.

2.4 Existing Algorithms

Following its definition [39, 76], exact BD computation requires the Moore-Penrose pseudoinverse of the Laplacian matrix, which has a time complexity of $O(n^3)$. Approximation methods, such as solving the linear Laplacian system $\boldsymbol{Lx} = \boldsymbol{b}$, remain computationally expensive for large graphs, despite optimizations reducing the time to $\tilde{O}(m)$ [12, 21, 61]. Random projection-based methods [76–78] allow approximate BD queries in $O(\log n)$ time but require $\tilde{O}(m/\epsilon^2)$ time in preprocessing, involving large-scale matrices impractical for massive networks or scenarios where only a limited set of node pairs are of interest. Additionally, these methods assume full graph knowledge, which is often unrealistic in practice.

To address these issues, Liu et al. [42] proposed novel algorithms that only require local information and can efficiently answer BD queries for a set of node pairs, without the need for costly preprocessing. Building on the transition probability matrix $\boldsymbol{P}$, they showed that the BD between nodes s and t can be expressed as

$$\beta(s,t) = \|\boldsymbol{h}\|_2^2 - \frac{1}{n}(\boldsymbol{h}^\top \mathbf{1})^2,$$

where $\boldsymbol{h}^\top = \sum_{i=0}^{\infty} \boldsymbol{b}_{st}^\top \boldsymbol{P}^i \boldsymbol{D}^{-1}$. To estimate $\beta(s,t)$, they truncate the summation in the expression of $\boldsymbol{h}$ using two kinds of truncation lengths K, and propose a SOTA algorithm, `SWF`, which speeds up computations by sampling simple random walks of length K and counting degree-normalized collisions (i.e., when two random walks pass through the same node). However, the main limitation of `SWF` is that, in large graphs, achieving good accuracy often requires a large K, leading to undesirable running times.

3 New formulation for BD

In this section, we develop two novel formulations for BD. These new formulations provide the foundation for our approximation algorithms to compute BD.

In the following, we designate an arbitrary node $v \in V$ as the *pivot* node. By reordering the vertices if necessary, we assume without loss of generality that v is indexed as the last node in $\boldsymbol{L}$ and $\widehat{\boldsymbol{L}}$. This permutation is purely for notational convenience and does not change the graph structure or any BD values, as they are invariant under node permutations.

It is worth noting that while the correctness of our formulation holds for any $v \in V$, the specific choice may affect algorithmic efficiency. In practice, high-degree nodes often serve as effective pivots. We provide an analysis of pivot selection strategies in Section 7.4.

We first make some preparatory work in the following two lemmas.

Lemma 3.1. *For any pivot node $v \in V$, the submatrix of the Laplacian square $\widehat{\boldsymbol{L}}_v$ is invertible and the inverse is given by $\boldsymbol{T} := \boldsymbol{L}_v^{-2} - \frac{\boldsymbol{L}_v^{-2}\boldsymbol{a}\boldsymbol{a}^\top \boldsymbol{L}_v^{-2}}{1+\boldsymbol{a}^\top \boldsymbol{L}_v^{-2}\boldsymbol{a}}$ where vector $\boldsymbol{a} = \boldsymbol{L}[1 : n-1, n]$.*

Proof. We can write $\boldsymbol{L}$ as a block matrix $\begin{pmatrix} L_v & \boldsymbol{a} \\ \boldsymbol{a}^\top & d_v \end{pmatrix}$. The square of the Laplacian matrix can then be written as

$$\widehat{\boldsymbol{L}} = \begin{pmatrix} L_v & \boldsymbol{a} \\ \boldsymbol{a}^\top & d_v \end{pmatrix} \begin{pmatrix} L_v & \boldsymbol{a} \\ \boldsymbol{a}^\top & d_v \end{pmatrix} = \begin{pmatrix} \boldsymbol{a}\boldsymbol{a}^\top + L_v^2 & d_v\boldsymbol{a} + L_v\boldsymbol{a} \\ d_v\boldsymbol{a}^\top + \boldsymbol{a}^\top L_v & d_v^2 + \boldsymbol{a}^\top\boldsymbol{a} \end{pmatrix}.$$

Therefore, $\widehat{\boldsymbol{L}}_v = \boldsymbol{a}\boldsymbol{a}^\top + L_v^2$. Thus, the following statement holds

$$\begin{aligned} \widehat{\boldsymbol{L}}_v\boldsymbol{T} &= (\boldsymbol{a}\boldsymbol{a}^\top + L_v^2)(L_v^{-2} - \frac{L_v^{-2}\boldsymbol{a}\boldsymbol{a}^\top L_v^{-2}}{1 + \boldsymbol{a}^\top L_v^{-2}\boldsymbol{a}}) \\ &= \boldsymbol{a}\boldsymbol{a}^\top L_v^{-2} + L_v^2 L_v^{-2} - \frac{\boldsymbol{a}\boldsymbol{a}^\top L_v^{-2}\boldsymbol{a}\boldsymbol{a}^\top L_v^{-2}}{1 + \boldsymbol{a}^\top L_v^{-2}\boldsymbol{a}} - \frac{L_v^2 L_v^{-2}\boldsymbol{a}\boldsymbol{a}^\top L_v^{-2}}{1 + \boldsymbol{a}^\top L_v^{-2}\boldsymbol{a}} \\ &= \boldsymbol{a}\boldsymbol{a}^\top L_v^{-2} + \boldsymbol{I} - \frac{(1 + \boldsymbol{a}^\top L_v^{-2}\boldsymbol{a})\boldsymbol{a}\boldsymbol{a}^\top L_v^{-2}}{1 + \boldsymbol{a}^\top L_v^{-2}\boldsymbol{a}} = \boldsymbol{I}. \end{aligned}$$

Similarly, we can prove that $\boldsymbol{T}\widehat{\boldsymbol{L}}_v = \boldsymbol{I}$. This implies $\boldsymbol{T} = \widehat{\boldsymbol{L}}_v^{-1}$. □

Lemma 3.2. *The summation of each column and each row of $\widehat{\boldsymbol{L}}$ and $\widehat{\boldsymbol{L}}^\dagger$ equals 0. Namely, $\widehat{\boldsymbol{L}}\mathbf{1} = \mathbf{0}$, $\mathbf{1}^\top\widehat{\boldsymbol{L}} = \mathbf{0}$, $\widehat{\boldsymbol{L}}^\dagger\mathbf{1} = \mathbf{0}$ and $\mathbf{1}^\top\widehat{\boldsymbol{L}}^\dagger = \mathbf{0}$.*

Proof. It was proven in [15] that $\boldsymbol{L}\mathbf{1} = \mathbf{0}$, $\mathbf{1}^\top\boldsymbol{L} = \mathbf{0}$, $\boldsymbol{L}^\dagger\mathbf{1} = \mathbf{0}$, and $\mathbf{1}^\top\boldsymbol{L}^\dagger = \mathbf{0}$. Thus, we could derive that $\widehat{\boldsymbol{L}}\mathbf{1} = \boldsymbol{L}\boldsymbol{L}\mathbf{1} = \mathbf{0}$, $\mathbf{1}^\top\widehat{\boldsymbol{L}} = \mathbf{1}^\top\boldsymbol{L}\boldsymbol{L} = \mathbf{0}$, $\widehat{\boldsymbol{L}}^\dagger\mathbf{1} = \boldsymbol{L}^\dagger\boldsymbol{L}^\dagger\mathbf{1} = \mathbf{0}$, and $\mathbf{1}^\top\widehat{\boldsymbol{L}}^\dagger = \mathbf{1}^\top\boldsymbol{L}^\dagger\boldsymbol{L}^\dagger = \mathbf{0}$. □

Having established Lemma 3.1 and Lemma 3.2, we proceed to relate $\widehat{\boldsymbol{L}}^\dagger$ to $\widehat{\boldsymbol{L}}_v^{-1}$ in the following lemma.

Lemma 3.3. *$\widehat{\boldsymbol{L}}^\dagger$ can be related to $\widehat{\boldsymbol{L}}_v^{-1}$ as follows.*

$$\widehat{\boldsymbol{L}}^\dagger = \begin{pmatrix} I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top \\ -\frac{1}{n}\mathbf{1}^\top \end{pmatrix} \widehat{\boldsymbol{L}}_v^{-1} \begin{pmatrix} I - \frac{1}{n}\mathbf{1}\mathbf{1}^T & -\frac{1}{n}\mathbf{1} \end{pmatrix}. \tag{2}$$

Proof. $\widehat{\boldsymbol{L}}$ can be written as a block matrix $\begin{pmatrix} \widehat{\boldsymbol{L}}_v & \boldsymbol{p} \\ \boldsymbol{p}^\top & \widehat{\boldsymbol{L}}_{vv} \end{pmatrix}$ where $\boldsymbol{p} = \widehat{\boldsymbol{L}}[1 : n-1, n]$. $\widehat{\boldsymbol{L}}^\dagger$ can be written as a block matrix $\begin{pmatrix} \widehat{\boldsymbol{L}}_v^\dagger & \boldsymbol{q} \\ \boldsymbol{q}^\top & \widehat{\boldsymbol{L}}_{vv}^\dagger \end{pmatrix}$ where $\boldsymbol{q} = \widehat{\boldsymbol{L}}^\dagger[1 : n-1, n]$. Based on Lemma 3.2, we have that $\widehat{\boldsymbol{L}}_v\mathbf{1} + \boldsymbol{p} = \mathbf{0}$, $\mathbf{1}^\top\widehat{\boldsymbol{L}}_v + \boldsymbol{p}^\top = \mathbf{0}$, and $\widehat{\boldsymbol{L}}_{vv} = -\mathbf{1}^\top\boldsymbol{p} = \mathbf{1}^\top\widehat{\boldsymbol{L}}_v\mathbf{1}$; we also have $(\widehat{\boldsymbol{L}}^\dagger)_v\mathbf{1} + \boldsymbol{q} = \mathbf{0}$, $\mathbf{1}^\top(\widehat{\boldsymbol{L}}^\dagger)_v + \boldsymbol{q}^\top = \mathbf{0}$, and $(\widehat{\boldsymbol{L}}^\dagger)_{vv} = -\mathbf{1}^\top\boldsymbol{q} = \mathbf{1}^\top(\widehat{\boldsymbol{L}}^\dagger)_v\mathbf{1}$. Then, we derive

$$\begin{aligned} \left(\boldsymbol{I} + \mathbf{1}\mathbf{1}^\top\right)(\widehat{\boldsymbol{L}}^\dagger)_v\left(\boldsymbol{I} + \mathbf{1}\mathbf{1}^\top\right) =& (\widehat{\boldsymbol{L}}^\dagger)_v + (\widehat{\boldsymbol{L}}^\dagger)_v\mathbf{1}\mathbf{1}^\top + \mathbf{1}\mathbf{1}^\top(\widehat{\boldsymbol{L}}^\dagger)_v + \mathbf{1}\mathbf{1}^\top(\widehat{\boldsymbol{L}}^\dagger)_v\mathbf{1}\mathbf{1}^\top \\ =& (\widehat{\boldsymbol{L}}^\dagger)_v - \boldsymbol{q}\mathbf{1}^\top - \mathbf{1}\boldsymbol{q}^\top + \mathbf{1}\widehat{\boldsymbol{L}}_{vv}^\dagger\mathbf{1}^\top = \begin{pmatrix} \boldsymbol{I} & -\mathbf{1} \end{pmatrix}\widehat{\boldsymbol{L}}^\dagger\begin{pmatrix} \boldsymbol{I} \\ -\mathbf{1}^\top \end{pmatrix}. \end{aligned} \tag{3}$$

Multiplying on the left and right sides of the term in the last line of Eq. (3) by $\widehat{\boldsymbol{L}}_v$ yields

$$\widehat{\boldsymbol{L}}_v\begin{pmatrix} \boldsymbol{I} & -\mathbf{1} \end{pmatrix}\widehat{\boldsymbol{L}}^\dagger\begin{pmatrix} \boldsymbol{I} \\ -\mathbf{1}^\top \end{pmatrix}\widehat{\boldsymbol{L}}_v = \begin{pmatrix} \widehat{\boldsymbol{L}}_v & \boldsymbol{p} \end{pmatrix}\widehat{\boldsymbol{L}}^\dagger\begin{pmatrix} \widehat{\boldsymbol{L}}_v \\ \boldsymbol{p}^\top \end{pmatrix} = \begin{pmatrix} \boldsymbol{I} & \mathbf{0} \end{pmatrix}\widehat{\boldsymbol{L}}\widehat{\boldsymbol{L}}^\dagger\widehat{\boldsymbol{L}}\begin{pmatrix} \boldsymbol{I} \\ \mathbf{0}^\top \end{pmatrix} = \begin{pmatrix} \boldsymbol{I} & \mathbf{0} \end{pmatrix}\widehat{\boldsymbol{L}}\begin{pmatrix} \boldsymbol{I} \\ \mathbf{0}^\top \end{pmatrix} = \widehat{\boldsymbol{L}}_v.$$

Multiplying both sides of the above equation by matrix $\widehat{\boldsymbol{L}}_v^{-1}$ on the left and right results in $\begin{pmatrix} \boldsymbol{I} & -\mathbf{1} \end{pmatrix}\widehat{\boldsymbol{L}}^\dagger\begin{pmatrix} \boldsymbol{I} \\ -\mathbf{1}^\top \end{pmatrix} = \widehat{\boldsymbol{L}}_v^{-1}$. Multiplying this equation on the left and right side respectively by $\begin{pmatrix} \boldsymbol{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top \\ -\frac{1}{n}\mathbf{1}^\top \end{pmatrix}$ and $\begin{pmatrix} \boldsymbol{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T & -\frac{1}{n}\mathbf{1} \end{pmatrix}$ gives us $\widehat{\boldsymbol{L}}^\dagger = \begin{pmatrix} \boldsymbol{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top \\ -\frac{1}{n}\mathbf{1}^\top \end{pmatrix}\widehat{\boldsymbol{L}}_v^{-1}\begin{pmatrix} \boldsymbol{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T & -\frac{1}{n}\mathbf{1} \end{pmatrix}$, ending the proof. □

Based on Lemma 3.3, we can obtain the following element-wise representation of $\widehat{\boldsymbol{L}}^{\dagger}$.

