BD-Index: Scalable Biharmonic Distance Queries on Large Graphs via Divide-and-Conquer Indexing

Yueyang Pan
Beijing Institute of Technology
Beijing, China
yypan@bit.edu.cn

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

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

ABSTRACT

Biharmonic distance (BD) is a powerful graph distance metric with many applications, including identifying critical links in road networks and mitigating over-squashing problem in GNN. However, computing BD is extremely difficult, especially on large graphs. In this paper, we focus on the problem of *single-pair* BD query. Existing methods mainly rely on random walk-based approaches, which work well on some graphs but become inefficient when the random walk cannot mix rapidly. To overcome this issue, we first show that the biharmonic distance between two nodes s, t, denoted by $b(s, t)$, can be interpreted as the distance between two random walk distributions starting from s and t. To estimate these distributions, the required random walk length is large when the underlying graph can be easily cut into smaller pieces. Inspired by this observation, we present novel formulas of BD to represent $b(s, t)$ by independent random walks within two node sets $\mathcal{V}_s, \mathcal{V}_t$ separated by a small *cut set* $\mathcal{V}_{cut}$, where $\mathcal{V}_s \cup \mathcal{V}_t \cup \mathcal{V}_{cut} = \mathcal{V}$ is the set of graph nodes. Building upon this idea, we propose BD-Index, a novel index structure which follows a divide-and-conquer strategy. The graph is first cut into pieces so that each part can be processed easily. Then, all the required random walk probabilities can be deterministically computed in a bottom-top manner. When a query comes, only a small part of the index needs to be accessed. We prove that BD-Index requires $O(n \cdot h)$ space, can be built in $O(n \cdot h \cdot (h + d_{max}))$ time, and answers each query in $O(n \cdot h)$ time, where h is the height of a hierarchy partition tree and d_{max} is the maximum degree, which are both usually much smaller than n. A striking feature of BD-Index is that it is a *theoretically exact* method, in contrast to existing random-walk-based approaches that only provide approximate estimates of BD. Extensive experiments on 10 large datasets demonstrate that BD-Index outperforms state-of-the-art (SOTA) exact methods by at least 2 orders of magnitude in speed. It is even an order of magnitude faster than SOTA approximate methods. For example, on a large road network Road-CA with 1,971,281 nodes and 2,766,607 edges, BD-Index consumes 2 seconds while its exact (approximate) competitors takes more than 2,600 (80) seconds, with a reasonably 9.3 GB index size. Furthermore, we also conduct two case studies to confirm the effectiveness of BD in real data-mining tasks.

1 INTRODUCTION

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected and connected graph with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be its adjacency matrix and $\mathbf{D} = \text{diag}(d_1, \ldots, d_n)$ be the degree matrix (a diagonal matrix) where $d_i = \sum_j \mathbf{A}_{ij}$. The graph Laplacian is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$. The *biharmonic distance* (BD) between nodes s and t is defined as

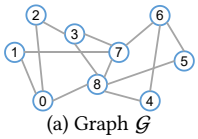
$$b(s, t) = (\mathbf{e}_s - \mathbf{e}_t)^\top \mathbf{L}^{2\dagger} (\mathbf{e}_s - \mathbf{e}_t),$$

where $\mathbf{L}^\dagger$ denotes the Moore–Penrose pseudoinverse of $\mathbf{L}$, $\mathbf{L}^{2\dagger}$ represents the square of $\mathbf{L}^\dagger$, and $\mathbf{e}_i$ is the i-th standard basis vector [52]. The *single-pair* BD query problem is: given two nodes $s, t \in V$, compute $b(s, t)$ efficiently without computing $\mathbf{L}^\dagger$ or all-pairs distances.

The BD metric has been applied in many areas of data management and network analysis [10, 11, 52, 53, 80–82]. Two representative applications are (1) identifying critical link identification on road networks [80], and (2) over-squashing mitigation in GNN [11]. Both applications require *many fast single-pair queries* rather than global computation. For example, in GNN rewiring, one needs to repeatedly evaluate BD between many node pairs to add or remove edges adaptively; in road networks, detecting critical links requires repeated distance evaluation between selected intersections or regions. Thus, the main challenge is to answer numerous single-pair BD queries accurately and efficiently on large graphs.

BD can be computed by applying exact Laplacian solvers such as Cholesky decomposition solver [36] and other Laplacian solvers [15, 21, 30, 44] with high precision. The most advanced Laplacian solver applies approximate Gaussian elimination to build preconditioners for the PCG (preconditioned conjugate gradient) routine [30], which requires $\widetilde{O}(m)$ time. However, as the hidden factor is large, it still requires high time and memory cost. To further enhance efficiency, existing approximate methods for computing BD can be divided into two categories. The first group, *Laplacian solver-based methods* [52], use random projections by iteratively solving a small number of Laplacian solvers. The second group, *random walk-based methods* [53], approximate BD via sampling random walks between nodes. These methods are lightweight and perform well on many graphs, but their efficiency strongly depends on the walk length l, which must be sufficiently large to achieve small approximation error. In practice, l can be very large; for instance, on the Amazon dataset, $l = 10^6$ is needed to reach relative error of 10^{-4}, and the average query time exceeds 10^2 seconds. Therefore, while random walk approaches are effective in some cases, they can be extremely slow or inaccurate on graphs where long walks dominate.

To overcome these limitations, we present several new insights into the structure of BD. We show that BD can be interpreted as the distance between two random walk distributions starting from nodes s and t. We further discover that the regions corresponding to long random walks are often easy to cut, meaning that the random walk can be restricted within two large subsets that are divided by a small cut set. This leads to a divide-and-conquer formulation: we can cut the graph into smaller pieces, compute local random



(a) Graph $\mathcal{G}$

$$\begin{bmatrix}
0.35 & 0.09 & 0.11 & -0.03 & -0.15 & -0.15 & -0.15 & -0.06 & -0.03 \\
0.09 & 0.51 & -0.04 & -0.07 & -0.16 & -0.16 & -0.13 & 0.03 & -0.07 \\
0.11 & -0.04 & 0.55 & 0.09 & -0.18 & -0.18 & -0.19 & -0.08 & -0.07 \\
-0.03 & -0.07 & 0.09 & 0.33 & -0.11 & -0.11 & -0.11 & 0.01 & -0.01 \\
-0.15 & -0.16 & -0.18 & -0.11 & 0.51 & 0.01 & 0.12 & -0.06 & 0.02 \\
-0.15 & -0.16 & -0.18 & -0.11 & 0.01 & 0.51 & 0.12 & -0.06 & 0.02 \\
-0.15 & -0.13 & -0.19 & -0.11 & 0.12 & 0.12 & 0.37 & 0.00 & -0.03 \\
-0.06 & 0.03 & -0.08 & 0.01 & -0.06 & -0.06 & 0.00 & 0.23 & -0.01 \\
-0.03 & -0.07 & -0.07 & -0.01 & 0.02 & 0.02 & -0.03 & -0.01 & 0.18
\end{bmatrix}$$

(b) Moore–Penrose pseudoinverse of $\mathbf{L}$

Figure 1: Graph $\mathcal{G}$ and its Moore–Penrose pseudoinverse $\mathbf{L}^\dagger$. For example, $b(2,5) = \left\|L^\dagger \mathbf{e}_2 - L^\dagger \mathbf{e}_5\right\|_2^2 = 1.28$.

walk probabilities within each piece, and then combine them hierarchically. These observations inspire our new index structure, named BD-Index, which deterministically stores intermediate random walk probabilities hierarchically.

BD-Index follows a bottom-up construction. We first cut the graph into subgraphs along small vertex cuts and recursively compute local random walk quantities inside each piece. Each piece contributes partial statistics that are merged upward until the entire graph is covered. During query time, only the relevant parts of the index along the cut paths of s and t need to be accessed. Theoretical analysis shows that the index size is $O(n \cdot h)$, it can be constructed in $O(n \cdot h \cdot (h + d_{max}))$ time, each query takes $O(n \cdot h)$ time, where h is the height of the hierarchy tree $\mathcal{H}$, typically much smaller than n (as confirmed in our experiments). It is important to notice that although BD is represented in terms of random walks, the proposed BD-Index is an exact method, as all random walk probabilities are computed deterministically without sampling, and the only negligible error comes from floating-point precision.

We conduct extensive experiments on 10 real-world datasets. On the Amazon graph, BD-Index achieves an order-of-magnitude speedup over approximate methods with a moderate 21 GB index. On road networks, BD-Index performs even better—for example, on the NewYork graph, it requires only 0.35 GB of index space while being two orders of magnitude faster over the fastest approximate method. Meanwhile, BD-Index is theoretically exact. In addition, two case studies demonstrate the practical value of BD in real applications—one in identifying critical links in urban transportation networks, and another in improving GNN performance through BD-guided rewiring.

2 PRELIMINARIES

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected and connected graph with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. The adjacency matrix is $\mathbf{A} \in \mathbb{R}^{n \times n}$, where $A_{ij} = 1$ if $(i, j) \in \mathcal{E}$ and 0 otherwise. The degree matrix is $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n)$ with $d_i = \sum_j A_{ij}$. The combinatorial Laplacian is $\mathbf{L} = \mathbf{D} - \mathbf{A}$, and $\mathbf{L}^\dagger$ denotes its Moore–Penrose pseudoinverse. The vector $\mathbf{e}_i$ represents the i-th standard basis vector. The biharmonic distance between nodes $s, t \in V$, denoted by $b(s, t)$, is defined by:

$$b(s,t) = (\mathbf{e}_s - \mathbf{e}_t)^\top \mathbf{L}^{2\dagger} (\mathbf{e}_s - \mathbf{e}_t),$$

where $\mathbf{L}^{2\dagger}$ denotes the square of $\mathbf{L}^\dagger$. Computing $b(s, t)$ exactly requires access to $\mathbf{L}^\dagger$, which is very expensive for large graphs.