Corollary 3.4. *$\widehat{\boldsymbol{L}}^{\dagger}$ can be represented using $\widehat{\boldsymbol{L}}_v^{-1}$ element-wisely:*

$$\widehat{\boldsymbol{L}}^{\dagger}_{st} = \boldsymbol{e}_s^{\top}\widehat{\boldsymbol{L}}_v^{-1}\boldsymbol{e}_t - \frac{1}{n}\boldsymbol{e}_s^{\top}\widehat{\boldsymbol{L}}_v^{-1}\mathbf{1} - \frac{1}{n}\boldsymbol{e}_t^{\top}\widehat{\boldsymbol{L}}_v^{-1}\mathbf{1} + \frac{1}{n^2}\mathbf{1}^{\top}\widehat{\boldsymbol{L}}_v^{-1}\mathbf{1}, \quad s,t \neq v, \tag{4}$$

$$\widehat{\boldsymbol{L}}^{\dagger}_{ss} = \boldsymbol{e}_s^{\top}\widehat{\boldsymbol{L}}_v^{-1}\boldsymbol{e}_s - 2\frac{1}{n}\boldsymbol{e}_s^{\top}\widehat{\boldsymbol{L}}_v^{-1}\mathbf{1} + \frac{1}{n^2}\mathbf{1}^{\top}\widehat{\boldsymbol{L}}_v^{-1}\mathbf{1}, \quad s \neq v, \tag{5}$$

$$\widehat{\boldsymbol{L}}^{\dagger}_{sv} = -\frac{1}{n}\boldsymbol{e}_s^{\top}\widehat{\boldsymbol{L}}_v^{-1}\mathbf{1} + \frac{1}{n^2}\mathbf{1}^{\top}\widehat{\boldsymbol{L}}_v^{-1}\mathbf{1}, \quad s \neq v, \tag{6}$$

$$\widehat{\boldsymbol{L}}^{\dagger}_{vv} = \frac{1}{n^2}\mathbf{1}^{\top}\widehat{\boldsymbol{L}}_v^{-1}\mathbf{1}. \tag{7}$$

New formulations for BD. Based on the above results, we can present novel formulations for BD.

Theorem 3.5. *For any two nodes $s, t \neq v$, we have*

$$\beta(s,t) = \boldsymbol{b}_{st}^{\top}\boldsymbol{L}_v^{-2}\boldsymbol{b}_{st} - \boldsymbol{b}_{st}^{\top}\frac{\boldsymbol{L}_v^{-2}\boldsymbol{a}\boldsymbol{a}^{\top}\boldsymbol{L}_v^{-2}}{1+\boldsymbol{a}^{\top}\boldsymbol{L}_v^{-2}\boldsymbol{a}}\boldsymbol{b}_{st}. \tag{8}$$

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

$$\beta(u,v) = \boldsymbol{e}_u^{\top}\boldsymbol{L}_v^{-2}\boldsymbol{e}_u - \boldsymbol{e}_u^{\top}\frac{\boldsymbol{L}_v^{-2}\boldsymbol{a}\boldsymbol{a}^{\top}\boldsymbol{L}_v^{-2}}{1+\boldsymbol{a}^{\top}\boldsymbol{L}_v^{-2}\boldsymbol{a}}\boldsymbol{e}_u. \tag{9}$$

Proof. According to Lemma 3.1 and plugging Eq. (4), Eq. (5), Eq. (6), and Eq. (7) into Eq. (1), it is straightforward to derive the new formulations. □

4 A Backward Push Algorithm

In this section, we propose a deterministic local backward push algorithm based on the proposed new formulations for BD.

4.1 High-level Ideas

Theorem 3.5 provides two combinatorial explanations of BD using the elements of $\boldsymbol{L}_v^{-1}$. It is well-established that these elements are closely related to the transition probability of absorbing random walks [82]. Specifically, we have: $(\boldsymbol{L}_v^{-1})_{us} = ((\boldsymbol{I}-\boldsymbol{P}_v)^{-1}\boldsymbol{D}_v^{-1})_{us} = (\boldsymbol{I}-\boldsymbol{P}_v)^{-1}_{us}/\boldsymbol{d}_s = \sum_{k=0}^{\infty} p_v^k(u,s)/\boldsymbol{d}_s$, where $p_v^k(u,s)$ denotes the k-hop transition probability from u to s for a random walk with node v as a trap. Therefore, $(\boldsymbol{L}_v^{-1})_{us}$ can be interpreted as the degree-normalized expected number of times, $\pi_v(u,s)$, that an absorbing random walk starting from u visits s. The first term in Eq. (8) can thus be expressed as

$$\sum\nolimits_{u\in V\setminus\{v\}}\left(\frac{\pi_v(u,s)}{\boldsymbol{d}_s} - \frac{\pi_v(u,t)}{\boldsymbol{d}_t}\right)^2.$$

The estimation of the second term poses significant computational challenges as there can be $\Theta(n)$ non-zero elements in $\boldsymbol{a}$, necessitating the estimation of $\pi_v(u,a)$ for $\Theta(n)$ distinct values of a. While the values of $\pi_v(u,s)$ and $\pi_v(u,t)$ can be estimated concurrently during the computation of the first term, the overall estimating process can be computationally inefficient.

To overcome this issue, we note that $\sum_{a\in\mathcal{N}(v)}\pi_v(u,a)/\boldsymbol{d}_a$ can be reformulated as $\pi_v(u,v)$, where $\pi_v(u,v)$ denotes the expected number of visits to node v. Based on this novel reformulation, we next proceed to decompose the second term and investigate efficient approximation strategies. Concretely, the numerator of the second item in Eq. (8) is the square of $\boldsymbol{b}_{st}^{\top}\boldsymbol{L}_v^{-2}\boldsymbol{a}$, which expands to

$$\sum\nolimits_{u\in V\setminus\{v\}}\left(\frac{\pi_v(u,s)}{\boldsymbol{d}_s} - \frac{\pi_v(u,t)}{\boldsymbol{d}_t}\right)\pi_v(u,v),$$

where $\pi_v(u,v) = \sum_{a\in\mathcal{N}(v)} \frac{\pi_v(u,a)}{d_a}$. The denominator of the second item can be expressed as

$$\sum\nolimits_{u\in V\setminus\{v\}} \pi_v^2(u,v).$$

According to the above decompositions of the terms in Eq. (8), if we could accurately estimate $\pi_v(u,s)$, $\pi_v(u,t)$, and $\pi_v(u,v)$, then we could get the estimation of BD, which is the key idea of the algorithm in this section.

4.2 The BackPush Algorithm

To estimate $\pi_v(u,x)$ for each $x \in \{s,t,v\}$, we establish a recursive relationship between $\pi_v(u,x)$ and $\pi_v(u,w)$ for $w \in \mathcal{N}(x)$. Specifically, for a random walk that reaches x at the L-th step, it must first reach one neighbor $w \neq v$ of x at the $(L-1)$-th step. Thus, for the L-hop transition probability $p_v^L(u,x)$, we derive that

$$p_v^L(u,x) = \sum\nolimits_{w\in\mathcal{N}(x), w\neq v} \frac{p_v^{L-1}(u,w)}{\boldsymbol{d}_w}.$$

Summing over all possible hops $L \in [1,+\infty)$, we obtain:

$$\sum\nolimits_{L=1}^{\infty} p_v^L(u,x) = \sum\nolimits_{w\in\mathcal{N}(x), w\neq v} \sum\nolimits_{L=0}^{\infty} \frac{p_v^L(u,w)}{\boldsymbol{d}_w},$$

which simplifies to $\sum_{w\in\mathcal{N}(x), w\neq v} \pi_v(u,w)\frac{1}{\boldsymbol{d}_w}$. Clearly, the initial condition $p_v^0(u,x)$ is 1 if $x = u$, since the node x always starts by visiting itself, and 0 otherwise. Consequently, we can derive:

$$\pi_v(u,x) = \mathbb{I}_{u=x} + \sum\nolimits_{w\in\mathcal{N}(x), w\neq v} \frac{\pi_v(u,w)}{\boldsymbol{d}_w},$$

where $\mathbb{I}_{u=x}$ is 1 if $u = x$, and 0 otherwise.

This recursive relationship demonstrates that the expected number of visits to a node x in a random walk can be expressed as the sum of visits to its neighbors w, excluding v. By extension, these visits to the 1-hop neighbors can, in turn, be represented by contributions from 2-hop neighbors, 3-hop neighbors, and so forth. Inspired by this insight, we introduce the BackPush algorithm, detailed in Algorithm 1, to approximate $\beta(s,t)$. Alongside the graph $\mathcal{G}$, the trap node v, and the query nodes s and t, BackPush also takes as input a user-defined parameter $r_{\max}$. Below, we explain how BackPush performs the approximations.

The BackPush algorithm approximates $\pi_v(u,x)$ by $\hat{\pi}_v(u,x)$ for each $x \in \{s,t,v\}$. Initially, the residual $r(u,x)$ is set to 0 for all $u \in V\setminus\{x\}$ and 1 for $r(x,x)$ (Lines 1-2), and then initializes $\hat{\pi}_v(u,s)$, $\hat{\pi}_v(u,t)$, and $\hat{\pi}_v(u,v)$ to 0 for all $u \in V\setminus\{v\}$ (Line 3). For each node y with $r(y,x)$ greater than $r_{\max}$ (Line 5), BackPush performs the following steps (Lines 6-9), as illustrated in Fig. 1:

(i) adds the residual $r(y,x)$ to $\hat{\pi}_v(y,x)$,
(ii) propagates a copy of $r(y,x)$ to its neighbors, excluding the trap node v . For each neighbor $w \neq v$ of y, $r(w,x)$ is incremented by $r(y,x)/\boldsymbol{d}_w$,
(iii) sets $r(y,x) = 0$.

We refer to these three steps as a *pushback* operation. Once the propagation process is complete, meaning no node y with $r(y,x) > r_{\max}$ remains, the estimations are obtained. Next, we compute X by summing $\left(\frac{\hat{\pi}_v(u,s))}{\boldsymbol{d}_s} - \frac{\hat{\pi}_v(u,t)}{\boldsymbol{d}_t}\right)^2$, which serves as an approximation for $\boldsymbol{b}_{st}^\top \boldsymbol{L}_v^{-2}\boldsymbol{b}_{st}$. We then compute Y by summing $\left(\frac{\hat{\pi}_v(u,s)}{\boldsymbol{d}_s} - \frac{\hat{\pi}_v(u,t)}{\boldsymbol{d}_t}\right)\hat{\pi}_v(u,v)$, which approximates $\boldsymbol{b}_{st}^\top \boldsymbol{L}_v^{-2}\boldsymbol{a}$. Subsequently, BackPush calculates Z by summing $\hat{\pi}_v^2(u,v)$, which approximates $\boldsymbol{a}^\top \boldsymbol{L}_v^{-2}\boldsymbol{a}$. Finally, BackPush computes $\beta'(s,t)$, the estimated BD between the query nodes (Line 15).

Algorithm 1: BackPush($\mathcal{G}, v, s, t, r_{\max}$)

Input : A connected graph $\mathcal{G} = (V, E)$; a node v; two query nodes s and t; a threshold $r_{\max}$
Output : Estimated BD $\beta'(s,t)$

for $x \in \{s, t, v\}$ **do**
  $r(u,x) \leftarrow 0\ \forall u \in V \backslash \{x, v\}$; $r(x,x) \leftarrow 1$
$\hat{\pi}_v(u,s) \leftarrow 0$, $\hat{\pi}_v(u,t) \leftarrow 0$, $\hat{\pi}_v(u,v) \leftarrow 0\ \forall u \in V \backslash \{v\}$
for $x \in \{s, t, v\}$ **do**
  while $\exists y \in V$ *s.t.* $r(y,x) \geq r_{\max}$ **do**
    $\hat{\pi}_v(y,x) = \hat{\pi}_v(y,x) + r(y,x)$
    for $w \in \mathcal{N}(y), w \neq v$ **do**
      $r(w,x) = r(w,x) + r(y,x)/\boldsymbol{d}_w$
    $r(y,x) = 0$
$X, Y, Z = 0$
for $u \in V \setminus \{v\}$ **do**
  $X = X + (\hat{\pi}_v(u,s)/\boldsymbol{d}_s - \hat{\pi}_v(u,t)/\boldsymbol{d}_t)^2$
  $Y = Y + (\hat{\pi}_v(u,s)/\boldsymbol{d}_s - \hat{\pi}_v(u,t)/\boldsymbol{d}_t)\,\hat{\pi}_v(u,v)$
  $Z = Z + \hat{\pi}_v^2(u,v)$
$\beta'(s,t) \leftarrow X - Y^2/(1+Z)$
return $\beta'(s,t)$

Note that in this work, we set $r_{\max}$ greater than 0 to save the time cost, following existing works [4, 24, 73]. If $r_{\max} = 0$, then we derive the exact BD value.