Random walk. A simple random walk on G starts from a node and at each step moves to one of its neighbors chosen with equal probability. The transition matrix is $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}, P_{ij} = \frac{A_{ij}}{d_i}$. Then $(\mathbf{P}^k)_{ij}$ is the probability that a walk beginning at node i is at node j after k steps. The mixing time is the number of steps required for the walk's distribution to become close to the stationary distribution. For a walk starting from node s, the expected number of times it visits node t is $\tau_{s,t} = \sum_{k=0}^{\infty} (\mathbf{P}^k)_{st}$. We also define the *degree-normalized expected visit count* $\tilde{\tau}_{s,t} = \frac{\tau_{s,t}}{d_t}$, which divides by the degree of t. A *v-absorbed random walk* is a random walk that starts from a node and stops once it first reaches node v. For such a walk from s, we define the expected number of visits to node t as $\tau_{s,t}^{(v)}$, and its degree-normalized counterpart as $\tilde{\tau}_{s,t}^{(v)} = \frac{\tau_{s,t}^{(v)}}{d_t}$. Previous work [50] has shown that if $\mathbf{L}_v$ denotes the principal submatrix of the graph Laplacian $\mathbf{L}$ obtained by removing the row and column corresponding to node v, then $\tilde{\tau}_{s,t}^{(v)} = \left(\mathbf{L}_v^{-1}\right)_{s,t}$.

Single-pair query problem. Given nodes $s, t \in \mathcal{V}$, a *single-pair query* returns $b(s, t)$. In many applications such as identifying critical links in road networks [80] or over-squashing mitigation in GNN [11], many such queries must be answered for different pairs. The goal is to build an index that enables efficient queries of BD.

2.1 Existing methods and their defects

Exact methods for computing BD formulate $b(s, t)$ as solving a linear system $\mathbf{L}^2\mathbf{x} = \mathbf{e}_s - \mathbf{e}_t$, $b(s, t) = (\mathbf{e}_s - \mathbf{e}_t)^\top \mathbf{x}$. A straightforward approach is to invoke a Cholesky factorization [36], or other high-precision Laplacian solvers [15, 21, 30, 44]. The state-of-the-art solvers construct a sequence of preconditioners via approximate Gaussian elimination and use these inside a preconditioned conjugate gradient (PCG) routine [30]. This yields a nearly-linear $\widetilde{O}(m)$ running time in theory. However, the hidden constants are large: constructing and storing multi-level preconditioners is memory-intensive and each solve still requires many PCG iterations. Consequently, these solvers remain costly on large graphs and are impractical for workloads with many single-pair BD queries, each requiring a separate linear solve.

Existing approximate methods can be grouped into two main categories: (i) *Laplacian-solver–based methods.* Yi et al. [80] use *random projection*, which projects vectors into a smaller subspace to estimate $\mathbf{L}^{2\dagger}(\mathbf{e}_s - \mathbf{e}_t)$. Let the projection dimension be r, the total complexity becomes $O(mr + nr^2)$ time and $O(nr)$ space. However, to obtain small error, r must still be large, so the method is not efficient on very large graphs. (ii) *Random walk–based methods* [53] reformulated $b(s, t)$ in terms of random walk expectations with the transition matrix $\mathbf{P}$. They proposed several algorithms—Push, STW, and the improved SWF —that compute partial walk probabilities up to a maximum length l. All of them approximate $b(s, t)$ by combining the contributions from walks of length at most l. The total time complexity of SWF is $O\left(\frac{2}{\epsilon^2 (\min d)^4} l^5\right)$, where ϵ is the target relative error and $\min d$ is the minimum node degree. When the graph has long paths or is very sparse, a large l is required, which makes the query slow and less accurate.

To overcome these problems, we first deeply investigate the random walk interpretation of BD, showing that random walk-based approaches perform poorly when the underlying graph is easily separable. Thus, we present several new interpretations of BD in Section 3 to represent BD in terms of random walks limit

in two independent sets separated by a small cut set. Then, we propose a novel index-based approach BD-Index in Section 4 to efficiently answer single-pair queries over the whole graph.

3 NEW THEORETICAL RESULTS

In this section, we present several new theoretical results on BD. We first show that previous *random walk–based methods* implicitly provide a random walk interpretation of BD: the BD between s and t can be viewed as the distance between two random walk distributions, where more similar distributions yield smaller BD values. Next, we derive a new formula, showing that BD can also be explained by random walks from s and t to any node v. When v separates the graph, we can restrict the walks to their respective subgraphs. Finally, we extend this to the case where the cut set $\mathcal{V}_{cut}$ contains multiple nodes: the walks can still be restricted to the two subgraphs, while the extra information can be stored with very little space. Our overall idea is illustrated in Figure 2.

3.1 Interpreting BD in terms of random walk

Lemma 3.1 (Global Random Walk Representation of BD). *Let $\tilde{\boldsymbol{\tau}}_s^{(\infty)}$ denote the degree-normalized distribution of the expected visit counts for an infinite random walk starting from node s. Then, the biharmonic distance satisfies*

$$b(s,t) \;=\; \|\tilde{\boldsymbol{\tau}}_s^{(\infty)} - \tilde{\boldsymbol{\tau}}_t^{(\infty)}\|_2^2 \;-\; \frac{1}{n}\left(\mathbf{1}^\top(\tilde{\boldsymbol{\tau}}_s^{(\infty)} - \tilde{\boldsymbol{\tau}}_t^{(\infty)})\right)^2. \tag{1}$$

Proof. For any two nodes s and t, we define $\tau_{s,t}^{(\ell)}$ as the expected number of visits to t made by a length-ℓ random walk starting from s. Formally, let $\text{path}_\ell = (v_0, v_1, \ldots, v_\ell)$ denote a walk of length ℓ and write $\Pr(\text{path}_\ell) = \prod_{i=0}^{\ell-1} \mathbf{P}_{v_i,v_{i+1}}$, with the convention $\Pr(\text{path}_0) = 1$ when $\ell = 0$. We use $[\cdot]$ for the Iverson bracket, i.e., $[\text{cond}] = 1$ if the condition holds and 0 otherwise. Then

$$\begin{aligned}
\tau_{s,t}^{(\ell)} &= \mathbb{E}\Big[\sum_{j=0}^{\ell}[v_j = t] \;\Big|\; v_0 = s\Big] = \sum_{\text{path}_\ell}[v_0 = s]\Pr(\text{path}_\ell)\sum_{j=0}^{\ell}[v_j = t] \\
&= \sum_{\text{path}_\ell}[v_0 = s]\Big(\prod_{i=0}^{\ell-1}\mathbf{P}_{v_i,v_{i+1}}\Big)\sum_{j=0}^{\ell}[v_j = t] \\
&= \sum_{j=0}^{\ell}\sum_{\text{path}_\ell}[v_0 = s][v_j = t]\prod_{i=0}^{\ell-1}\mathbf{P}_{v_i,v_{i+1}} \\
&= \sum_{j=0}^{\ell}\sum_{\text{path}_\ell}[v_0 = s][v_j = t]\Big(\prod_{i=0}^{j-1}\mathbf{P}_{v_i,v_{i+1}}\Big)\Big(\prod_{i=j}^{\ell-1}\mathbf{P}_{v_i,v_{i+1}}\Big) \\
&= \sum_{j=0}^{\ell}\sum_{\text{path}_j}[v_0 = s][v_j = t]\Pr(\text{path}_j)\sum_{\text{path}_{\ell-j}}[v_0' = t]\Pr(\text{path}_{\ell-j}) \\
&= \sum_{j=0}^{\ell}\mathbf{P}_{s,t}^j\sum_{\text{path}_{\ell-j}}[v_0' = t]\Pr(\text{path}_{\ell-j}) \\
&= \sum_{j=0}^{\ell}\mathbf{P}_{s,t}^j\sum_{k=1}^{n}\mathbf{P}_{t,k}^{\ell-j} = \sum_{j=0}^{\ell}\mathbf{P}_{s,t}^j,
\end{aligned}$$

where v_0' denotes the starting node of the suffix walk $\text{path}_{\ell-j}$ and we used that each row of $\mathbf{P}^{\ell-j}$ sums to 1. Hence, whenever the series converges, the expected number of visits to t along an infinite-length walk from s is $\tau_{s,t}^{(\infty)} := \sum_{j=0}^{\infty}\mathbf{P}_{s,t}^j$. It is convenient to collect these expectations into a degree-normalized visit-count vector $\tilde{\boldsymbol{\tau}}_s := \sum_{i=0}^{\infty}\mathbf{e}_s\mathbf{P}^i\mathbf{D}^{-1}$, where $\mathbf{e}_s$ is the one-hot row vector for node s; the t-th entry of $\tilde{\boldsymbol{\tau}}_s$ equals $\tau_{s,t}^{(\infty)}/d_t$. Previous work [53] has proved that

$$b(s,t) = \left\|\sum_{i=0}^{\infty}(\mathbf{e}_s - \mathbf{e}_t)^\top\mathbf{P}^i\mathbf{D}^{-1}\right\|_2^2 + \frac{1}{n}\left\|\sum_{i=0}^{\infty}(\mathbf{e}_s - \mathbf{e}_t)^\top\mathbf{P}^i\mathbf{D}^{-1}\mathbf{1}\right\|_2^2.$$

By substituting $\tilde{\boldsymbol{\tau}}_s = \sum_{i=0}^{\infty}\mathbf{e}_s\mathbf{P}^i\mathbf{D}^{-1}$ into the above expression, we obtain the desired result, which completes the proof. □

Lemma 3.1 expresses BD as the difference between two infinite random walk distributions. However, in some parts of the graph, random walks mix slowly. According to the Cheeger inequality [18], slow mixing implies a small spectral gap, which reflects the existence of a small cut. Such regions are structurally easy to separate, as random walks tend to remain within them for a long time. Therefore, we aim to cut the graph at these places. To this end, we introduce v-absorbed random walk, which terminates when it hits vertex v. If v is a cut vertex, the random walk can then be decomposed into two independent walks within the separated subgraphs.