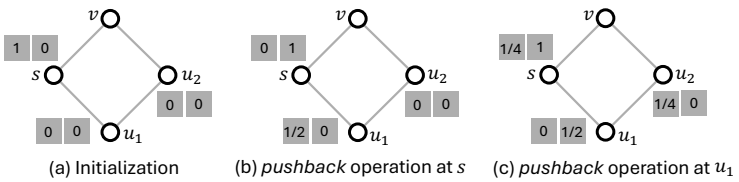


Fig. 1. Illustration of the *pushback* operation originated from s in BackPush. For each node, values denote $r(\cdot, s)$ (left) and $\hat{\pi}_v(\cdot, s)$ (right). (a) Initialization with all residuals concentrated at the source s. (b) A *pushback* operation at s adds its residual into estimates and propagates a fraction of $1/2$ to its neighbors, excluding the trap node v. (c) A subsequent *pushback* operation at u_1 processes its residual ($1/2$) by updating $\hat{\pi}_v(u_1, s)$ and distributing $1/4$ to each neighbor (s and u_2).

4.3 Theoretical Analysis

Correctness of BackPush. The *pushback* operation in BackPush maintains an invariant as stated in the following lemma. This invariant underpins the correctness of the algorithm.

Lemma 4.1. *(invariant) For each node $u \in V \setminus \{v\}$, the following invariant is maintained.*

$$\pi_v(u,x) = \hat{\pi}_v(u,x) + \sum\nolimits_{w \in V} r(w,x)\pi_v(u,w). \tag{10}$$

Proof. We prove this invariant by induction. Initially, $r(x,x) = 1$ and 0 otherwise, so $\pi_v(u,x) = 0 + \pi_v(u,x)$, and the invariant holds. Assume Eq. (10) holds before the *pushback* operation. We

now show it holds after the operation on node y. During this operation, $\hat{\pi}_v(y,x)$ increases by $r(y,x)$, $r(w,x)$ increases by $\frac{r(y,x)}{\boldsymbol{d}_w}$ for each $w \in \mathcal{N}(y), w \neq v$, and $r(y,x)$ becomes 0. Thus, after the *pushback* operation on y, the right-hand side of Eq. (10) becomes $\hat{\pi}(u,x) + r(y,x)\mathbb{I}_{y=u} + \sum_{w\in V} r(w,x)\pi_v(u,w) + \sum_{w\in\mathcal{N}(y)\setminus\{v\}} \frac{r(y,x)}{\boldsymbol{d}_w}\pi_v(u,w) - r(y,x)\pi_v(u,y)$. That is, this side increases by $r(y,x)\mathbb{I}_{y=u} - r(y,x)\pi_v(u,y) + \sum_{w\in\mathcal{N}(y)\setminus\{v\}} \frac{r(y,x)}{\boldsymbol{d}_w}\pi_v(u,w)$, which we can verify is 0. Hence, the invariant holds after the *pushback* operation, ending the proof. □

Complexity and Error Analysis. The following lemma states the performance of BackPush.

Lemma 4.2. *BackPush runs in* $O\Big(\sum_{x\in\{s,t,v\}}\sum_{y\in V}\frac{\pi_v(y,x)\boldsymbol{d}_y}{r_{\max}}\Big)$ *time. The additive error between the true value* $\pi_v(u,x)$ *and its approximation* $\hat{\pi}_v(u,x)$ *for* $x \in \{s,t,v\}$ *is bounded by* $\sum_{w\in V}\pi_v(u,w)r_{\max}$.

Proof. We begin by proving the time complexity. The time complexity of BackPush is dominated by performing *pushback* operations. Each *pushback* operation on node y increases $\hat{\pi}_v(y,x)$ by at least $r_{\max}$ at the cost of $O(\boldsymbol{d}_y)$ time since $r(w,x)$ is updated for each neighbor w of node y. Therefore, $\frac{r_{\max}}{\boldsymbol{d}_y}$ can be viewed as the "unit cost-benefit" of the *pushback* operation in terms of increasing $\hat{\pi}_v(y,x)$ which is upper bounded by $\pi_v(y,x)$. By considering each node $y \in V$ separately, we attain the stated total complexity.

For the approximate error, according to Eq. (10), $\pi_v(u,x) = \hat{\pi}_v(u,x) + \sum_{w\in V} r(w,x)\pi_v(u,w)$. In Algorithm 1, $r(y,x)$ is bounded by $r_{\max}$, thus the error can be bounded by $\sum_{w\in V}\pi_v(u,w)r_{\max}$. □

5 Algorithm via Random Walks

In Section 4, the proposed *pushback* operation-based algorithm essentially approximates $\pi_v(u,s)$, $\pi_v(u,t)$, and $\pi_v(u,v)$ for all $u \in V\setminus\{v\}$. Since these quantities represent fundamental random walk statistics, they naturally suggest a Monte Carlo alternative. In this section, we propose FastWalk, a heuristic Monte Carlo algorithm guided by a novel interpretation of BD via *collisions* between absorbing random walks.

5.1 Novel Interpretations

A straightforward Monte-Carlo extension of BackPush would require estimating the expected visit counts $\pi_v(u,x)$ (where $x \in \{s,t,v\}$) for every node $u \in V\setminus\{v\}$. Such an approach is computationally prohibitive, as it would necessitate initiating random walks from $\Theta(n)$ source nodes to estimate these quantities individually. To circumvent this, we employ a rigorous algebraic transformation. Starting from our pivot-based formulation and the absorbing random-walk representation of $\boldsymbol{L}_v^{-1}$, we rewrite each term in Eq. (8) as a double series. By carefully reindexing these series, we derive novel collision-based estimators.

Another challenge involves the terms involving $\boldsymbol{a}$ (i.e., $\boldsymbol{b}_{st}^\top\boldsymbol{L}_v^{-2}\boldsymbol{a}$ and $\boldsymbol{a}^\top\boldsymbol{L}_v^{-2}\boldsymbol{a}$). A naïve absorbing-walk interpretation would require initiating walks from all neighbors in $\mathcal{N}(v)$, which becomes intractable when the degree $|\mathcal{N}(v)|$ is large. We instead derive a formulation that utilizes walks starting from the single pivot v (by treating the first step to a neighbor as part of the process), making the collision-based estimator practical.

We derive the novel interpretations by expanding $\boldsymbol{L}_v^{-2}$ using the absorbing random-walk identity stated in Section 4.1. Recall that for any $x, y \in V\setminus\{v\}$, the inverse Laplacian relates to random walks as $(\boldsymbol{L}_v^{-1})_{xy} = \sum_{k=0}^{\infty}(\boldsymbol{P}_v^k)_{xy}/\boldsymbol{d}_y$ [82]. The first term $\boldsymbol{b}_{st}^\top\boldsymbol{L}_v^{-2}\boldsymbol{b}_{st}$ can be rewritten as

$$\begin{aligned}\boldsymbol{b}_{st}^\top\boldsymbol{L}_v^{-2}\boldsymbol{b}_{st} &= \sum\nolimits_{u=1}^{n-1}\Big(\sum\nolimits_{k=0}^{\infty}((\boldsymbol{P}_v^k)_{su} - (\boldsymbol{P}_v^k)_{tu})/\boldsymbol{d}_u\Big)^2\\ &= \sum\nolimits_{i=0}^{\infty}\sum\nolimits_{j=0}^{\infty} f^{i,j}(s,s) + f^{i,j}(t,t) - f^{i,j}(s,t) - f^{i,j}(t,s).\end{aligned}$$

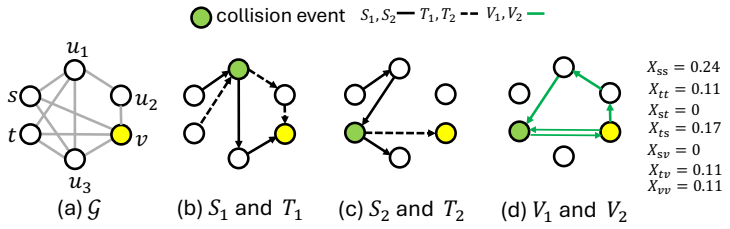


Fig. 2. Illustration of FastWalk where we simply set $l = 3$. The rightmost panel lists the estimators derived by accumulating the degree-normalized collisions. For instance, $X_{ss} = 0.24$ arises from the collisions between walks S_1 and S_2 at nodes s, u_1, u_3, calculated as $1/3^2 + 1/4^2 + 1/4^2 \approx 0.24$.

where $f^{i,j}(x,y) = \sum_{u=1}^{n-1} (\boldsymbol{P}_v^i)_{xu} (\boldsymbol{P}_v^j)_{yu} / \boldsymbol{d}_u^2$. The quantity $f^{i,j}(x,y)$ represents the degree-normalized probability that two absorbing random walks of lengths i and j respectively, starting from nodes x and y, collide at some node.

The numerator of the second term in Eq. (8) is the square of $\boldsymbol{b}_{st}^\top \boldsymbol{L}_v^{-2} \boldsymbol{a}$ which can be rewritten as

$$\begin{aligned}
\boldsymbol{b}_{st}^\top \boldsymbol{L}_v^{-2} \boldsymbol{a} &= \sum_{u=1}^{n-1} \frac{1}{\boldsymbol{d}_u^2} \sum_{k=0}^{\infty} ((\boldsymbol{P}_v^k)_{su} - (\boldsymbol{P}_v^k)_{tu}) \sum_{a \in \mathcal{N}(v)} \sum_{k=0}^{\infty} (\boldsymbol{P}_v^k)_{au} \\
&= \sum_{u=1}^{n-1} \frac{\boldsymbol{d}_v}{\boldsymbol{d}_u^2} \sum_{k=0}^{\infty} ((\boldsymbol{P}_v^k)_{su} - (\boldsymbol{P}_v^k)_{tu}) \sum_{k=0}^{\infty} \sum_{a \in \mathcal{N}(v)} \boldsymbol{P}_{va} (\boldsymbol{P}_v^k)_{au} \\
&= \boldsymbol{d}_v \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} f^{i,j}(s) - f^{i,j}(t),
\end{aligned}$$

where $f^{i,j}(s) = \sum_{u=1}^{n-1} (\boldsymbol{P}_v^i)_{su} \sum_{a \in \mathcal{N}(v)} \boldsymbol{P}_{va} (\boldsymbol{P}_v^j)_{au} / \boldsymbol{d}_u^2$ represents the degree-normalized probability that two random walks collide at some node, where the first is an absorbing random walk starting from node s with maximum length i, and the second is a random walk starting from node v, taking one step to a neighbor, and then continuing as an absorbing random walk for up to j steps.

For the denominator $\boldsymbol{a}^\top \boldsymbol{L}_v^{-2} \boldsymbol{a}$, we can rewrite it as

$$\begin{aligned}
\boldsymbol{a}^\top \boldsymbol{L}_v^{-2} \boldsymbol{a} &= \sum_{u=1}^{n-1} \left(\sum_{a \in \mathcal{N}(v)} \sum_{k=0}^{\infty} (\boldsymbol{P}_v^k)_{au} \right)^2 / \boldsymbol{d}_u^2 \\
&= \sum_{u=1}^{n-1} \left(\sum_{a \in \mathcal{N}(v)} \boldsymbol{d}_v \boldsymbol{P}_{va} \sum_{k=0}^{\infty} (\boldsymbol{P}_v^k)_{au} \right)^2 / \boldsymbol{d}_u^2 \\
&= \boldsymbol{d}_v^2 \sum_{i,j=0}^{\infty} \sum_{u=1}^{n=1} \sum_{a \in \mathcal{N}(v)} \boldsymbol{P}_{va} \sum_{b \in \mathcal{N}(v)} \boldsymbol{P}_{vb} \frac{(\boldsymbol{P}_v^i)_{au} (\boldsymbol{P}_v^j)_{bu}}{\boldsymbol{d}_u^2}.
\end{aligned}$$

The quantity following the first summation sign represents the degree-normalized probability that two random walks starting from node v, collide at some node. In such walks, the first step is a simple random walk, and from the second step onward, they become absorbing random walks with maximum lengths i and j, respectively.

5.2 The FastWalk Algorithm and Analysis

Based on the interpretations of the terms in the new BD formulation, we can sample ω_w absorbing random walks (illustrated in Figure 2) with v as the trap node and estimate the corresponding quantities by counting collisions between random walks. The following lemma demonstrates that we can derive novel unbiased estimators for the quantities of interest.

Lemma 5.1. *Given a graph $\mathcal{G}$, a trap node v, and query nodes $s \neq t$, consider two absorbing random walks S_1 and S_2 starting from s, two absorbing random walks T_1 and T_2 starting from t, and two*

absorbing random walks V_1 and V_2 starting from v. Define the variables: X_{ss}, X_{tt}, X_{st}, X_{ts}, X_{sv}, X_{tv}, and X_{vv}, as the number of degree-normalized collisions between walks starting from the nodes indicated by their subscripts (see Fig. 2). This yields the following unbiased estimators: $X_{ss} + X_{tt} - X_{st} - X_{ts}$ estimates $\boldsymbol{b}_{st}^\top \boldsymbol{L}_v^{-1} \boldsymbol{b}_{st}$, $\boldsymbol{d}_v(X_{sv} - X_{tv})$ estimates $\boldsymbol{b}_{st}^\top \boldsymbol{L}_v^{-2} \boldsymbol{a}$, and $\boldsymbol{d}_v^2 X_{vv}$ estimates $\boldsymbol{a}^\top \boldsymbol{L}_v^{-1} \boldsymbol{a}$.

Proof. By the linearity of expectation, we have $\mathbb{E}(X_{ss} + X_{tt} - X_{st} - X_{ts}) = \mathbb{E}(X_{ss}) + \mathbb{E}(X_{tt}) - \mathbb{E}(X_{st}) - \mathbb{E}(X_{ts}) = \boldsymbol{b}_{st}^\top \boldsymbol{L}_v^{-1} \boldsymbol{b}_{st}$, $\mathbb{E}(\boldsymbol{d}_v(X_{sv} - X_{tv})) = \boldsymbol{d}_v \mathbb{E}(X_{sv}) - \boldsymbol{d}_v \mathbb{E}(X_{tv}) = \boldsymbol{b}_{st}^\top \boldsymbol{L}_v^{-2} \boldsymbol{a}$, and $\mathbb{E}(\boldsymbol{d}_v^2 X_{vv}) = \boldsymbol{d}_v^2 \mathbb{E}(X_{vv}) = \boldsymbol{a}^\top \boldsymbol{L}_v^{-1} \boldsymbol{a}$. □

Algorithm 2: FastWalk($\mathcal{G}, v, s, t, \omega_w, l$)

Input : A graph $\mathcal{G} = (V, E)$; a node v; two query nodes s and t; the sampling size ω_w; the truncation length l

Output : Estimated BD $\beta'(s,t)$

$X_{ss} = 0; X_{tt} = 0; X_{st} = 0; X_{ts} = 0; X_{sv} = 0; X_{tv} = 0; X_{vv} = 0$

for $i = 1 : \omega_w$ **do**

  if $s \neq v$ **then** Perform two random walks $S_{i,1}$ and $S_{i,2}$ from s before they reach v or their lengths are l

  if $t \neq v$ **then** Perform two random walks $T_{i,1}$ and $T_{i,2}$ from t before they reach v or their lengths are l

  Perform two random walks $V_{i,1}$ and $V_{i,2}$ from v, starting as a simple random walk, and from the second step, check if they reach v or exceed length l

  for $x \in S_{i,1}, y \in S_{i,2}$ **do**

    if $x = y$ **then** $X_{ss} +=$ 1 with probability $1/\boldsymbol{d}_x^2$

  for $x \in T_{i,1}, y \in T_{i,2}$ **do**

    if $x = y$ **then** $X_{tt} +=$ 1 with probability $1/\boldsymbol{d}_x^2$

  for $x \in S_{i,1}, y \in T_{i,2}$ **do**

    if $x = y$ **then** $X_{st} +=$ 1 with probability $1/\boldsymbol{d}_x^2$

  for $x \in T_{i,1}, y \in S_{i,2}$ **do**

    if $x = y$ **then** $X_{ts} +=$ 1 with probability $1/\boldsymbol{d}_x^2$

  for $x \in S_{i,1}, y \in V_{i,2}$ **do**

    if $x = y$ **then** $X_{sv} +=$ 1 with probability $1/\boldsymbol{d}_x^2$

  for $x \in T_{i,1}, y \in V_{i,2}$ **do**

    if $x = y$ **then** $X_{tv} +=$ 1 with probability $1/\boldsymbol{d}_x^2$

  for $x \in V_{i,1}, y \in V_{i,2}$ **do**

    if $x = y$ **then** $X_{vv} +=$ 1 with probability $1/\boldsymbol{d}_x^2$

$\beta'(s,t) = \frac{X_{ss} + X_{tt} - X_{st} - X_{ts}}{\omega_w} - \frac{\boldsymbol{d}_v^2 (X_{sv} - X_{tv})^2}{\omega_w^2 + \omega_w \boldsymbol{d}_v^2 X_{vv}}$

return $\beta'(s,t)$

The expected length of the above random walks from s is $l_{\text{exp}} = h_{s,v}$, where $h_{s,v}$ is the hitting time [13, 44], defined as the expected number of steps for a random walk starting from node s to reach node v for the first time. In networks with scale-free [7] and small-world [69] properties, the length of such random walks can become prohibitively large, scaling at least sub-linearly with n [38, 58], which is inefficient for large-scale networks. One approach to mitigate this is to truncate the random walks at a predefined length l, as done in [56, 83]. However, truncation introduces a

trade-off: shorter walks reduce computation but omit long-range collision information. This raises the key question: how can we balance the maximum walk length l and the number of invalid walks (those that don't reach the trap node within l)? The following lemma guarantees that any threshold $\alpha > 0$ for invalid walk ratio can be achieved through appropriate l selection.

Lemma 5.2. *[83] Given a graph $\mathcal{G} = (V, E)$, a node $v \in V$, and a ratio $\alpha > 0$, if the maximum length l of random walks satisfies $l = \log(m\alpha/\sqrt{n-1}\,\|\boldsymbol{d}_{-v}\|_2)/\log(\lambda)$, where λ is the spectral radius of matrix $\boldsymbol{P}_v$, then the expected ratio of invalid walks is less than α.*

Building on Lemmas 5.1 and 5.2, we introduce the FastWalk algorithm, with its pseudocode provided in Algorithm 2. The algorithm runs in ω rounds and in each round, it generates l-truncated absorbing walks from nodes s, t, and v respectively (Lines 3-5). Through iterative pairwise node comparisons between these walks (Lines 6-19), it then accumulates degree-normalized collision counts for computing the BD estimator $\beta'(s,t)$ (Line 20). Special handling occurs when s or t coincides with v: the corresponding walk sets ($S_{i,1}, S_{i,2}$ or $T_{i,1}, T_{i,2}$) will be empty, which automatically skips the associated computations in Lines 6-17.

Complexity Analysis. Following the steps in FastWalk, it is straightforward to derive its time complexity as follows.

Lemma 5.3. *FastWalk's time complexity is $O(\omega_w \cdot \min(6l, 2h_{s,v} + 2h_{t,v} + 2\sum_{u\in N(v)} h_{u,v}))$.*

Proof. In each of the ω_w iterations the algorithm does two things: (i) it generates six truncated/absorbed random walks (two from s, two from t, two starting at v), and (ii) it counts intersections for seven fixed pairs of these walks. Both parts run in time proportional to the total length of the six walks in that iteration.

Deterministically, each walk has length at most l, so one iteration costs $O(6l)$. Alternatively, the expected total length in one iteration is bounded by the sum of two hitting times from s to v, two from t to v, and two from a neighbor of v back to v, yielding the hitting-time bound. Multiplying by ω_w iterations ends the proof. □

6 Algorithm via Spanning trees

In this section, we establish the connection between BD and spanning trees and propose an algorithm, FastTree, which uniformly samples spanning trees and information encoded in these spanning trees is then leveraged to approximate BD.

6.1 A Tree-based Estimator

As an initial step, we represent the elements of the matrix $\boldsymbol{L}_v^{-1}$ using spanning trees, as shown as follows.

Lemma 6.1. *Given an edge (a, b), denote by $N_s^v(a, b)$ the number of spanning trees of $\mathcal{G}$ in which the unique path from s to v contains a and b, in this order. Define $N_s^v(b, a)$ analogously and write N for the total number of spanning trees in the whole graph. Let $\mathcal{P}_u^v$ denote an arbitrary path from u to v in the graph. Then we have*

$$(\boldsymbol{L}_v^{-1})_{su} = \sum\nolimits_{(a,b)\in\mathcal{P}_u^v} \frac{N_s^v(a,b) - N_s^v(b,a)}{N} \tag{11}$$

Proof. We prove this lemma by treating the network as an electrical circuit where all edges in E are considered as unit resistors, and nodes in V are viewed as junctions between resistors. We then introduce a third quantity, the voltage v_u of node u, and demonstrate that both sides of Eq. (11) equal v_u. Firstly, according to [57], when unit flow comes in through node s, and unit flow

comes out through node v, the current flow on edge (a, b) is

$$c_{ab} = \frac{N_s^v(a,b) - N_s^v(b,a)}{N}. \tag{12}$$

Based on the Kirchhoff's law [30], the voltage of node u can be computed through the current flow along a path from u to the "0" voltage node v. Specifically, for a considered network, the voltage v_u of node u can be computed as $v_u = \sum_{(a,b)\in\mathcal{P}_u^v} c_{ab}$. Plugging Eq. (12) into this equation, we can drive that the right-hand side of Eq. (11) is equal to v_u. Secondly, according to [11], $(\boldsymbol{L}_v^{-1})_{su}$ is the voltage at node $u \in V$ when unit flow comes in through node s, and unit flow comes out through node v. This completes the proof. □

Based on the above lemma, we next express $\beta(s, t)$ using spanning trees in the following natural corollary.

Corollary 6.2. *Let $\boldsymbol{x}^a = \boldsymbol{a}^\top L_v^{-1}$ and $\boldsymbol{x}^b = \boldsymbol{b}_{st}^\top \boldsymbol{L}_v^{-1}$, we have that*

$$\beta(s,t) = (\boldsymbol{x}^b)^\top \boldsymbol{x}^b - \frac{((\boldsymbol{x}^b)^\top \boldsymbol{x}_a)^2}{1 + (\boldsymbol{x}^a)^\top \boldsymbol{x}^a}.$$

For each node u, the components of $\boldsymbol{x}^a$ and $\boldsymbol{x}^b$ admit the following spanning-tree-based expressions

$$\boldsymbol{x}_u^a = \frac{1}{N}\sum\nolimits_{x\in\mathcal{N}(v)}\sum\nolimits_{(a,b)\in\mathcal{P}_u^v}(N_x^v(a,b) - N_x^v(b,a)),$$
$$\boldsymbol{x}_u^b = \frac{1}{N}\sum\nolimits_{(a,b)\in\mathcal{P}_u^v}(N_s^v(a,b) - N_s^v(b,a) - N_t^v(a,b) + N_t^v(b,a)).$$

Therefore, if we can obtain satisfactory approximations $\tilde{\boldsymbol{x}}_u^a$ and $\tilde{\boldsymbol{x}}_u^b$ for $\boldsymbol{x}_u^a$ and $\boldsymbol{x}_u^b$ respectively, we can thus obtain an accurate estimator of $\beta(s, t)$ as

$$\beta'(s,t) = (\tilde{\boldsymbol{x}}^b)^\top \tilde{\boldsymbol{x}}^b - \frac{((\tilde{\boldsymbol{x}}^b)^\top \tilde{\boldsymbol{x}}^a)^2}{1 + (\tilde{\boldsymbol{x}}^a)^\top \tilde{\boldsymbol{x}}^a}. \tag{13}$$

To achieve this, we can sample ω_t spanning trees. For each tree T_i, where $1 \le i \le \omega_t$, define $T_i^{s,v}(a, b)$ as a binary variable indicating whether the path from s to v in T_i passes through edge (a, b). Let $\tilde{\boldsymbol{x}}_u^{a_i}$ and $\tilde{\boldsymbol{x}}_u^{b_i}$ represent the accumulated visits to the edges along the path $\mathcal{P}_u^v$ (visualized as green paths in Figure 3), with $\tilde{\boldsymbol{x}}_u^{a_i}$ corresponding to paths from nodes in $\mathcal{N}(v)$ to v, and $\tilde{\boldsymbol{x}}_u^{b_i}$ corresponding to paths from nodes s and t to v:

$$\tilde{\boldsymbol{x}}_u^{a_i} = \sum\nolimits_{x\in\mathcal{N}(v)}\sum\nolimits_{(a,b)\in\mathcal{P}_u^v}(T_i^{x,v}(a,b) - T_i^{x,v}(b,a)),$$
$$\tilde{\boldsymbol{x}}_u^{b_i} = \sum\nolimits_{(a,b)\in\mathcal{P}_u^v} T_i^{s,v}(a,b) - T_i^{s,v}(b,a) - T_i^{t,v}(a,b) + T_i^{t,v}(b,a).$$

Then, the averages

$$\tilde{\boldsymbol{x}}_u^a = \frac{1}{\omega_t}\sum_{i=1}^{\omega_t}\tilde{\boldsymbol{x}}_u^{a_i} \text{ and } \tilde{\boldsymbol{x}}_u^b = \frac{1}{\omega_t}\sum_{i=1}^{\omega_t}\tilde{\boldsymbol{x}}_u^{b_i}$$

are *unbiased estimators* of $\boldsymbol{x}_u^a$ and $\boldsymbol{x}_u^b$, respectively, since $\mathbb{E}(\tilde{\boldsymbol{x}}_u^a) = \frac{1}{\omega_t}\sum_{i=1}^{\omega^t}\mathbb{E}(\tilde{\boldsymbol{x}}_u^{a_i}) = \boldsymbol{x}_u^a$ and $\mathbb{E}(\tilde{\boldsymbol{x}}_u^b) = \frac{1}{\omega_t}\sum_{i=1}^{\omega_t}\mathbb{E}(\tilde{\boldsymbol{x}}_u^{b_i}) = \boldsymbol{x}_u^b$.

In summary, the key idea of the FastTree algorithm is to sample a number ω_t of spanning trees and obtain the unbiased estimates as stated above. Finally, FastTree computes the approximated BD according to Eq. (13).