Lemma 3.2 (v-absorbed Random Walk Representation of BD). *For any node $v \in V$, let $\tilde{\boldsymbol{\tau}}_s^{(v)}$ denote the degree-normalized distribution of the expected visit counts for an v-absorbed random walk starting from node s. Then, the biharmonic distance satisfies*

$$\begin{aligned}
b(s,t) &= \left\|\mathbf{L}_v^{-1}(\mathbf{e}_s - \mathbf{e}_t)\right\|_2^2 - \frac{1}{n}\left(\mathbf{1}^\top\mathbf{L}_v^{-1}(\mathbf{e}_s - \mathbf{e}_t)\right)^2 \\
&= \left\|\tilde{\boldsymbol{\tau}}_s^{(v)} - \tilde{\boldsymbol{\tau}}_t^{(v)}\right\|_2^2 - \frac{1}{n}\left(\mathbf{1}^\top(\tilde{\boldsymbol{\tau}}_s^{(v)} - \tilde{\boldsymbol{\tau}}_t^{(v)})\right)^2.
\end{aligned} \tag{2}$$

Proof. For any vector $\mathbf{x} \in \mathbb{R}^n$, write $\mathbf{x}^{(-v)} \in \mathbb{R}^{n-1}$ for the vector obtained by deleting its v-th entry, and let $\mathbf{1}$ denote the all-ones vector in $\mathbb{R}^{n-1}$. Set $\mathbf{x} = \mathbf{e}_s - \mathbf{e}_t$, so that $\mathbf{1}^\top\mathbf{x} = 0$. A direct calculation using the fact that $\mathbf{L}^\dagger$ is the Moore–Penrose pseudoinverse of $\mathbf{L}$ and that $\mathbf{L}_v$ is nonsingular shows that, for any zero-sum $\mathbf{x}$,

$$(\mathbf{L}^\dagger\mathbf{x})^{(-v)} = \mathbf{L}_v^{-1}\mathbf{x}^{(-v)} - \tfrac{1}{n}\,\mathbf{1}\mathbf{1}^\top\mathbf{L}_v^{-1}\mathbf{x}^{(-v)}.$$

Using also that for any $u \neq v$, $\mathbf{e}_u^\top\mathbf{L}^\dagger\mathbf{x} = \mathbf{e}_u^\top\mathbf{L}_v^{-1}\mathbf{x}^{(-v)}$, we obtain

$$\begin{aligned}
b(s,t) &= \mathbf{x}^\top(\mathbf{L}^\dagger)^2\mathbf{x} \\
&= \mathbf{x}^{(-v)\top}\mathbf{L}_v^{-1}(\mathbf{L}^\dagger\mathbf{x})^{(-v)} \\
&= \mathbf{x}^{(-v)\top}\mathbf{L}_v^{-1}\Big(\mathbf{L}_v^{-1}\mathbf{x}^{(-v)} - \tfrac{1}{n}\mathbf{1}\mathbf{1}^\top\mathbf{L}_v^{-1}\mathbf{x}^{(-v)}\Big) \\
&= \left\|\mathbf{L}_v^{-1}\mathbf{x}^{(-v)}\right\|_2^2 - \tfrac{1}{n}\left(\mathbf{1}^\top\mathbf{L}_v^{-1}\mathbf{x}^{(-v)}\right)^2 \\
&= \left\|\mathbf{L}_v^{-1}(\mathbf{e}_s - \mathbf{e}_t)\right\|_2^2 - \tfrac{1}{n}\left(\mathbf{1}^\top\mathbf{L}_v^{-1}(\mathbf{e}_s - \mathbf{e}_t)\right)^2.
\end{aligned}$$

For the random walk representation, previous work [50] shows that the degree-normalized expected visit counts of the v-absorbed random walk satisfy $\tilde{\tau}_{u,w}^{(v)} = (\mathbf{L}_v^{-1})_{u,w}$ for all $u, w \neq v$. In particular, $\mathbf{L}_v^{-1}\mathbf{e}_s = \tilde{\boldsymbol{\tau}}_s^{(v)}$ and $\mathbf{L}_v^{-1}\mathbf{e}_t = \tilde{\boldsymbol{\tau}}_t^{(v)}$, hence $\mathbf{L}_v^{-1}(\mathbf{e}_s - \mathbf{e}_t) = \tilde{\boldsymbol{\tau}}_s^{(v)} - \tilde{\boldsymbol{\tau}}_t^{(v)}$. Substituting this into the expression above gives

$$b(s,t) = \left\|\tilde{\boldsymbol{\tau}}_s^{(v)} - \tilde{\boldsymbol{\tau}}_t^{(v)}\right\|_2^2 - \tfrac{1}{n}\left(\mathbf{1}^\top(\tilde{\boldsymbol{\tau}}_s^{(v)} - \tilde{\boldsymbol{\tau}}_t^{(v)})\right)^2,$$

as claimed. □

Based on Lemma 3.2, BD is given by the difference between the distributions of two v-absorbed random walks starting from s and t. If v serves as a cut vertex separating s and t, the two walks are restricted to their own subgraphs on each side of the cut.

Lemma 3.3 (Cut-vertex Random Walk Representation of BD). *Suppose v is a cut vertex that divides the graph into two disconnected components $\mathcal{V}_s$ and $\mathcal{V}_t$, with $s \in \mathcal{V}_s$ and $t \in \mathcal{V}_t$. If we order the vertices as $(\mathcal{V}_s, \mathcal{V}_t)$, the matrix $\mathbf{L}_v^{-1}$ becomes block diagonal and can be written as $\mathbf{L}_v^{-1} = \begin{bmatrix} \mathbf{L}_{\mathcal{V}_s} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}_{\mathcal{V}_t} \end{bmatrix}$, where $\mathbf{L}_{\mathcal{V}_s}$ is the part of $\mathbf{L}_v^{-1}$ that contains the rows and columns of the vertices in $\mathcal{V}_s \cup \{v\}$, and $\mathbf{L}_{\mathcal{V}_t}$ is defined in the same way for $\mathcal{V}_t \cup \{v\}$. Let $\tilde{\boldsymbol{\tau}}_s^{(v,\mathcal{V}_s)}$ and $\tilde{\boldsymbol{\tau}}_t^{(v,\mathcal{V}_t)}$ denote the degree-normalized distributions of random walks starting from s and t, respectively, each restricted to its corresponding subgraph $(\mathcal{V}_s \cup \{v\})$ and $(\mathcal{V}_t \cup \{v\})$ with absorption at v. Then*

$$b(s,t) = \left\|\mathbf{L}_{\mathcal{V}_s}^{-1}\mathbf{e}_s - \mathbf{L}_{\mathcal{V}_t}^{-1}\mathbf{e}_t\right\|_2^2 - \frac{1}{n}\left(\mathbf{1}^\top(\mathbf{L}_{\mathcal{V}_s}^{-1}\mathbf{e}_s - \mathbf{L}_{\mathcal{V}_t}^{-1}\mathbf{e}_t)\right)^2$$
$$= \left\|\tilde{\boldsymbol{\tau}}_s^{(v,\mathcal{V}_s)} - \tilde{\boldsymbol{\tau}}_t^{(v,\mathcal{V}_t)}\right\|_2^2 - \frac{1}{n}\left(\mathbf{1}^\top(\tilde{\boldsymbol{\tau}}_s^{(v,\mathcal{V}_s)} - \tilde{\boldsymbol{\tau}}_t^{(v,\mathcal{V}_t)})\right)^2. \quad (3)$$

Proof. By Lemma 3.2, for any choice of absorbing node v we have $b(s,t) = \left\|\mathbf{L}_v^{-1}(\mathbf{e}_s - \mathbf{e}_t)\right\|_2^2 - \frac{1}{n}\left(\mathbf{1}^\top\mathbf{L}_v^{-1}(\mathbf{e}_s - \mathbf{e}_t)\right)^2$. Since v is a cut vertex, removing v disconnects the remaining vertices into the two components $\mathcal{V}_s$ and $\mathcal{V}_t$. If we order the vertices as $(\mathcal{V}_s, \mathcal{V}_t)$, the grounded Laplacian $\mathbf{L}_v$ is block diagonal, so its inverse has the form $\mathbf{L}_v^{-1} = \begin{bmatrix} \mathbf{L}_{\mathcal{V}_s}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}_{\mathcal{V}_t}^{-1} \end{bmatrix}$, where $\mathbf{L}_{\mathcal{V}_s}^{-1}$ and $\mathbf{L}_{\mathcal{V}_t}^{-1}$ are the inverses of the corresponding diagonal blocks of $\mathbf{L}_v$. Identifying $\mathbb{R}^{n-1}$ with $\mathbb{R}^{|\mathcal{V}_s|} \oplus \mathbb{R}^{|\mathcal{V}_t|}$, and viewing $\mathbf{L}_{\mathcal{V}_s}^{-1}\mathbf{e}_s$ and $\mathbf{L}_{\mathcal{V}_t}^{-1}\mathbf{e}_t$ as vectors extended by zeros outside $\mathcal{V}_s$ and $\mathcal{V}_t$, respectively, we obtain

$$\mathbf{L}_v^{-1}(\mathbf{e}_s - \mathbf{e}_t) = \begin{bmatrix} \mathbf{L}_{\mathcal{V}_s}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}_{\mathcal{V}_t}^{-1} \end{bmatrix}\begin{bmatrix} \mathbf{e}_s \\ -\mathbf{e}_t \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{\mathcal{V}_s}^{-1}\mathbf{e}_s \\ -\mathbf{L}_{\mathcal{V}_t}^{-1}\mathbf{e}_t \end{bmatrix} = \mathbf{L}_{\mathcal{V}_s}^{-1}\mathbf{e}_s - \mathbf{L}_{\mathcal{V}_t}^{-1}\mathbf{e}_t.$$