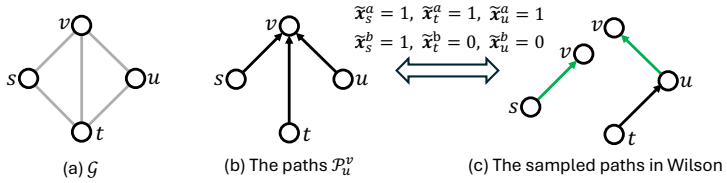


Fig. 3. Illustration of the main idea of FastTree. The green paths represent the traversed edges in paths $\{\mathcal{P}_u^v\}_{u\in\{s,t,\mathcal{N}(v)\}}$. For $\tilde{\boldsymbol{x}}_s^a$, we aggregate visits from neighbors $\mathcal{N}(v)$: since $(s,v) \in \mathcal{P}_s^v$ is traversed by the tree path from neighbor s (+1) but not by others, we have $\tilde{\boldsymbol{x}}_s^a = 1$. Similarly, for $\tilde{\boldsymbol{x}}_s^b$, the edge $(s,v) \in \mathcal{P}_s^v$ is traversed by the tree path from s (+1) but not t (0), yielding $1-0=1$. The other quantities are computed analogously.

6.2 Optimizations

We use the widely used Wilson algorithm [72] to sample random spanning trees. It begins by selecting a node v as the root of a spanning tree and then following a predefined node order $Order$, it performs a loop-erased random walk [33, 34]—a random walk where loops in the trajectory are erased—until it encounters a node already included in the tree. The loop-erased path is then added to the spanning tree. This process continues until all nodes are included in the tree. However, a direct application of the Wilson algorithm would bring computational challenges as stated below.

Our goal is to approximate all $n-1$ elements of $\boldsymbol{x}^a$ (or $\boldsymbol{x}^b$). Directly computing each element via path traversal requires $O(|\mathcal{N}(v)|\Delta^2)$ time per element, where Δ is the graph diameter, as paths $\mathcal{P}_u^v$ and paths in sampled trees can reach length Δ. This results in a total cost of $O(n|\mathcal{N}(v)|\Delta^2)$, which is infeasible for large networks. To address this, we introduce optimization techniques below.

Shortest-path Anchoring. First, we observe that shorter paths $\mathcal{P}_u^v$ offer two key advantages: (1) reduced computational overhead, and (2) minimized approximation errors. This is because shorter paths reduce the number of terms in $\tilde{\boldsymbol{x}}_u^a$ and $\tilde{\boldsymbol{x}}_u^b$, and as each term is an approximation, fewer edges in $\mathcal{P}_u^v$ lead to smaller cumulative errors. To achieve this, we construct a BFS tree T_B rooted at v (as shown in Figure 3 (b)), and for each node $u \in V \setminus \{v\}$, we define $\mathcal{P}_u^v$ as the unique path from u to v in T_B. This ensures that $\mathcal{P}_u^v$ is the shortest possible path for each node u and reduces storage requirements since fixed tree paths only require tracking $O(n)$ edges compared to $O(m)$ edges.

Overlap-aware Reuse of Paths on T_B. Second, we exploit the inherent path overlaps in T_B through strategic computational reuse. All paths $\{\mathcal{P}_u^v\}$ collectively span every edge in the BFS tree. Rather than processing each path independently, we first compute edge-wise inclusion counts across all relevant paths (from s, t, and $\mathcal{N}(v)$ nodes to v). These aggregated counts enable efficient derivation of path-specific quantities through summation, avoiding redundant computations for shared tree segments. This optimization significantly reduces the computational overhead associated with overlapping path segments.

Order-restricted Wilson Sampling. Third, we leverage the order invariance property of Wilson's algorithm [72] to optimize node traversal. By restricting the node sequence $Order$ to only include s, t, and $\mathcal{N}(v)$ nodes, we focus computational resources on generating loop-erased random walks from these critical nodes. This selective ordering preserves statistical validity while eliminating unnecessary computations for irrelevant nodes.

Reverse Accumulation with Counters. Fourth, we implement a reverse traversal mechanism with auxiliary counters to handle path overlaps within sampled trees. During Wilson's execution, we record loop-erased paths ($path[u]$) (see examples in Figure 3 (c)) for later analysis. For $\boldsymbol{x}^a$, we maintain: $vis_a(u)$: Node visit counter (initialized as 1 for nodes in $\mathcal{N}(v)$); $vis_a(e)$: Edge visit counter

for T_B edges. Traversing paths in reverse order, we propagate visit counts from leaf nodes toward v through parent relationships. For each edge (x, y) in $path[u]$, we update $vis_a(y)$ += $vis_a(x)$ and $vis_a(e)$ ±= $vis_a(x)$ where the edge operation depends on traversal direction. The final $\boldsymbol{x}_u^a$ values are obtained by summing $vis_a(e)$ along $\mathcal{P}_u^v$. Approximating $\boldsymbol{x}^b$ follows analogous procedures with different initialization ($vis_b(s) = 1, vis_b(t) = -1$). This approach eliminates redundant calculations for shared path segments.

Remark. Similar optimization ideas have appeared for other network metrics—most notably for effective resistance estimation [36]. Directly applying their method to BD estimation is non-trivial. The reason lies in the fact that BD is governed by the grounded Laplacian square and couples first-order visit counts with second-order, degree-normalized interactions between two walks. This changes both the variance profile and what can be precomputed and reused, so the primitives that help resistance distance are not directly applicable. Our key innovations include: (1) Enhanced utilization of BFS tree properties with path overlap analysis, enabling efficient information reuse across both BFS and sampled spanning trees; (2) A novel adaptive sampling strategy tailored for BD's unique characteristics, contrasting with [36]'s approach through selective tree partitioning. These adaptations specifically address BD's computational challenges while improving efficiency.

6.3 The FastTree Algorithm and Analysis

Based on the optimizations discussed above, we propose the FastTree algorithm, outlined in Algorithm 3. It begins by constructing a BFS tree T_B rooted at v and initializing an array Par to record the parent of each node in T_B (Lines 1-2). Next, we initialize the *visit* arrays (Lines 3-4). The algorithm then samples ω_t spanning trees using the Wilson algorithm. For each sampled tree, we record the loop-erased path of each node in array $path[u]$ and store the outgoing node of each node u in array $next(u)$ (Line 7). We then traverse the loop-erased paths in reverse order of their generation and check whether they pass through edges in T_B and update *visit* frequencies for corresponding edges, using the *next* and *Par* arrays (Lines 8-19). We then process the nodes in $V \setminus \{v\}$ following the order of their appearance in T_B. For each node u, we compute $\tilde{\boldsymbol{x}}_u^b$ and $\tilde{\boldsymbol{x}}_u^a$ by summing the *visit* counts of the edges from u to the root v, as specified in Corollary 6.2 (Lines 20-24). Finally, the algorithm computes $\beta'(s, t)$ as the output (Line 25).

Theoretical Analysis. The following theorem summarizes the performance of FastTree.

Theorem 6.3. *For an error parameter $\epsilon \in (0, 1)$ and a failure probability $\delta \in (0, 1)$, if $\omega_t = \frac{\boldsymbol{d}_v^2 \Delta^2 \log \frac{2n}{\delta}}{2\epsilon^2}$ where Δ is the graph diameter, FastTree runs in $O(\omega_t \cdot (h_{s,v} + h_{t,v} + \sum_{a \in \mathcal{N}(v)} h_{a,v}))$ time by sampling ω_t trees and the corresponding estimators satisfy that for each node u, $|\boldsymbol{x}_a(u) - \tilde{\boldsymbol{x}}_a(u)| \leq \epsilon$ and $|\boldsymbol{x}_b(u) - \tilde{\boldsymbol{x}}_b(u)| \leq \epsilon$.*

To prove the above theorem, we make use of the following theorem.

Theorem 6.4. *(Chernoff-Hoeffding's Inequality [26]). Let $Z_1, Z_2, \ldots, Z_{n_z}$ be i.i.d. random variables with Z_j ($\forall 1 \leq j \leq n_z$) being strictly bounded by the interval $[a_j, b_j]$. We define the mean of these variables by $Z = \frac{1}{n_z} \sum_{j=1}^{n_z} Z_j$. Then, we have $\mathbb{P}[|Z - \mathbb{E}[Z]| \geq \varepsilon] \leq 2 \exp\left(- \frac{2n_z^2 \varepsilon^2}{\sum_{j=1}^{n_z} (b_j - a_j)^2} \right)$.*

Proof. We next prove the estimation error. For a random spanning tree T_i, It follows that $|\tilde{\boldsymbol{x}}_{a_i}| \leq \boldsymbol{d}_v \Delta$ and $|\tilde{\boldsymbol{x}}_{b_i}| \leq 2\Delta$. Next, applying the Chernoff-Hoeffding's Inequality [26], if $\omega_t \geq \frac{\boldsymbol{d}_v^2 \Delta^2 \log\left(\frac{2n}{\delta}\right)}{2\epsilon^2}$, we can bound the probability that the mean of $\tilde{\boldsymbol{x}}_{a_i}(u)$ deviates from its expectation by more than ϵ

$$\mathbb{P}\left(\left| \frac{1}{\omega_t} \sum_{i=1}^{\omega_t} \tilde{\boldsymbol{x}}_{a_i}(u) - \boldsymbol{x}_a(u) \right| \geq \epsilon \right) \leq \frac{\delta}{n}.$$

Algorithm 3: FastTree($\mathcal{G}, v, s, t, \omega_t$)

Input : A graph $\mathcal{G} = (V, E)$; a node v; two query nodes s and t; the sampling size ω_t
Output : Estimated BD $\beta'(s,t)$

Construct a BFS tree T_B with root v
$Par(u) \leftarrow$ the parent node of node u in T_B for $u \in V \setminus \{v\}$
$vis_a(x) = 1\ \forall x \in \mathcal{N}(v), vis_b(s) = 1, vis_b(t) = -1$
$vis_a(x,y) = 0, vis_b(x,y) = 0\ \forall (x,y) \in T_B$
$Order \leftarrow \{s,t\} \cup \mathcal{N}(v)$ // Only keep relevant nodes
for $i = 1 : \omega_t$ **do**
  Execute the Wilson algorithm following a fixed node order $Order$. $next$ stores the tree structure by recording the next node for each node. $path[u]$ records the loop-erased path starting from node u
  for $u \in reverse(Order)$ **do**
    for $x \in path[u]$ **do**
      $y = next[x]$
      $vis_a(y) = vis_a(y) + vis_a(x)$
      $vis_b(y) = vis_b(y) + vis_b(x)$
      if $Par(x) = y$ **then**
        $vis_a(x,y) = vis_a(x,y) + vis_a(x)$
        $vis_b(x,y) = vis_b(x,y) + vis_b(x)$
      else if $Par(y) = x$ **then**
        $vis_a(x,y) = vis_a(x,y) - vis_a(x)$
        $vis_b(x,y) = vis_b(x,y) - vis_b(x)$
$\tilde{\boldsymbol{x}}^a = \boldsymbol{0}^{(n-1)\times 1}, \tilde{\boldsymbol{x}}^b = \boldsymbol{0}^{(n-1)\times 1}$
for $u \in T_B$ **do**
  $x = Par(u)$
  $\tilde{\boldsymbol{x}}^a_u = \tilde{\boldsymbol{x}}^a_x + vis_a(u,x)/\omega_t$
  $\tilde{\boldsymbol{x}}^b_u = \tilde{\boldsymbol{x}}^b_x + vis_b(u,x)/\omega_t$
$\beta'(s,t) = \tilde{\boldsymbol{x}}_b^\top \tilde{\boldsymbol{x}}_b - (\tilde{\boldsymbol{x}}_b^\top \tilde{\boldsymbol{x}}_a)^2/(1 + \tilde{\boldsymbol{x}}_a^\top \tilde{\boldsymbol{x}}_a)$
return $\beta'(s,t)$

By a union bound, the probability that this holds for all nodes $u \in V$ is at least $1 - \delta$. Similarly, if $\omega_t \geq \frac{2\Delta^2 \log\left(\frac{2n}{\delta}\right)}{\epsilon^2}$, we can bound the probability that the average of $\tilde{\boldsymbol{x}}_{b_i}(u)$ deviates from its expectation

$$\mathbb{P}\left(\left|\frac{1}{\omega_t}\sum_{i=1}^{\omega_t} \tilde{\boldsymbol{x}}_{b_i}(u) - \boldsymbol{x}_b(u)\right| \geq \epsilon\right) \leq \frac{\delta}{n}.$$

Again, by a union bound, the probability that this holds for all nodes $u \in V$ is at least $1 - \delta$.

For the running time, to draw a sample, Algorithm 3 takes $O(h_{s,v})$ and $O(h_{t,v})$ to run a loop-erased random walk from respectively s and t to v. Then, it takes $O(\sum_{a\in\mathcal{N}(v)} h_{a,v})$ time to simulate such random walks from the neighbors to v. As a result, the total time to draw ω_t samples is in $O(\omega_t(h_{s,v} + h_{t,v} + \sum_{a\in\mathcal{N}(v)} h_{a,v}))$. □

Table 2. Some statistics of the studied real-world networks.

Network	#nodes (n)	#edges (m)	diameter (Δ)
Facebook	4,039	88,234	8
ca-GrQc	4,158	13,428	17
PowerGrid	4,941	6,594	46
Road-PA	1,087,562	1,541,514	786
Youtube	1,134,890	2,987,624	24
Orkut	3,072,441	117,185,083	10
soc-Livejournal	3,997,962	34,681,189	16
hetero-Livejournal	12,678,882	160,995,330	17

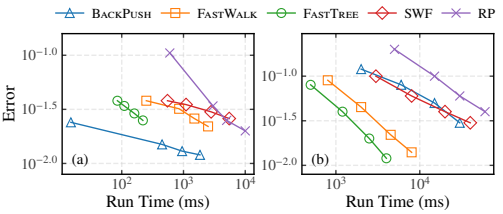


Fig. 4. Validation of ground truth approximation on two real-world networks: (a) ca-GrQc, (b) PowerGrid.

7 Experiments

7.1 Experimental Setting

Datasets and Queries. To evaluate the efficiency and accuracy of our proposed algorithms, namely BackPush, FastWalk and FastTree, we conducted experiments on 6 real-world networks from SNAP [35], details of which are provided in Table 2. For each network, we randomly pick 1000 node pairs as the query set.

Implementation Details. All experiments were run on a Linux machine with an Intel Xeon(R) Gold 6240@2.60GHz 32-core processor and 256GB of RAM. Before timing, the graph data is loaded into memory. All algorithms are implemented in Julia for a fair comparison. Results are excluded if the algorithm does not return within 24 hours. Experiments are conducted on the largest connected component of each network. To ensure a fair comparison that reflects intrinsic algorithmic efficiency rather than implementation optimization, we executed all methods using a single thread. For all our algorithms, we choose the highest-degree node as the pivot node. Our code is publicly available at https://github.com/biharmonic.

Competitors and Ground Truth. To demonstrate the superiority of proposed algorithms, we compare them with the SOTA algorithm SWF [42]. Additionally, we consider the RP algorithm from [76–78]. For SWF and RP, we use the open-sourced code from the authors. For the Facebook dataset, we compute exact BDs by inverting $\boldsymbol{L}$ as the ground truth. However, for larger-scale real-world graphs, obtaining the exact BD values using existing methods is computationally infeasible. Thus, following popular works [14, 24, 42, 75] and given the high accuracy of BackPush and FastTree, as will be demonstrated in Figures 5, 4, and 8, we use conservative parameter choices FastTree (ω_t = 1e8) output as ground truth for Road-PA, and BackPush ($r_{\max}$ = 1e−8) for BD in other real-world networks.

Evaluation Measures. We measure the mean relative errors of 1000 queries to evaluate approximation *accuracy*. We use the wall-clock running time in milliseconds to evaluate the algorithm *efficiency*.

7.2 Results on Real-world Networks

We carefully set the algorithm parameters to ensure reasonable runtime. For BackPush, $r_{\max}$ is varied in {1e−4, 5e−4, 1e−5, 1e−6}, except for Road-PA where it is {5e−3, 1e−4, 5e−4, 1e−5}. For FastWalk, the sample size ω_w is varied in {1e3, 5e3, 1e4, 5e4} with a truncation length of 1e5. For FastTree, ω_t is also varied in {1e3, 5e3, 1e4, 5e4}. For RP and SWF, the error parameter ϵ is set in {0.01, 0.02, 0.05, 0.1}, except for hetero-Livejournal where SWF uses {0.04, 0.06, 0.08, 0.1}.

To validate the rationality of using FastTree and BackPush as the ground truth, we first conducted experiments on two medium-sized networks, ca-GrQc and PowerGrid, where exact BD is computable. As shown in Figure 4, tightening the algorithm parameters increases runtime

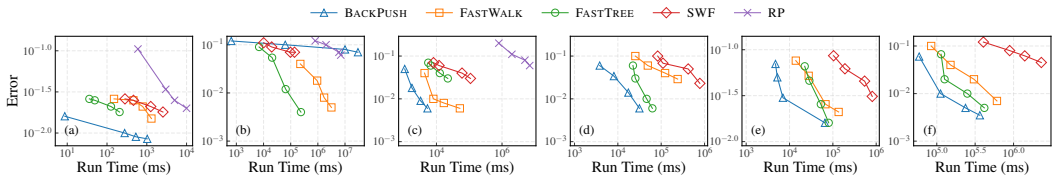


Fig. 5. Comparison of algorithms on six real-world networks: (a) Facebook, (b) Road-PA, (c) Youtube, (d) Orkut, (e) soc-Livejournal and (f) hetero-Livejournal.

yet consistently yields lower relative errors, with BackPush excelling on the small-diameter network (ca-GrQc) and FastTree on the large-diameter network (PowerGrid). These convergence behaviors, consistent with the exact results on Facebook (Figure 5 (a)) and the model networks in Figure 8, empirically justify our choice of using these algorithms with conservative parameters as high-precision references of ground truth for large graphs.

We next investigate the trade-off between approximation accuracy and computational efficiency. The results are shown in Figure 5. The results for RP is not included for some networks because it doesn't finish before the threshold of 24 hours. We observe that on all six datasets, the proposed algorithms all demonstrate better efficiency (smaller run time) and accuracy (smaller error value) in most scenarios in comparison to SWF and RP. This demonstrates the effectiveness of our pivot node based algorithms. Among these algorithms, BackPush obtains the best performance. For example, on soc-Livejournal, BackPush can achieve a relative error 0.03 in 7 seconds, while the SOTA method SWF takes 810 seconds. Thus, our algorithm is up to 115 times faster than the SOTA methods. BackPush excels on social networks because it relies on sparse, local push operations. On these highly connected, small-diameter graphs, the pivot efficiently absorbs residuals, allowing the algorithm to prune computations early via the threshold $r_{\max}$. In contrast, FastWalk and FastTree outperform BackPush on the road network Road-PA. This is because, on road networks, the hitting time from a node u to the pivot node v is relatively large, making the *pushback* operation inefficient.

7.3 Results on Model Networks

We further demonstrate the performance of our proposed algorithms by their study on some model (synthetic) networks. One advantage of such model graphs is that the exact value of BD can be derived as a closed-form formula. They also allow us to study our algorithms on networks with a wider spectrum of graph properties.

First, we consider a star graph consisting of an internal node and $n-1$ leaves. It is known [78] that in a star network $\mathcal{G}$, the BD between the internal node and a leaf is equal to $(n-1)/n$ and the BD between any two leaves is 2.

We also consider two types of tree-like structures introduced in [81]: non-fractal scale-free networks and fractal scale-free networks. The first type exhibits "small-world" phenomenon, while the second type shows "large-world" phenomenon [81]. Both networks grow deterministically and exhibit a tree-like structure. The g-th generation of these networks, denoted as $\mathcal{T}_g$ and $\mathcal{F}_g$, has $(2q+1)^g+1$ and $(2q+2)^g+1$ nodes, respectively, where q is a positive integer. The detailed building process for these network types is as follows.

Building Process of the Scale-free Tree Models. The recursive non-fractal scale-free trees, denoted by $\mathcal{T}_g$ after $g \geqslant 0$ generation evolution, are constructed as follows [29]. For $g=0$, $\mathcal{T}_0$ is an edge connecting two nodes. For $g \geqslant 1$, $\mathcal{T}_g$ is obtained from $\mathcal{T}_{g-1}$: for each of the existing edges in $\mathcal{T}_{g-1}$, we introduce $2q$ (q is a positive integer) new nodes; half of them are connected to one end of the edge, and half of them are linked to the other end. That is, we obtain $\mathcal{T}_g$ from $\mathcal{T}_{g-1}$ via

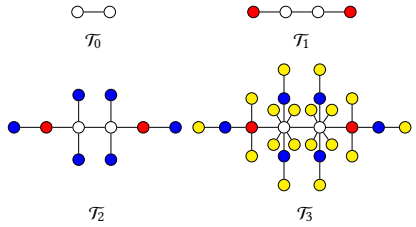


Fig. 6. Iterations of the non-fractal scale-free trees.

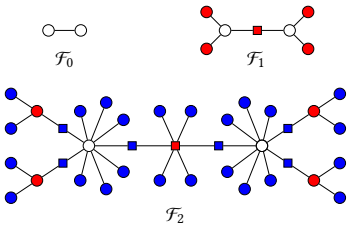


Fig. 7. Iterations of the fractal scale-free trees.

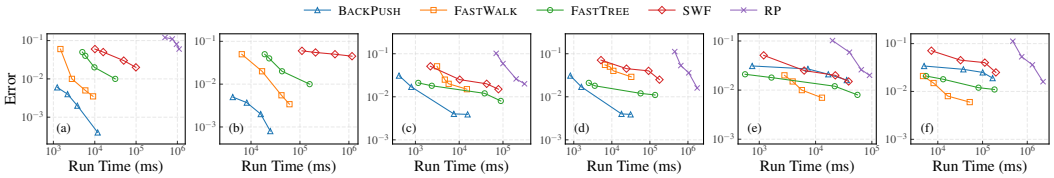


Fig. 8. Comparison of algorithms on six model networks: (a) Star-1e6, (b) Star-1e7, (c) $\mathcal{T}_{12}$, (d) $\mathcal{T}_{13}$, (e) $\mathcal{F}_7$ and (f) $\mathcal{F}_8$.

replacing every edge in $\mathcal{T}_{g-1}$ by the cluster. The construction process for the particular case of $q = 1$ is shown in Figure 6. By construction, $\mathcal{T}_g$ has $(2q + 1)^g + 1$ nodes.

The fractal scale-free networks, denoted by $\mathcal{F}_g$ after $g \geqslant 0$ iterations, are constructed in the following way [55, 60]. For $g = 0$, $\mathcal{F}_0$ consists of two nodes connected by an edge. For $g \geqslant 1$, $\mathcal{F}_g$ is obtained from $\mathcal{F}_{g-1}$ by performing the following operations on every edge in $\mathcal{F}_{g-1}$: replace the edge by a path of 2 edges long, with the two endpoints of the path being the same endpoints of the original edge (the new node having an initial degree 2 in the middle of the path is referred to as an internal node); then attach q new nodes with the initial degree of 1 (called external nodes) to each endpoint of the path. Figure 7 illustrates the construction process for a limiting case of $q = 2$, showing the first two iterative processes. By construction, $\mathcal{F}_g$ has $(2q + 2)^g + 1$ nodes.

Consider a cut edge $e = (s, t) \in E$ in a network $\mathcal{G}$ and let S and T be the node sets of the two connected components formed by removing this cut edge. It is known [9] that the BD between s and t is given by $\beta(s, t) = \frac{|S||T|}{n}$. This allows us to easily compute BD for the graphs $\mathcal{T}_g$ and $\mathcal{F}_g$ since they have a tree-like structure.

We study the following 6 networks: Star-1e6, Star-1e7, $\mathcal{T}_{12}$, $\mathcal{T}_{13}$, $\mathcal{F}_7$, $\mathcal{F}_8$ where Star-x means a star network with x nodes. Furthermore, for $\mathcal{T}_g$, we set $q = 1$, while for $\mathcal{F}_g$, we set $q = 2$. Again, we carefully set the algorithm parameters. For BackPush, $r_{\max}$ is varied in {1e−4, 5e−4, 1e−5, 1e−6}, except for $\mathcal{F}_7$ and $\mathcal{F}_8$ where it is {5e−3, 1e−4, 5e−4, 1e−5}. For FastWalk, the sample size ω_w is varied in {1e3, 5e3, 1e4, 5e4} with a truncation length of 1e5. For FastTree, ω_t is also varied in {1e3, 5e3, 1e4, 5e4}. For RP and SWF, the error parameter ϵ is set in {0.01, 0.02, 0.05, 0.1}, except for $\mathcal{T}_{13}$ where SWF uses {0.04, 0.06, 0.08, 0.1}. For each model network, we compare the approximated values of BD returned from the proposed algorithms with that of baseline methods. Since we can obtain explicit expression of BDs for these model networks, the mean relative error of approximation results can be directly computed. The experimental results are reported in Figure 8.

As demonstrated in Figure 8, in most cases, our proposed algorithms still outperform the baseline methods. Moreover, we have once again observed phenomena akin to those in real-world networks: within the small-world networks Star-1e6, Star-1e7, $\mathcal{T}_{12}$, and $\mathcal{T}_{13}$, BackPush executes more swiftly than FastWalk and FastTree. However, in the "large-world" networks $\mathcal{F}_7$ and $\mathcal{F}_8$, FastWalk

Table 3. Mean relative error (E) and per-query time (T, ms) for three algorithms under four pivot heuristics.