Substituting this into the expression for $b(s,t)$ yields

$$b(s,t) = \left\|\mathbf{L}_{\mathcal{V}_s}^{-1}\mathbf{e}_s - \mathbf{L}_{\mathcal{V}_t}^{-1}\mathbf{e}_t\right\|_2^2 - \tfrac{1}{n}\left(\mathbf{1}^\top(\mathbf{L}_{\mathcal{V}_s}^{-1}\mathbf{e}_s - \mathbf{L}_{\mathcal{V}_t}^{-1}\mathbf{e}_t)\right)^2,$$

which is the first line of (3).

For the random walk representation, previous work [50] implies $\tilde{\boldsymbol{\tau}}_u^{(v)} = \mathbf{L}_v^{-1}\mathbf{e}_u$ for every u. When $u \in \mathcal{V}_s$, any v-absorbed random walk starting from u stays in $\mathcal{V}_s \cup \{v\}$ until it is absorbed at v, so the coordinates of $\tilde{\boldsymbol{\tau}}_u^{(v)}$ on $\mathcal{V}_t$ are zero, and its restriction to $\mathcal{V}_s \cup \{v\}$ coincides with $\tilde{\boldsymbol{\tau}}_u^{(v,\mathcal{V}_s)}$. Thus $\tilde{\boldsymbol{\tau}}_s^{(v,\mathcal{V}_s)}$ is exactly $\mathbf{L}_{\mathcal{V}_s}^{-1}\mathbf{e}_s$ (extended by zeros on $\mathcal{V}_t$), and similarly $\tilde{\boldsymbol{\tau}}_t^{(v,\mathcal{V}_t)}$ is $\mathbf{L}_{\mathcal{V}_t}^{-1}\mathbf{e}_t$ (extended by zeros on $\mathcal{V}_s$). Replacing $\mathbf{L}_{\mathcal{V}_s}^{-1}\mathbf{e}_s$ and $\mathbf{L}_{\mathcal{V}_t}^{-1}\mathbf{e}_t$ above by $\tilde{\boldsymbol{\tau}}_s^{(v,\mathcal{V}_s)}$ and $\tilde{\boldsymbol{\tau}}_t^{(v,\mathcal{V}_t)}$ gives exactly the identity in (3). □

Lemma 3.3 states that if a vertex v is a cut vertex that divides the graph into two disconnected components $\mathcal{V}_s$ and $\mathcal{V}_t$, with $s \in \mathcal{V}_s$ and $t \in \mathcal{V}_t$, then $b(s,t)$ can be formulated in terms of random walks restricted within $\mathcal{V}_s$ and $\mathcal{V}_t$. This division of the graph naturally prevents the slow mixing that tends to appear around small cuts.

3.2 Cutting random walks by a cut set

On real-world graphs, a single cut vertex that separates the graph may not exist, or may be difficult to identify. We therefore extend the previous idea to a small cut set $\mathcal{V}_{cut} = \{c_1, c_2, \ldots, c_k\}$ that cut graph into disconnected components $\mathcal{V}_s$ and $\mathcal{V}_t$. We show that $b(s,t)$ can also be represented by distribution of random walks in $\mathcal{V}_s$ and $\mathcal{V}_t$ (i.e., $\tilde{\boldsymbol{\tau}}_s^{(v,\mathcal{V}_s)}$ and $\tilde{\boldsymbol{\tau}}_t^{(v,\mathcal{V}_t)}$) additional with some matrices $\mathbf{M}$, which can also be represented by random walks.

In the following lemma, given a small cut set $\mathcal{V}_{cut}$ with a fixed order, we remove node $c_i \in \mathcal{V}_{cut}$ one by one so that, after removing all of them, the graph becomes two disconnected parts $\mathcal{V}_s$ and $\mathcal{V}_t$ containing s and t.

Lemma 3.4 (Cut-set Random Walk Representation of BD). *Let $R^{(1)} := V \setminus \{v\}$, and after removing node c_j, define the remaining vertex set $R^{(j)} := R^{(j-1)} \setminus \{c_j\}$. Reorder the Laplacian $\mathbf{L}_{R^{(j-1)}}$ so that c_j is placed last: $\mathbf{L}_{R^{(j-1)}} = \begin{bmatrix} \mathbf{L}_{R^{(j)}} & -\boldsymbol{a}_{c_j} \\ -\boldsymbol{a}_{c_j}^\top & d_{c_j} \end{bmatrix}$, where $\boldsymbol{a}_{c_j}$ is the vector of edge weights connecting c_j with the other vertices in $R^{(j)}$, and d_{c_j} is the degree of c_j. Let $S_{c_j} = d_{c_j} - \boldsymbol{a}_{c_j}^\top\mathbf{L}_{R^{(j)}}^{-1}\boldsymbol{a}_{c_j}$, and define the contribution matrix $\mathbf{M}_{c_j}^{(j)} = \begin{bmatrix} \mathbf{L}_{R^{(j)}}^{-1}\boldsymbol{a}_{c_j} \\ 1 \end{bmatrix} S_{c_j}^{-1}\left[\boldsymbol{a}_{c_j}^\top\mathbf{L}_{R^{(j)}}^{-1},\ 1\right]$. $\widetilde{\mathbf{M}}_{c_j}$ denotes $\mathbf{M}_{c_j}^{(j)}$ padded with 0 to match the full node set $V \times V$. Then,*

$$b(s,t) = \left\|\tilde{\boldsymbol{\tau}}_s^{(\mathcal{V}_{cut},\mathcal{V}_s)} - \tilde{\boldsymbol{\tau}}_t^{(\mathcal{V}_{cut},\mathcal{V}_t)} + \sum_{i=1}^{j}\widetilde{\mathbf{M}}_{c_i}(\mathbf{e}_s - \mathbf{e}_t)\right\|_2^2$$
$$- \frac{1}{n}\left(\mathbf{1}^\top\left(\tilde{\boldsymbol{\tau}}_s^{(\mathcal{V}_{cut},\mathcal{V}_s)} - \tilde{\boldsymbol{\tau}}_t^{(\mathcal{V}_{cut},\mathcal{V}_t)} + \sum_{i=1}^{j}\widetilde{\mathbf{M}}_{c_i}(\mathbf{e}_s - \mathbf{e}_t)\right)\right)^2. \quad (4)$$

Proof. By construction $\mathbf{L}_v = \mathbf{L}_{R^{(1)}}$. For each $i \in \{1, \ldots, j\}$, reorder the vertices in $R^{(i-1)}$ so that c_i is placed last, and write

$$\mathbf{L}_{R^{(i-1)}} = \begin{bmatrix} \mathbf{L}_{R^{(i)}} & -\boldsymbol{a}_{c_i} \\ -\boldsymbol{a}_{c_i}^\top & d_{c_i} \end{bmatrix}, \qquad S_{c_i} = d_{c_i} - \boldsymbol{a}_{c_i}^\top\mathbf{L}_{R^{(i)}}^{-1}\boldsymbol{a}_{c_i}.$$

The standard block matrix inversion formula gives

$$\mathbf{L}_{R^{(i-1)}}^{-1} = \begin{bmatrix} \mathbf{L}_{R^{(i)}} & -\boldsymbol{a}_{c_i} \\ -\boldsymbol{a}_{c_i}^\top & d_{c_i} \end{bmatrix}^{-1}$$
$$= \begin{bmatrix} \mathbf{L}_{R^{(i)}}^{-1} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} \mathbf{L}_{R^{(i)}}^{-1}\boldsymbol{a}_{c_i}S_{c_i}^{-1}\boldsymbol{a}_{c_i}^\top\mathbf{L}_{R^{(i)}}^{-1} & \mathbf{L}_{R^{(i)}}^{-1}\boldsymbol{a}_{c_i}S_{c_i}^{-1} \\ S_{c_i}^{-1}\boldsymbol{a}_{c_i}^\top\mathbf{L}_{R^{(i)}}^{-1} & S_{c_i}^{-1} \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{L}_{R^{(i)}}^{-1} & 0 \\ 0 & 0 \end{bmatrix} + \mathbf{M}_{c_i}^{(i)}.$$

Padding by zeros to the full vertex set $V \times V$, this can be written as $\mathbf{L}_{R^{(i-1)}}^{-1} = \widetilde{\mathbf{L}}_{R^{(i)}}^{-1} + \widetilde{\mathbf{M}}_{c_i}$. Iterating from $i = 1$ to $i = j$ yields

$$\mathbf{L}_v^{-1} = \mathbf{L}_{R^{(1)}}^{-1} = \widetilde{\mathbf{L}}_{R^{(j)}}^{-1} + \sum_{i=1}^{j}\widetilde{\mathbf{M}}_{c_i}.$$