Dataset	BackPush								FastWalk								FastTree							
	Degree		PageRank		k-Core		Closeness		Degree		PageRank		k-Core		Closeness		Degree		PageRank		k-Core		Closeness	
	E	T	E	T	E	T	E	T	E	T	E	T	E	T	E	T	E	T	E	T	E	T	E	T
Facebook	0.008	1027	0.009	2182	0.01	2301	0.011	3319	0.015	1310	0.015	1690	0.013	1610	0.012	2180	0.018	125	0.02	412	0.021	371	0.017	721
Road–RA	0.07	3e7	0.072	3e7	0.06	2e8	0.081	1e8	0.005	3e6	0.005	2e7	0.01	3e6	0.012	2e7	0.004	2e5	0.006	1e6	0.006	2e6	0.01	2e6
Youtube	0.006	1073	0.008	2110	0.01	1920	0.013	2973	0.006	5e4	0.007	5e4	0.01	5e4	0.013	2e5	0.03	2e4	0.03	5e4	0.032	2e5	0.033	4e5
Orkut	0.006	3e4	0.007	5e4	0.01	5e4	0.02	6e4	0.029	2e5	0.02	4e5	0.03	3e5	0.03	7e5	0.006	6e4	0.007	6e4	0.01	9e4	0.012	1e5
soc–LJ	0.016	6e4	0.018	6e4	0.02	6e4	0.02	1e5	0.021	1e5	0.02	2e5	0.02	2e5	0.023	3e5	0.016	8e4	0.02	8e4	0.02	1e5	0.023	1e5
hetero–LJ	0.003	3e5	0.005	3e5	0.008	4e5	0.01	4e5	0.007	6e5	0.008	6e5	0.01	6e5	0.01	1e6	0.005	4e5	0.007	4e5	0.01	5e5	0.01	1e6

and FastTree outpace BackPush. This discrepancy arises because in such "large-world" networks, multiple hops are required for nodes to reach one another, resulting in the *pushback* operations within BackPush proceeding at a lower pace. On the other hand, FastWalk and FastTree, which are based on random walks, can explore the network more rapidly, thereby reaching the pivot node faster, which contributes to their superior performance.

7.4 Parameter Sensitivity Analysis

7.4.1 Choice of the Pivot Node. Selecting a good pivot node can significantly reduce the computational cost. Ideally one would choose the vertex that minimises the total absorbing-walk hitting time, i.e. the *absorbing random-walk centrality* (ARWC) defined in [48]. However, evaluating ARWC exactly entails inverting the Laplacian (or solving n Laplacian systems), which costs $O(n^3)$ time and is infeasible on our tested large graphs.

To assess the practical gap between this ideal and inexpensive heuristics, we test four pivot selection heuristics—*degree*, *PageRank*, *k-core index*, and *closeness centrality*—on all data sets and across our algorithms. We configure each algorithm with its strictest parameter settings defined in Section 7.2. For each configuration we fix 1000 random queries.

As shown in Table 3, the *highest-degree* pivot consistently yields the best or near-best accuracy while being the fastest by a large margin. On *Facebook*, for instance, it achieves the lowest error ($E = 0.008$ in BackPush) and cuts the runtime by over 50% relative to PageRank and by more than 3× compared to Closeness. Similar trends are observed on *Road–RA* and *Youtube*. This observation aligns with prior studies identifying degree as a strong indicator of node accessibility [8, 16, 45]. In random-walk methods such as FastWalk, this high reachability enables more valid random walks, thereby directly reducing the estimation error. In BackPush, the pivot's dense connectivity allows it to rapidly absorb the residuals from neighbors, preventing them from diffusing to values below the threshold $r_{\max}$ and thereby minimizing approximation bias. To further validate this design choice, we extended our evaluation to include a Random (select a pivot node uniformly at random) and Betweenness Centrality (BC) in Table 4. The results demonstrate that Degree significantly outperforms the random strategy. More importantly, it achieves accuracy comparable to the computationally intensive BC. Therefore, we confirm it as the default strategy.

7.4.2 On the Truncation Length. For FastWalk, choosing ℓ is critical: a small value speeds up sampling but may truncate walks prematurely thus reducing approximate accuracy, while an overly large value only inflates running time. Figure 9 (a) reports the *empirical* mean length of absorbing walks on six real graphs, using the highest-degree node as the pivot. We denote Facebook, Road-PA, Youtube, Orkut, soc-LiveJournal, and hetero-LiveJournal by *FB*, *RA*, *YT*, *OK*, *SLJ*, and *HLJ*, respectively. Across all datasets, the average walk is shorter than 10^4 steps, indicating that

Table 4. Performance of BackPush with different pivot selection strategies. Degree outperforms Random in both accuracy (E) and query time (T, ms), and achieves accuracy comparable to Betweenness.

Dataset	Random		Degree		Betweenness	
	E	T	E	T	E	T
ca-GrQc	0.045	2300	0.008	1100	0.008	1105
PowerGrid	0.072	3450	0.010	1300	0.010	1312
Facebook	0.058	2150	0.008	1027	0.007	1035

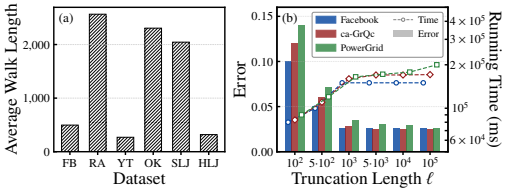


Fig. 9. The effect of the truncation length ℓ. In Figure (b), the data points for each dataset, from left to right, correspond to truncation lengths $\ell \in (10^2, 5\times10^2, 10^3, 5\times10^3, 10^4, 10^5)$.

most walks reach the pivot very quickly. Based on these broad observations, we determined a conservative default truncation of $\ell = 10^5$.

To verify the cross-dataset generalization of our choice, we conducted a validation experiment visualized in Figure 9 (b). We selected three representative networks—one observed benchmark (Facebook) and two *unseen* datasets (ca-GrQc and PowerGrid)—where we could compute the exact BD. The results confirm the efficacy of our strategy across different datasets. On Facebook and ca-GrQc, increasing ℓ beyond 10^3 incurs negligible additional cost. This occurs because random walks on these graphs terminate naturally before reaching the truncation limit. Conversely, on PowerGrid, although convergence is slower due to the high network diameter, the error drops to an acceptable level once ℓ exceeds 10^3. Our default $\ell = 10^5$ further ensures high precision. This validates that our default choice is safe and generalizes well.

8 Related Work

In this section, we review existing BD algorithms and similar optimization techniques applied to other computing problems.

Computation of BD. Given its wide range of applications including networked systems [18, 76–78], graph matching [17], and graph machine learning [10, 32, 66], among others [39, 65, 76], the computation of BD has garnered significant attention. Yi et al. [76–78] proposed an algorithm with linear runtime scaling in terms of the number of edges, leveraging random projections and symmetric diagonally dominant (SDD) solvers to approximate BD. More recently, Liu et al. [42] introduced a novel formulation of BD based on an infinite series of random walk probabilities and developed several approximation algorithms for pairwise BD computation. However, existing methods struggle to efficiently handle massive-scale networks. Even the random walk-based SOTA algorithm SWF from [42] also struggles with large-scale networks due to the excessive length and number of random walks required for simulation. In contrast, our approach, rooted in the newly developed pivot node-based formulation, achieves a substantial efficiency gain without sacrificing the accuracy.

It is worth emphasizing the methods used in this paper, like random walks and spanning trees, have been applied in previous work for computing other graph-based quantities. However, as discussed, extending these algorithms to estimate BD is non-trivial. Below, we provide more details on their usage in prior works.

Random Walks and Push Operation. Random walk-based and push-based methods have been widely used to approximate various graph metrics, such as personalized PageRank and resistance distance (ER) [3, 14, 25, 27, 36, 43, 46, 54, 67, 68, 68, 71, 73, 75]. In particular, [36, 37] introduced a novel formulation of ER using the Laplacian $\boldsymbol{L}$ and developed random walk-based algorithms

tailored to its estimation. However, in this study, our focus is on BD, which is another distance metric. The complex formulation of BD involving $\widehat{\boldsymbol{L}}_v^{-1}$ makes direct adaptation of these methods non-trivial, necessitating significant algorithmic innovation.

Furthermore, we should mention that our algorithms are fundamentally different from those in [42]. We introduce a novel formulation for BD using absorbing random walks centered around a pivot node, unlike the infinite series expansion in [42]. This new formulation enables flexible pivot selection and underpins our distinct algorithms. More specifically, while [42] performs a forward push from the query nodes, our BackPush algorithm propagates backward from the pivot node. More importantly, the stopping criteria are different: [42] uses a fixed number of steps K, while BackPush dynamically terminates based on a threshold $r_{\max}$, allowing for adaptive convergence depending on the local structure. Our FastWalk simulates absorbing random walks, using an ℓ-truncated strategy to avoid excessively long walks and improve efficiency. In contrast, SWF uses fixed-length walks, which can be inefficient in large-diameter networks.

Spanning Trees. Spanning tree-based methods, rooted in the classic matrix-tree theorem [30], have found applications in constructing cut sparsifiers [19, 20], solving optimization problems such as the traveling salesman problem [5, 22], and computing the random walk centrality [41], among others [37, 40]. In this work, we uncover a novel connection between BD and spanning trees, which inspires the design of FastTree that leverages this structure for efficient computation.

Pivot Node-based Methods. Furthermore, pivot node-based methods, widely used in shortest path [2, 23, 47, 53, 63, 79] and nearest neighbor (NN) computations [1, 49, 50], precompute distances to a set of pivots to accelerate queries via the triangle inequality. Liao et. al. [36, 37] recently also developed several pivot node-based algorithms to compute effective resistance. These methods influence the structural optimization aspects of our algorithms. Our algorithms, motivated by these foundational approaches, integrate and extend these techniques to address the unique challenges of BD, achieving efficient and accurate computation tailored to this metric.

9 Conclusion

In this paper, we studied the problem of approximately computing BD, particularly on large networks. We proposed novel formulations for BD, which form the foundation for developing several new and efficient algorithms for the pairwise BD query problem. We established several connections among BD, random walk, random spanning trees, and deterministic push operation, and bring new and deep insights on efficient BD computations. We conduct extensive experiments on 6 real-world datasets and 6 model networks to evaluate our algorithms. The results show that our proposed algorithms consistently outperform the SOTA algorithms.

One potential direction for future work is to enhance the proposed algorithms by incorporating multiple pivot nodes to accelerate the absorbing process. However, the corresponding formulation will become more complex, and thus we will need to develop further novel algorithmic techniques to address this problem. Moreover, since our sampling-based approaches are naturally parallelizable, we intend to explore multi-core implementations. This would allow us to leverage modern hardware for handling massive-scale graphs.

ACKNOWLEDGEMENTS

The work was supported by the National Natural Science Foundation of China (Nos. 62372112 and 61872093).

References

[1] Tenindra Abeywickrama and Muhammad Aamir Cheema. 2017. Efficient Landmark-Based Candidate Generation for kNN Queries on Road Networks. In *Database Systems for Advanced Applications*. Springer International Publishing,

425–440.
[2] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast Exact Shortest-path Distance Queries on Large Networks by Pruned Landmark Labeling. In *Proceedings of the SIGMOD International Conference on Management of Data*. 349–360.
[3] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S. Mirrokni, and Shang-Hua Teng. 2007. Local Computation of PageRank Contributions. In *Proceedings of the 5th International Conference on Algorithms and Models for the Web-Graph*. 150–165.
[4] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local Graph Partitioning using PageRank Vectors. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*. 475–486.
[5] Arash Asadpour, Michel X. Goemans, Aleksander Mźdry, Shayan Oveis Gharan, and Amin Saberi. 2017. An $O(\log n/\log\log n)$-Approximation Algorithm for the Asymmetric Traveling Salesman Problem. *Operation Research* 65, 4 (2017), 1043–1061.
[6] Bassam Bamieh, Mihailo R Jovanovic, Partha Mitra, and Stacy Patterson. 2012. Coherence in Large-Scale Networks: Dimension-dependent Limitations of Local Feedback. *IEEE Trans. Automat. Control* 57, 9 (2012), 2235–2249.
[7] Albert-László Barabási and Réka Albert. 1999. Emergence of Scaling in Random Networks. *Science* 286, 5439 (1999), 509–512.
[8] K. Batool and M. A. Niazi. 2014. Towards a Methodology for Validation of Centrality Measures in complex networks. *PLOS ONE* 9, 4 (2014), e90283.
[9] Mitchell Black, Lucy Lin, Weng-Keen Wong, and Amir Nayyeri. 2024. Biharmonic Distance of Graphs and its Higher-order Variants: Theoretical Properties with Applications to Centrality and Clustering. In *Proceedings of the 41st International Conference on Machine Learning*. 4077–4102.
[10] Mitchell Black, Zhengchao Wan, Amir Nayyeri, and Yusu Wang. 2023. Understanding Oversquashing in GNNs through the Lens of Effective Resistance. In *Proceedings of the 40th International Conference on Machine Learning*. 2528–2547.
[11] Béla Bollobás. 1998. *Modern Graph Theory*. Vol. 184. Springer Science & Business Media.
[12] Michael B Cohen, Rasmus Kyng, Gary L Miller, Jakub W Pachocki, Richard Peng, Anup B Rao, and Shen Chen Xu. 2014. Solving SDD Linear Systems in Nearly $m\log^{1/2} n$ Time. In *Proceedings of the 46th Annual Symposium on Theory of Computing*. 343–352.
[13] S. Condamin, O. Bénichou, V. Tejedor, R. Voituriez, and J. Klafter. 2007. First-passage Times in Complex Scale-invariant Media. *Nature* 450, 7166 (2007), 77–80.
[14] Guanyu Cui, Hanzhi Wang, and Zhewei Wei. 2025. Mixing Time Matters: Accelerating Effective Resistance Estimation via Bidirectional Method. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 177–188.
[15] P.G. Doyle and J.L. Snell. 1984. *Random Walks and Electric Networks*. Mathematical Association of America.
[16] T. S. Evans and B. Chen. 2022. Linking the Network Centrality Measures Closeness and Degree. *Communications Physics* 5, 1 (2022), 172.
[17] Zhou Fan, Cheng Mao, Yihong Wu, and Jiaming Xu. 2020. Spectral Graph Matching and Regularized Quadratic Relaxations: Algorithm and Theory. In *Proceedings of the 37th International Conference on Machine Learning*. 2985–2995.
[18] Katherine Fitch and Naomi Leonard. 2014. Joint Centrality Distinguishes Optimal Leaders in Noisy Networks. *IEEE Transactions on Control of Network Systems* 3 (2014), 366–378.
[19] Alan Frieze, Navin Goyal, Luis Rademacher, and Santosh Vempala. 2014. Expanders via Random Spanning Trees. *SIAM J. Comput.* 43, 2 (2014), 497–513.
[20] Wai Shing Fung and Nicholas J. A. Harvey. 2010. Graph Sparsification by Edge-Connectivity and Random Spanning Trees. *CoRR* abs/1005.0265 (2010). arXiv:1005.0265
[21] Yuan Gao, Rasmus Kyng, and Daniel A. Spielman. 2023. Robust and Practical Solution of Laplacian Equations by Approximate Elimination. *CoRR* abs/2303.00709 (2023). arXiv:2303.00709
[22] Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. 2011. A Randomized Rounding Approach to the Traveling Salesman Problem. In *IEEE 52nd Annual Symposium on Foundations of Computer Science*. 550–559.
[23] Andrew V. Goldberg and Chris Harrelson. 2005. Computing the Shortest Path: A Search Meets Graph Theory. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. 156–165.
[24] Qintian Guo, Dandan Lin, Sibo Wang, Raymond Chi-Wing Wong, and Wenqing Lin. 2024. Efficient Algorithms for Group Hitting Probability Queries on Large Graphs. *IEEE Transactions on Knowledge and Data Engineering* 36, 7 (2024), 2995–3008.
[25] Taher H. Haveliwala. 2002. Topic-sensitive PageRank. In *Proceedings of the 11th International Conference on World Wide Web*. 517–526.
[26] Wassily Hoeffding. 1963. Probability Inequalities for Sums of Bounded Random Variables. *J. Amer. Statist. Assoc.* 58, 301 (1963), 13–30.