After removing the cut set $\mathcal{V}_{cut}$, the remaining vertices split into two components $\mathcal{V}_s$ and $\mathcal{V}_t$ containing s and t, respectively. By Lemma 3.3, the matrix $\mathbf{L}_{R^{(j)}}^{-1}$ is block diagonal: $\mathbf{L}_{R^{(j)}}^{-1} = \begin{bmatrix} \mathbf{L}_{\mathcal{V}_s}^{-1} & 0 \\ 0 & \mathbf{L}_{\mathcal{V}_t}^{-1} \end{bmatrix}$,

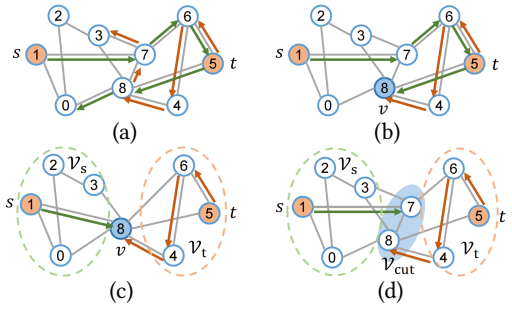


Figure 2: Illustration of the proposed formulas of BD. **(a)** BD **can be interpreted by random walks from s and t on the whole graph; (b)** BD **can be interpreted by random walks from s and t until hitting v; (c) If v is a cut vertex,** BD **can be interpreted by random walks independently on $\mathcal{V}_s$ and $\mathcal{V}_t$; (d)** BD **can be interpreted by random walks independently on $\mathcal{V}_s$ and $\mathcal{V}_t$, separated by a small cut set $\mathcal{V}_{cut} = \{v_7, v_8\}$.**

so that, viewing the blocks as zero-padded to V,

$$\begin{aligned}\mathbf{L}_v^{-1}(\mathbf{e}_s - \mathbf{e}_t) &= \widetilde{\mathbf{L}}_{R^{(j)}}^{-1}(\mathbf{e}_s - \mathbf{e}_t) + \sum_{i=1}^{j} \widetilde{\mathbf{M}}_{c_i}(\mathbf{e}_s - \mathbf{e}_t) \\ &= \begin{bmatrix} \widetilde{\mathbf{L}}_{\mathcal{V}_s}^{-1} & 0 \\ 0 & \widetilde{\mathbf{L}}_{\mathcal{V}_t}^{-1} \end{bmatrix} (\mathbf{e}_s - \mathbf{e}_t) + \sum_{i=1}^{j} \widetilde{\mathbf{M}}_{c_i}(\mathbf{e}_s - \mathbf{e}_t) \\ &= \widetilde{\mathbf{L}}_{\mathcal{V}_s}^{-1}\mathbf{e}_s - \widetilde{\mathbf{L}}_{\mathcal{V}_t}^{-1}\mathbf{e}_t + \sum_{i=1}^{j} \widetilde{\mathbf{M}}_{c_i}(\mathbf{e}_s - \mathbf{e}_t).\end{aligned}$$

By Lemma 3.2, $\widetilde{\mathbf{L}}_{\mathcal{V}_s}^{-1}\mathbf{e}_s$ and $\widetilde{\mathbf{L}}_{\mathcal{V}_t}^{-1}\mathbf{e}_t$ are exactly the degree-normalized expected visit distributions $\tilde{\boldsymbol{\tau}}_s^{(\mathcal{V}_{cut},\mathcal{V}_s)}$ and $\tilde{\boldsymbol{\tau}}_t^{(\mathcal{V}_{cut},\mathcal{V}_t)}$, respectively. Therefore

$$\mathbf{L}_v^{-1}(\mathbf{e}_s - \mathbf{e}_t) = \tilde{\boldsymbol{\tau}}_s^{(\mathcal{V}_{cut},\mathcal{V}_s)} - \tilde{\boldsymbol{\tau}}_t^{(\mathcal{V}_{cut},\mathcal{V}_t)} + \sum_{i=1}^{j} \widetilde{\mathbf{M}}_{c_i}(\mathbf{e}_s - \mathbf{e}_t).$$

Finally, substituting this expression into the v-absorbed representation of BD in Lemma 3.2, gives exactly the identity in (4). □

In Lemma 3.4, $\tilde{\boldsymbol{\tau}}_s^{(\mathcal{V}_{cut},\mathcal{V}_s)}$ and $\tilde{\boldsymbol{\tau}}_s^{(\mathcal{V}_{cut},\mathcal{V}_s)}$ represents two independent v-absorbed random walks, one inside each subgraph $\mathcal{V}_s, \mathcal{V}_t$. Each contribution matrix $\widetilde{\mathbf{M}}_{c_j}$ deterministically adds the paths that go through node $c_j \in \mathcal{V}_{cut}$ while avoiding all the previous random paths. Together, these terms recover the full v-absorbed random walk behavior on the original graph. This insight motivates our approach: we iteratively cut the graph into smaller pieces and deterministically store the corresponding random walk distributions.

4 THE PROPOSED APPROACH: BD-INDEX

4.1 High-level idea

Divide: cut graph into pieces. Building upon Lemma 3.4, the BD can be expressed as two parts: (i) the difference between the degree-normalized random walk distributions inside the two subgraphs $\mathcal{V}_s$ and $\mathcal{V}_t$, and (ii) a term that contains $k = |\mathcal{V}_{\text{cut}}|$ contribution matrices $\sum_{j=1}^{k} \widetilde{\mathbf{M}}_{c_j}$, which together deterministically describe the distribution of random walks that pass through the cut set $\mathcal{V}_{\text{cut}}$.

By cutting the graph with a small cut set $\mathcal{V}_{\text{cut}}$, we can transform random walks on the full graph into random walks restricted in the two subgraphs $\mathcal{V}_s$ and $\mathcal{V}_t$, together with a small number of matrices $\widetilde{\mathbf{M}}_{c_j}$. When the cut set size k is small, these matrices can be computed efficiently.

However, a single cut divides the graph into only two parts. It is easy to observe that the same property holds recursively for each subgraph: the random walk distribution within any subgraph can again be decomposed by using another small cut set that splits the subgraph into smaller pieces. Repeating this process eventually leads to subgraphs that are so small that their degree-normalized random walk distributions can be computed exactly. In the case where a subgraph contains only one node v, the random walk distribution equals 1, and its degree-normalized form is $1/d_v$. Therefore, by recursively partitioning the graph with a series of small cut sets until each block has only one vertex, we can deterministically compute the random walk distributions on the entire graph with very low computational cost.

Then, a natural question arises: how to store all the contribution matrices $\widetilde{\mathbf{M}}$ efficiently? At first sight, storing all contribution matrices $\{\widetilde{\mathbf{M}}_{c_j}\}$ seems to require $O(n^3)$ space. However, each matrix $\widetilde{\mathbf{M}}_{c_j}$, which represents all random walks that pass through the cut set, can be decomposed into three parts: (i) walks that start from nodes in $\mathcal{V}_s \cup \mathcal{V}_t$ and first reach the cut set $\mathcal{V}_{\text{cut}}$; (ii) walks that start from the cut set and reach nodes in $\mathcal{V}_s \cup \mathcal{V}_t$; and (iii) walks that start and end within the cut set itself. The first two parts can be represented by one vector of length at most n, while the third part corresponds to a constant value shared by all node pairs $s, t \in \mathcal{V}_s \cup \mathcal{V}_t$. Hence, the information in each matrix $\widetilde{\mathbf{M}}_{c_j}$ can be stored using only one vector and one constant, which greatly reduces the total space needed to represent all random walk distributions.

Lemma 4.1 (Compact Representation of the Contribution Matrices). *Consider one cut node $x \in \mathcal{V}_{\text{cut}}$ in the process described in Lemma 3.4. Let R be the current set of nodes that still contains x, and let L_R denote the submatrix of the Laplacian on the nodes in $R \setminus \{x\}$. Let $\boldsymbol{a}_x$ be the vector that contains the edges between x and its neighboring nodes in $R \setminus \{x\}$, and let d_x be the degree of node x.*

Then, the contribution matrix of x can be written as the product of two vectors:

$$\mathbf{M}_x = \begin{bmatrix} \boldsymbol{m}_x \\ 1 \end{bmatrix} f_x^{-1} \begin{bmatrix} \boldsymbol{m}_x^\top & 1 \end{bmatrix}, \tag{5}$$

where

$$\boldsymbol{m}_x = L_R^{-1}\boldsymbol{a}_x, \qquad f_x = d_x - \boldsymbol{a}_x^\top L_R^{-1}\boldsymbol{a}_x.$$

Proof. Consider the Laplacian on the current node set R, reordered so that x is placed last. By definition of L_R, $\boldsymbol{a}_x$, and d_x, this Laplacian has the block form $\widehat{\mathbf{L}}_R = \begin{bmatrix} L_R & -\boldsymbol{a}_x \\ -\boldsymbol{a}_x^\top & d_x \end{bmatrix}$. Since L_R is nonsingular, we can invert $\widehat{\mathbf{L}}_R$ using the standard block-matrix inversion formula. With $A = L_R$, $B = -\boldsymbol{a}_x$, $C = -\boldsymbol{a}_x^\top$, and $D = d_x$, the Schur complement of A is

$$S = D - CA^{-1}B = d_x - \boldsymbol{a}_x^\top L_R^{-1}\boldsymbol{a}_x = f_x.$$

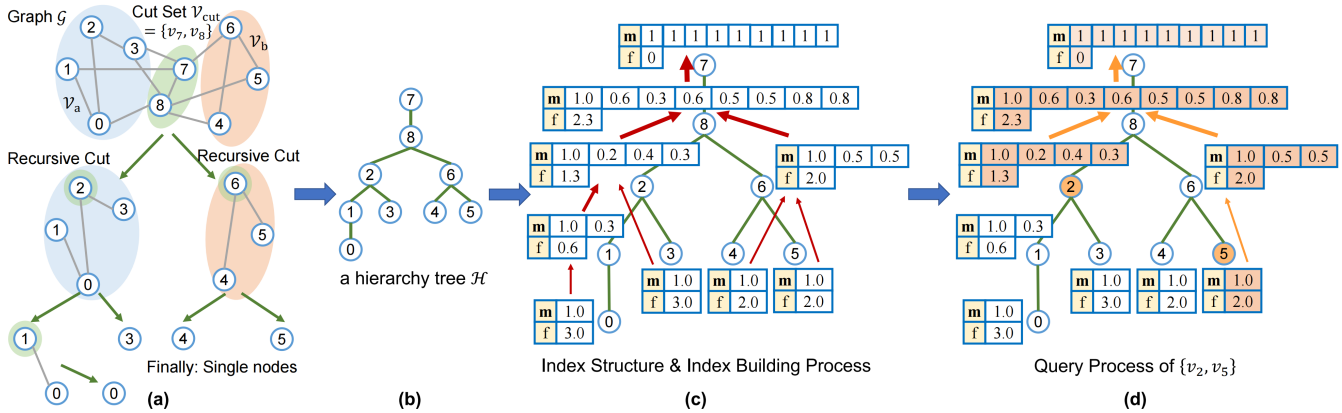


Figure 3: Illustration of BD-Index's index structure, index building process and query process

The inverse is then

$$\widehat{\mathbf{L}}_R^{-1} = \begin{bmatrix} L_R^{-1} + L_R^{-1}BS^{-1}CL_R^{-1} & -L_R^{-1}BS^{-1} \\ -S^{-1}CL_R^{-1} & S^{-1} \end{bmatrix} = \begin{bmatrix} L_R^{-1} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} L_R^{-1}\boldsymbol{a}_x \\ 1 \end{bmatrix} f_x^{-1} \begin{bmatrix} \boldsymbol{a}_x^\top L_R^{-1}, & 1 \end{bmatrix}.$$

By definition (cf. Lemma 3.4), the second term is exactly the contribution matrix $\mathbf{M}_x$ associated with eliminating node x. Writing $\boldsymbol{m}_x = L_R^{-1}\boldsymbol{a}_x$ gives

$$\mathbf{M}_x = \begin{bmatrix} \boldsymbol{m}_x \\ 1 \end{bmatrix} f_x^{-1} \begin{bmatrix} \boldsymbol{m}_x^\top & 1 \end{bmatrix},$$

which is (5). □

By Lemma 4.1, all information of every step in Lemma 3.4 can be kept by storing only the pair $(\boldsymbol{m}_x, f_x)$, where $\boldsymbol{m}_x$ is kept on the nodes of R that are connected to x. This form avoids building the whole matrix $\mathbf{M}_x$ and keeps exactly the same contribution.

We first build a recursive cut hierarchy, denoted by $\mathcal{H}$, by cutting the graph until all leaves become single vertices. Any effective routine can be applied; we adopt a recursive minimum vertex cut [40], which repeatedly finds a small near-minimum cut at each level, as the default, and also evaluate a degree-based heuristic cut [62] that splits the graph around nodes with small degrees.

Conquer: a bottom-up method. After dividing the graph, we obtain a hierarchical structure $\mathcal{H}$. For each node v, its contribution matrix $\widetilde{\mathbf{M}}_v$ depends only on its descendants, denoted $\mathrm{Desc}(v)$, that is, all nodes below it in the hierarchy. Hence, the contribution matrices can be computed in a bottom-up manner, where each node v is updated from the information of its descendants.

Lemma 4.2 (Bottom-Up Aggregation). *Let $(\mathbf{m}_u, f_u)$ be the stored pair for each $u \in \mathrm{Desc}(v)$. Then the pair for node v can be obtained by combining the pairs of all its descendants:*

$$\mathbf{m}_v = \mathbf{e}_v + \sum_{u \in \mathrm{Desc}(v)} \left(\frac{\sum_{x \in N(v) \cap \mathrm{Desc}(u)} \mathbf{m}_u[x]}{f_u} \right) \mathbf{m}_u,$$
$$f_v = d_v - \sum_{x \in N(v) \cap \mathrm{Desc}(v)} \mathbf{m}_v[x]. \tag{6}$$

Proof. For every descendant $u \in \mathrm{Desc}(v)$, the stored pair $(\mathbf{m}_u, f_u)$ summarizes the total contribution of all cut nodes in the subtree rooted at u: the sum of their contribution matrices has the rank–one form

$$\mathbf{M}_u = \begin{bmatrix} \mathbf{m}_u \\ 1 \end{bmatrix} f_u^{-1} \begin{bmatrix} \mathbf{m}_u^\top & 1 \end{bmatrix},$$

supported only on $\mathrm{Desc}(u) \cup \{v\}$. Since different subtrees are disjoint, the reduced Laplacian on $\{v\} \cup \mathrm{Desc}(v)$ after eliminating all descendants can be obtained by adding, for each $u \in \mathrm{Desc}(v)$, the rank–one update $\mathbf{M}_u$ to the original block Laplacian.

Now consider eliminating v in this reduced graph. Let $\widehat{R} = \mathrm{Desc}(v)$ and let $\widehat{L}_{\widehat{R}}$ be the Laplacian on $\widehat{R}$ after all these updates. By construction, the new adjacency between v and the nodes in the subtree of u is the sum of the entries in the last column of $\mathbf{M}_u$ on the neighbors of v, i.e., $\sum_{x \in N(v) \cap \mathrm{Desc}(u)} \frac{\mathbf{m}_u[x]}{f_u}$. Therefore the updated adjacency vector from v to $\widehat{R} = \mathrm{Desc}(v)$ can be written as

$$\boldsymbol{a}_v = \sum_{u \in \mathrm{Desc}(v)} \left(\frac{\sum_{x \in N(v) \cap \mathrm{Desc}(u)} \mathbf{m}_u[x]}{f_u} \right) \mathbf{m}_u.$$

Applying Lemma 4.1 once more to node v, with $\widehat{L}_{\widehat{R}}$ and $\boldsymbol{a}_v$, we obtain

$$\mathbf{m}_v = \widehat{L}_{\widehat{R}}^{-1}\boldsymbol{a}_v = \sum_{u \in \mathrm{Desc}(v)} \left(\frac{\sum_{x \in N(v) \cap \mathrm{Desc}(u)} \mathbf{m}_u[x]}{f_u} \right) \mathbf{m}_u + \mathbf{e}_v,$$

where the term $\mathbf{e}_v$ accounts for the unit entry at v itself. This is exactly the first line in (6). The corresponding Schur complement at v is

$$f_v = d_v - \boldsymbol{a}_v^\top \mathbf{m}_v = d_v - \sum_{x \in N(v) \cap \mathrm{Desc}(v)} \mathbf{m}_v[x],$$

which is the second line in (6). Hence the pair $(\mathbf{m}_v, f_v)$ is obtained by aggregating the pairs of all descendants, as claimed. □

4.2 Index structure

A rooted tree is a hierarchical structure with a single distinguished node called the *root*. Each edge connects a node to one of its lower nodes, forming a parent–child relation. A *branch* refers to a point in the tree where a node has more than one child. For any node v, we use $\mathrm{Anc}(v)$ to denote the set of all its ancestors, that is, the

nodes on the path from the root to v (including v itself). Similarly, we use $\mathrm{Desc}(v)$ to denote the set of all its descendants, that is, the nodes in the subtree rooted at v (including v itself).

The index structure of BD-Index is to store the pair $(\mathbf{m}_v, f_v)$ for each node v. The cut hierarchy $\mathcal{H}$ is organized as a rooted tree, where each branching corresponds to a cut. The vertices within each cut set are arranged in a fixed order, consistent with the order defined in Section 3.2.

According to Lemma 4.1, each node v is associated with a vector $\mathbf{m}_v$ and a scalar f_v. The length of $\mathbf{m}_v$ equals the size of $\mathrm{Desc}(v)$, that is, the number of nodes in the subtree rooted at v. To avoid storing a full n-dimensional vector with many empty entries, we construct $\mathbf{m}_v$ as a compact vector of size $|\mathrm{Desc}(v)|$. We implicitly index its entries using the depth-first search (DFS) order of the hierarchy tree, so that the i-th entry of $\mathbf{m}_v$ corresponds to the i-th node in the DFS sequence under v. For example, in Figure 3(c), BD-Index on node v_2 stores: its parent node number v_8, a constant $f_{v_2} = 1.3$, and a vector $\mathbf{m}_{v_2}$. Specifically, $\mathbf{m}_{v_2}[0] = 1.0$ for v_2 itself, $\mathbf{m}_{v_2}[1] = 0.2$ for v_1, $\mathbf{m}_{v_2}[2] = 0.4$ for v_0, and $\mathbf{m}_{v_2}[3] = 0.3$ for v_3, where the entries follow the DFS order under v_2.