[27] Glen Jeh and Jennifer Widom. 2003. Scaling Personalized Web Search. In *Proceedings of the 12th International Conference on World Wide Web*. 271–279.

[28] Yujia Jin, Qi Bao, and Zhongzhi Zhang. 2019. Forest Distance Closeness Centrality in Disconnected Graphs. In *Proceedings of the IEEE International Conference on Data Mining*. 339–348.

[29] S. Jung, S. Kim, and B. Kahng. 2002. Geometric Fractal Growth Model for Scale-free Networks. *Physical Review E* 65 (2002), 056101.

[30] G. Kirchhoff. 1847. Ueber die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird. *Annalen der Physik* 148, 12 (1847), 497–508.

[31] D.J. Klein and M. Randić. 1993. Resistance Distance. *Journal of Mathematical Chemistry* 12, 1 (1993), 81–95.

[32] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. 2021. Rethinking Graph Transformers With Spectral Attention. *Proceedings of the 35th International Conference on Neural Information Processing Systems* 34, 21618–21629.

[33] Lawler and F. Gregory. 1980. A Self-avoiding Random Walk. *Duke Mathematical Journal* 47, 3 (1980), 655–693.

[34] Lawler and F. Gregory. 2012. *Intersections of Random Walks*. Springer New York.

[35] Jure Leskovec and Rok Sosič. 2016. SNAP: A General-purpose Network Analysis and Graph-mining Library. *ACM Transactions on Intelligent Systems and Technology* 8, 1 (2016), 1.

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

[37] Meihao Liao, Junjie Zhou, Rong-Hua Li, Qiangqiang Dai, Hongyang Chen, and Guoren Wang. 2024. Efficient and Provable Effective Resistance Computation on Large Graphs: An Index-based Approach. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1 – 27.

[38] Yuan Lin, Alafate Julaiti, and Zhongzhi Zhang. 2012. Mean First-passage Time for Random Walks in General Graphs with a Deep Trap. *The Journal of Chemical Physics* 137, 12 (2012), 124104.

[39] Y. Lipman, R.M. Rustamov, and T.A. Funkhouser. 2010. Biharmonic Distance. *ACM Transactions on Graphics* 29 (2010), 1–11.

[40] Changan Liu, Haoxin Sun, Ahad N. Zehmakan, and Zhongzhi Zhang. 2026. Efficient Edge Rewiring Strategies for Enhancing PageRank Fairness. *Theoretical Computer Science* 1067 (2026), 115765.

[41] Changan Liu, Zixuan Xie, Ahad N. Zehmakan, and Zhongzhi Zhang. 2026. Efficient Algorithms for Computing Random Walk Centrality. *IEEE Transactions on Knowledge and Data Engineering* 38, 1 (2026), 235–247.

[42] Changan Liu, Ahad N. Zehmakan, and Zhongzhi Zhang. 2024. Fast Query of Biharmonic Distance in Networks. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1887–1897.

[43] Changan Liu, Xiaotian Zhou, Ahad N. Zehmakan, and Zhongzhi Zhang. 2024. A Fast Algorithm for Moderating Critical Nodes via Edge Removal. *IEEE Transactions on Knowledge and Data Engineering* 36, 4 (2024), 1385–1398.

[44] L. Lovász. 1996. Random Walks on Graphs: A Survey. In *Combinatorics, Paul Erdős is Eighty*. Vol. 2. János Bolyai Mathematical Society, 353–398.

[45] Linyuan Lü, Duanbing Chen, Xiao-Long Ren, Qian-Ming Zhang, Yi-Cheng Zhang, and Tao Zhou. 2016. Vital Nodes Identification in Complex Networks. *Physics Reports* 650 (2016), 1–63.

[46] Takanori Maehara, Takuya Akiba, Yoichi Iwata, and Ken-ichi Kawarabayashi. 2014. Computing Personalized PageRank Quickly by Exploiting Graph Structures. *Proceedings of the VLDB Endowment* 7, 12 (2014), 1023–1034.

[47] Shlomi Maliah, Rami Puzis, and Guy Shani. 2017. Shortest Path Tree Sampling for Landmark Selection in Large Networks. *Journal of Complex Networks* 5, 5 (2017), 795–815.

[48] Charalampos Mavroforakis, Michael Mathioudakis, and Aristides Gionis. 2015. Absorbing Random-Walk Centrality: Theory and Algorithms. In *Proceedings of the 2015 IEEE International Conference on Data Mining*. 901–906.

[49] Luisa Micó, Jose Oncina, and Rafael C. Carrasco. 1996. A Fast Branch & Bound Nearest Neighbour Classifier in Metric Spaces. *Pattern Recognition Letters* 17, 7 (1996), 731–739.

[50] María Luisa Micó, José Oncina, and Enrique Vidal. 1994. A New Version of the Nearest-neighbour Approximating and Eliminating Search Algorithm (AESA) with Linear Preprocessing Time and Memory Requirements. *Pattern Recognition Letters* 15, 1 (1994), 9–17.

[51] Ulla Miekkala. 1993. Graph Properties for Splitting with Grounded Laplacian Matrices. *BIT Numerical Mathematics* 33, 3 (1993), 485–495.

[52] Mark Newman. 2018. *Networks*. Oxford University Press.

[53] Dian Ouyang, Lu Qin, Lijun Chang, Xuemin Lin, Ying Zhang, and Qing Zhu. 2018. When Hierarchy Meets 2-Hop-Labeling: Efficient Shortest Distance Queries on Road Networks. In *Proceedings of the International Conference on Management of Data*. 709–724.

[54] Pan Peng, Daniel Lopatta, Yuichi Yoshida, and Gramoz Goranci. 2021. Local Algorithms for Estimating Effective Resistance. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1329–1338.

[55] Hernán D Rozenfeld, Shlomo Havlin, and Daniel ben Avraham. 2007. Fractal and Transfractal Recursive Scale-free Nets. *New Journal of Physics* 9, 6 (2007), 175.
[56] Purnamrita Sarkar, Andrew W Moore, and Amit Prakash. 2008. Fast Incremental Proximity Search in Large Graphs. In *Proceedings of the 25th International Conference on Machine Learning*. 896–903.
[57] Louis W. Shapiro. 1987. An Electrical Lemma. *Mathematics Magazine* 60, 1 (1987), 36–38.
[58] Yibin Sheng and Zhongzhi Zhang. 2019. Low-mean Hitting Time for Random Walks on Heterogeneous Networks. *IEEE Transactions on Information Theory* 65, 11 (2019), 6898–6910.
[59] Yutaka Shimada, Yoshito Hirata, Tohru Ikeguchi, and Kazuyuki Aihara. 2016. Graph Distance for Complex Networks. *Scientific Reports* 6, 1 (2016), 34944.
[60] Chaoming Song, Shlomo Havlin, and Hernán A. Makse. 2006. Origins of Fractality in the Growth of Complex Networks. *Nature Physics* 2, 4 (2006), 275–281.
[61] Daniel A. Spielman and Shang-Hua Teng. 2014. Nearly Linear Time Algorithms for Preconditioning and Solving Symmetric, Diagonally Dominant Linear Systems. *SIAM J. Matrix Anal. Appl.* 35, 3 (2014), 835–885.
[62] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. 2022. Understanding Over-squashing and Bottlenecks on Graphs via Curvature. In *International Conference on Learning Representations*.
[63] Konstantin Tretyakov, Abel Armas-Cervantes, Luciano García-Bañuelos, Jaak Vilo, and Marlon Dumas. 2011. Fast Fully Dynamic Landmark-based Estimation of Shortest Path Distances in Very Large Graphs. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*. 1785–1794.
[64] Anton Tsitsulin, Marina Munkhoeva, and Bryan Perozzi. 2020. Just SLaQ When You Approximate: Accurate Spectral Distances for Web-Scale Graphs. In *Proceedings of The Web Conference*. 2697–2703.
[65] Melvyn Tyloo, Tommaso Coletta, and Philippe Jacquod. 2017. Robustness of Synchrony in Complex Networks and Generalized Kirchhoff Indices. *Physical Review Letters* 120 8 (2017), 084101.
[66] Saurabh Verma and Zhi-Li Zhang. 2017. Hunt for the Unique,Stable, Sparse and Fast Feature Learning on Graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 87–97.
[67] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. HubPPR: Effective Indexing for Approximate Personalized PageRank. *Proceedings of the VLDB Endowment* 10, 3 (2016).
[68] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: Simple and Effective Approximate Single-source Personalized Pagerank. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 505–514.
[69] Duncan J Watts and Steven H Strogatz. 1998. Collective Dynamics of 'Small-world' Networks. *Nature* 393, 6684 (1998), 440–442.
[70] Y. Wei, R. Li, and W. Yang. 2021. Biharmonic Distance of Graphs. *arXiv preprint arXiv:2110.02656v1* (2021).
[71] Zhewei Wei, Ji-Rong Wen, and Mingji Yang. 2024. Approximating Single-source Personalized PageRank with Absolute Error Guarantees. In *27th International Conference on Database Theory*, Vol. 290. 9:1–9:19.
[72] David Bruce Wilson. 1996. Generating Random Spanning Trees More Quickly than the Cover Time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. 296–303.
[73] Mingji Yang, Hanzhi Wang, Zhewei Wei, Sibo Wang, and Ji-Rong Wen. 2024. Efficient Algorithms for Personalized PageRank Computation: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 36, 9 (2024), 4582–4602.
[74] Renchi Yang. 2022. Efficient and Effective Similarity Search over Bipartite Graphs. In *Proceedings of the ACM Web Conference*. 308–318.
[75] Renchi Yang and Jing Tang. 2023. Efficient Estimation of Pairwise Effective Resistance. *Proceedings of the ACM on Management of Data* 1, 1 (2023).
[76] Yuhao Yi, Liren Shan, Huan Li, and Zhongzhi Zhang. 2018. Biharmonic Distance Related Centrality for Edges in Weighted Networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. 3620–3626.
[77] Yuhao Yi, Bingjia Yang, Zhongzhi Zhang, and Stacy Patterson. 2018. Biharmonic Distance and Performance of Second-Order Consensus Networks with Stochastic Disturbances. In *Annual American Control Conference*. 4943–4950.
[78] Yuhao Yi, Bingjia Yang, Zuobai Zhang, Zhongzhi Zhang, and Stacy Patterson. 2022. Biharmonic Distance-based Performance Metric for Second-order Noisy Consensus Networks. *IEEE Transactions on Information Theory* 68 (2022), 1220–1236.
[79] Hongzhi Yin, Bin Cui, Jing Li, Junjie Yao, and Chen Chen. 2012. Challenging the Long Tail Recommendation. *Proceedings of the VLDB Endowment* 5, 9 (2012), 896–907.
[80] Hongyang Zhang, Huacheng Yu, and Ashish Goel. 2019. Pruning Based Distance Sketches with Provable Guarantees on Random Graphs. In *The World Wide Web Conference*. 2301–2311.
[81] Zhongzhi Zhang, Yuan Lin, and Youjun Ma. 2011. Effect of Trap Position on the Efficiency of Trapping in Treelike Scale-free Networks. *Journal of Physics A: Mathematical and Theoretical* 44, 7 (2011), 075102.

[82] Zhongzhi Zhang, Yihang Yang, and Yuan Lin. 2012. Random Walks in Modular Scale-free Networks with Multiple Traps. *Physical Review E* 85 (2012), 011106.
[83] Xiaotian Zhou and Zhongzhi Zhang. 2023. Opinion Maximization in Social Networks via Leader Selection. In *Proceedings of the ACM Web Conference*. 133–142.

Received October 2025; revised January 2026; accepted February 2026