Lemma 4.3 (Space Complexity of the Index Structure). *Let h be the height of the hierarchy tree $\mathcal{H}$. The total space required to store all pairs $(\mathbf{m}_v, f_v)$ in the index structure is $O(n \cdot h)$.*

Proof. By construction, $|\mathbf{m}_v| = |\mathrm{Desc}(v)|$. Summing over all v counts each vertex x once for each ancestor of x:

$$\sum_v |\mathrm{Desc}(v)| = \sum_x |\mathrm{Anc}(x)| \le nh.$$

Each pair $(\mathbf{m}_v, f_v)$ thus costs $O(|\mathrm{Desc}(v)| + 1)$ space; the total is $O(n \cdot h)$. □

In real-world graphs, h is typically much smaller than n (as confirmed in our experiments), resulting in a small size of our index structure. For example, on Full-USA dataset ($n = 23,947,348$, $m = 28,854,319$), our index size is only 168GB.

4.3 Index building

The index construction begins with the hierarchy tree $\mathcal{H}$. Starting from the original graph, we recursively find small cut sets to divide the graph into subgraphs. For each cut set, the vertices are arranged into a short chain following the same order as defined in Section 3.2. Each subgraph separated by the cut set is then attached as a child subtree. Repeating this process recursively yields the complete hierarchy tree $\mathcal{H}$ (corresponding to Line 14 in Algorithm 3). As finding an optimal cut hierarchy is challenging, we adpot two heuristics: degree-based heuristic and recursive minimum cut heuristic.

Recursive minimum cut heuristic. The detail of the minimum cut-based method is illustrated in Algorithm 1. Given a graph $\mathcal{G}$, the algorithm constructs a hierarchy tree $\mathcal{H}$ in a recursive manner. For the input graph $\mathcal{G}$, we first call the function GetApproxCutSet to obtain an approximate vertex cut set. The vertices in this cut set are connected sequentially to form a chain in the hierarchy tree $\mathcal{H}$. Removing the cut set divides $\mathcal{G}$ into k subgraphs (Lines 9-10). For each subgraph, we again apply GetApproxCutSet to compute its own vertex cut set, which is linked into another chain and attached as a child subtree below the corresponding node of the previous chain in $\mathcal{H}$ (Lines 11-12). This recursive process continues until a subgraph contains only a single vertex, at which point GetApproxCutSet is no longer invoked (Lines 6-8, 13-14).

Algorithm 1: Minimum cut-based hierarchy construction

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Output: Hierarchy tree $\mathcal{H}$

1 **Function** `BuildHierarchyTree`($\mathcal{G}$):
2 Initialize an empty hierarchy tree $\mathcal{H}$;
3 `BuildSubtree`($\mathcal{G}$, *null*);
4 **return** $\mathcal{H}$;
5 **Function** `BuildSubtree`($\mathcal{G} = (\mathcal{V}, \mathcal{E})$, *parent node* p):
6 **if** $|\mathcal{V}| = 1$ **then**
7 Add a node v labeled by the single vertex in $\mathcal{V}$ under parent p in $\mathcal{H}$;
8 **return**;
9 $C \leftarrow$ GetApproxCutSet($\mathcal{G}$) ; `// Approximate minimum vertex cut returned by METIS [39]`
10 Add all vertices in C as a chain under parent p in $\mathcal{H}$;
11 Let b be the bottom node of this chain (if $C = \emptyset$, $b \leftarrow p$);
12 $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k\} \leftarrow$ connected components of the induced subgraph $\mathcal{G}[\mathcal{V} \setminus C]$;
13 **foreach** $\mathcal{G}_i$ **do**
14 `BuildSubtree`($\mathcal{G}_i, b$);

Algorithm 2: Minimum degree-based hierarchy construction [62]

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Output: Hierarchy tree $\mathcal{H}$

1 Initialize an empty hierarchy tree $\mathcal{H}$;
2 $\mathcal{G}' \leftarrow \mathcal{G}$;
3 **while** $\mathcal{V}(\mathcal{G}') \neq \emptyset$ **do**
4 select $v \in \mathcal{V}(\mathcal{G}')$ with minimum degree in $\mathcal{G}'$;
5 $\mathcal{N} \leftarrow u \in \mathcal{V}(\mathcal{G}') \mid (u, v) \in \mathcal{E}(\mathcal{G}')$;
6 create a new node h in $\mathcal{H}$ representing $v \cup \mathcal{N}$;
7 **if** $\mathcal{H}$ *is not empty* **then**
8 connect h to the node $h^\star \in \mathcal{H}$ that shares the largest overlap with $v \cup \mathcal{N}$;
9 add fill-in edges to make $\mathcal{N}$ a clique in $\mathcal{G}'$;
10 remove v and its incident edges from $\mathcal{G}'$;
11 **return** $\mathcal{H}$;

Finding the minimum vertex cut of a graph is NP-hard [13]. In our implementation (Line 9, function GetApproxCutSet), we adopt the widely used approximation algorithm MEITS [39, 40], whose computational complexity is $O(m)$. Since our hierarchy construction algorithm applies this process recursively at each level, the total running time is proportional to the number of edges times the hierarchy depth, giving an overall complexity of $O(m \cdot h)$.

Minimum degree heuristic. The detail of the minimum degree-based method [62] is illustrated in Algorithm 2. Starting from the input graph $\mathcal{G}$, the algorithm repeatedly removes the vertex with the smallest degree from the working graph $\mathcal{G}'$. At each step, it forms the set $v \cup \mathcal{N}$, where $\mathcal{N}$ is the set of neighbors of v, and creates

Algorithm 3: Index building algorithm of BD-Index

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Output: Hierarchy tree $\mathcal{H}$, $(\mathbf{m}_v, f_v)$ for each node $v \in \mathcal{V}$

1 **Function** BuildNode(v):
2 **if** *v has been built* **then return** $(\mathbf{m}_v, f_v)$;
3 $\mathbf{m}_v \leftarrow \emptyset, f_v \leftarrow 0$;
4 **foreach** $u \in \mathrm{Desc}(v)$ **do**
5 $(\mathbf{m}_u, f_u) \leftarrow$ BuildNode(u);
6 **foreach** $u \in \mathrm{Desc}(v)$ **do**
7 $C \leftarrow N(v) \cap \mathrm{Desc}(u)$;
8 $\sigma \leftarrow \sum_{x \in C} \mathbf{m}_u[x]$;
9 $\mathbf{m}_v \leftarrow \mathbf{m}_v + \frac{\sigma}{f_u} \cdot \mathbf{m}_u$;
10 $\mathbf{m}_v[v] \leftarrow 1$;
11 $C \leftarrow N(v) \cap \mathrm{Desc}(v)$;
12 $f_v \leftarrow d_v - \sum_{x \in C} \mathbf{m}_v[x]$;
13 **return** $(\mathbf{m}_v, f_v)$;
14 **Function** Main($\mathcal{G}$):
15 $\mathcal{H} \leftarrow$ BuildHierarchyTree($\mathcal{G}$);
16 $(\mathbf{m}_v, f_v)$ for all $v \in \mathcal{V} \leftarrow$ BuildNode($\mathcal{H}$.root);
17 **return** *Hierarchy tree $\mathcal{H}$, $(\mathbf{m}_v, f_v)$ for all $v \in \mathcal{V}$*;

a node h in the hierarchy tree $\mathcal{H}$ to represent this set. Then h is connected to the existing node in $\mathcal{H}$ that shares the most common vertices with it. Before removing v, the algorithm adds fill-in edges so that $\mathcal{N}$ becomes a clique in $\mathcal{G}'$, making $v \cup \mathcal{N}$ a small group of nodes that divides the remaining graph into several parts. Because each new node is linked to exactly one parent, $\mathcal{H}$ forms a tree. The worst-case time complexity of the minimum degree–based hierarchy construction is $O(n \cdot m)$, but in our experiments its practical running time remains efficient and acceptable for all graphs tested.

From a structural view, the top-down chain in $\mathcal{H}$ records the cut sets created step by step, while each branch growing from a node on this chain corresponds to a subgraph formed when that cut divides the graph. This process continues inside every subgraph until no vertices are left. As a result, the minimum degree-based construction produces a hierarchy tree whose vertical chains show the sequence of cut sets, and whose branches represent the subgraphs obtained after each cut, matching the organization of the algorithm.

For example, in Figure 3(a), we first find a minimum vertex cut $\{v_7, v_8\}$. After removing v_7 and v_8, the graph splits into two disconnected parts. We fix an order on the cut set, say (v_7, v_8), then root of the hierarchy tree $\mathcal{H}$ is v_7, and v_7 has a single child v_8. Next, we search for a minimum vertex cut in the two parts remaining. The cut sets are $\{v_2\}$ and $\{v_6\}$. We add v_2 and v_6 as the children of v_8 in $\mathcal{H}$. We repeat this process on every new part and stop when every remaining part is a single vertex. The resulting tree shown in Figure 3(b) is the hierarchy $\mathcal{H}$.

Index Building. Once the hierarchy is built, we perform a recursive traversal from the root. For each node v, if there exist nodes in $\mathrm{Desc}(v)$ whose indices have not yet been computed, the algorithm first processes those descendants (Lines 2–4). After all indices of $\mathrm{Desc}(v)$ are available, the index of v is computed using Equation 6, obtaining the pair $(\mathbf{m}_v, f_v)$ (Lines 5–11). This procedure continues until all nodes have been processed, resulting in the complete index structure for the entire graph. This process correspond to Figure 3(c).

Algorithm 4: Query processing algorithm with BD-Index

Input: Hierarchy tree $\mathcal{H}$, $(\mathbf{m}_v, f_v)$ for all $v \in \mathcal{V}$, query pair (s, t)
Output: Biharmonic distance $b(s, t)$

1 $\tilde{\boldsymbol{\tau}}_s[v], \tilde{\boldsymbol{\tau}}_t[v] \leftarrow 0$ for all $v \in \mathcal{V}$;
2 **foreach** $u \in \mathrm{Anc}(s)$ **do**
3 **foreach** $k \in \mathrm{Desc}(u)$ **do**
4 $\tilde{\boldsymbol{\tau}}_s[k] \leftarrow \tilde{\boldsymbol{\tau}}_s[k] + \frac{\mathbf{m}_u[s]}{f_u} \cdot \mathbf{m}_u[k]$;
5 **foreach** $u \in \mathrm{Anc}(t)$ **do**
6 **foreach** $k \in \mathrm{Desc}(u)$ **do**
7 $\tilde{\boldsymbol{\tau}}_t[k] \leftarrow \tilde{\boldsymbol{\tau}}_t[k] + \frac{\mathbf{m}_u[t]}{f_u} \cdot \mathbf{m}_u[k]$;
8 **return** $b(s, t) = \|\tilde{\boldsymbol{\tau}}_s - \tilde{\boldsymbol{\tau}}_t\|_2^2 - \frac{1}{n}\left(\mathbf{1}^\top(\tilde{\boldsymbol{\tau}}_s - \tilde{\boldsymbol{\tau}}_t)\right)^2$ according to Lemma 3.2;

Lemma 4.4 (Time Complexity of Index Building). *Let h be the height of the hierarchy tree $\mathcal{H}$. The total running time of the index building algorithm (Algorithm 3) is $O(n \cdot h \cdot (h + d_{max}))$.*

Proof. Lines 1–5 ensure that the procedure is invoked once for each vertex $v \in V$, thus contribute only $O(n)$ time. The dominant cost comes from Lines 6–9, whose total work over all nodes can be expressed as $\sum_{v \in V} \sum_{u \in \mathrm{Desc}(v)} (\deg(u) + |\mathrm{Desc}(u)|)$. For the first term, $\sum_{v \in V} \sum_{u \in \mathrm{Desc}(v)} \deg(u)$, by reindexing over descendants we obtain $\sum_{v \in V} \sum_{u \in \mathrm{Desc}(v)} \deg(u) = \sum_{u \in V} \deg(u) \cdot |\mathrm{Anc}(u)|$, where $\mathrm{Anc}(u)$ denotes the set of ancestors of u in the hierarchy. Since the hierarchy tree has height h, we have $|\mathrm{Anc}(u)| \le h$ for all u, and letting $d_{\max} = \max_{u \in V} \deg(u)$, this term is bounded by $O(nhd_{\max})$. For the second term, $\sum_{v \in V} \sum_{u \in \mathrm{Desc}(v)} |\mathrm{Desc}(u)|$, a similar reindexing yields $\sum_{v \in V} \sum_{u \in \mathrm{Desc}(v)} |\mathrm{Desc}(u)| = \sum_{w \in V} \sum_{v \in \mathrm{Anc}(w)} \sum_{u \in \mathrm{Anc}(v)} 1$. Each node has at most h ancestors, so the inner double sum is $O(h^2)$ for every w, and hence this term is bounded by $O(nh^2)$. Combining the two parts, the overall time complexity is $O(nhd_{\max} + nh^2)$. □

As discussed earlier, h is typically small relative to n, thus the time required to build BD-Index remains acceptable in practice. For example, on Full-USA, our index is constructed within 4.5 hours.

4.4 Query processing

With the constructed BD-Index, we can derive the contribution matrix $\widetilde{\mathbf{M}}_v$ for every node $v \in \mathcal{V}$. Based on these matrices, the deterministic random walk distribution $\tilde{\boldsymbol{\tau}}$ for any starting node s can be directly computed.

During query processing, given a query pair (s, t), we only need to access the stored pairs $(\mathbf{m}, f)$ along the ancestor sets $\mathrm{Anc}(s)$ and $\mathrm{Anc}(t)$. By accumulating the stored information of these ancestors, we obtain the deterministic random walk distributions $\tilde{\boldsymbol{\tau}}_s$ and $\tilde{\boldsymbol{\tau}}_t$ for the two source nodes (corresponding to Lines 2–7 in Algorithm 4). Finally, using the random walk formulation of the biharmonic distance in Lemma 3.2, we compute the value $b(s, t)$ (Line 8). For example, in Figure 3(d), the query for (v_2, v_5) needs access to BD-Index on $\mathrm{Anc}(v_2) \cup \mathrm{Anc}(v_5) = \{v_2, v_5, v_6, v_7, v_8\}$ and does not access $\{v_0, v_1, v_3, v_4\}$.

Lemma 4.5 (Time Complexity of Query Processing). *Let h be the height of the hierarchy tree $\mathcal{H}$. For a query pair (s,t), the time complexity of computing $b(s,t)$ using the index structure is $O(n \cdot h)$ in the worst case.*

Proof. Algorithm 4 accesses only $\mathrm{Anc}(s)$ and $\mathrm{Anc}(t)$. For each u on those two chains (each of length $\leq h$) it loops over $\mathrm{Desc}(u)$ to update $\boldsymbol{\tau}_s$ or $\boldsymbol{\tau}_t$. In the worst case, the sum of $|\mathrm{Desc}(u)|$ over all u on a chain is $O(n \cdot h)$. Therefore, the time complexity is $O(n \cdot h)$. □

Since each query traverses only a small portion of the hierarchy, most computations are localized to a few relevant subtrees. In practice, the query time is far below the worst-case bound. On Full-USA dataset, our average query time is less than 10^2s and is almost exact, while other methods require more than 10^5s.

Discussion: BD-Index **is an exact method.** Although we represent BD using random walks, it is important to emphasize that BD-Index is numerically exact up to floating-point precision. The random walk probabilities are computed deterministically in our approach, and no random walk sampling is involved at any stage. Consequently, BD-Index fundamentally differs from previous projection-based [81] and random walk-based [53] approximate methods. The only source of numerical error arises from floating-point precision, which is inherent to all methods for computing BD, since BD is a numerical quantity. The experiments further demonstrate that BD-Index is highly accurate: across all datasets, the relative error consistently remains below 10^{-9}.

5 EXPERIMENTS

5.1 Experimental setup

Datasets, query sets, and ground truth. All datasets used are listed in Table 1, which are publicly available from the SNAP [46] and DIMACS [63]. For each dataset, we uniformly sample 100 distinct source–target vertex pairs as the query set. Although BD-Index is an exact method, we require ground-truth results to evaluate prior approximate methods for BD [53, 81]. For ground-truth BD values, we exactly solve the underlying Laplacian linear systems using a Cholesky factorization of $\boldsymbol{L}$, i.e., a direct sparse linear solve [36].

Environment. All experiments are conducted on a Linux server with Intel Xeon E5-2680 v4 CPU and 512 GB RAM. Algorithms are implemented in C++ and compiled with g++ 7.5.0.

Different algorithms. We compare BD-Index with 2 SOTA exact methods. Except for Cholesky factorization, we also evaluate LapSolver, a Laplacian solver that combines approximate Gaussian elimination with a preconditioned conjugate gradient (PCG) phase [15, 21, 30, 44], with its accuracy parameter set to 10^{-15}. We also compare against 5 SOTA approximate methods for single-pair BD query, including (i) *Laplacian solver*-based method. RP [81] uses JL-sketch that solves $O(\log n)$ Laplacian systems in preprocessing and answers each query by computing an inner product. (ii) *Random walk*-based approaches. STW [53] is a sampled walk estimator that draws r pairs of truncated random walks up to length ℓ to form an additive-accuracy estimate. SWF [53] is a sampling-with-feedback variant that uses empirical-variance tests for early stopping. Push+ [53] is a local push method on a truncated expansion with a pair-dependent truncation length (we report Push+ and do not run Push separately). Unless otherwise noted, the default $\epsilon = 0.1$ follows the previous study [53]. For BD-Index, we use recursive minimum cut heuristic [39] to build the hierarchy tree $\mathcal{H}$.

Table 1: Datasets

Dataset	n	m	d_{max}	Type
Facebook	4,039	88,234	1,045	Social
CAIDA	26,475	53,381	2,628	Internet
Email-Enron	33,696	180,811	1,383	Social
NewYork	264,346	365,050	8	Road
DBLP	317,080	1,049,866	343	Collaboration
Amazon	334,863	925,872	549	Co-purchase
Road-PA	1,090,920	1,541,898	9	Road
Road-TX	1,393,383	1,921,660	12	Road
Road-CA	1,971,281	2,766,607	12	Road
Full-USA	23,947,348	28,854,319	9	Road

5.2 Query processing performance

We evaluate the query processing performance of different algorithms on the same set of 100 queries, reporting the average *query time* for all methods and *relative error* only for approximate methods. The comparison with exact solvers is reported in Fig. 4, while the comparison with approximate methods is shown in Figs. 6 and 7.

Comparison with exact methods. Figure 4 reports the query time of BD-Index against two exact baselines, LapSolver and a Cholesky decomposition, across all datasets. BD-Index is consistently 2–4 orders of magnitude faster than both exact methods. For example, on the large road network Full-USA, a single BD query requires about 1×10^6 seconds with Cholesky and about 4×10^5 seconds with LapSolver, which is clearly impractical for single-pair queries, whereas our index answers the same query in only 64 seconds. By constructing BD-Index, our approach makes exact BD queries for large numbers of single pairs on massive networks practically feasible.

Figure 5 reports the relative error of BD-Index and LapSolver when using the Cholesky factorization as the ground-truth baseline. As shown in the figure, the relative error of BD-Index remains below 10^{-9} on all datasets. The residual error of our method stems solely from floating-point roundoff: the index is stored in double-precision floating-point format (`double` in C++), which provides about 15 significant digits, and the local rounding errors introduced at each step of the hierarchy are propagated and accumulated, resulting in the observed 10^{-9}-level global error.

Comparison with approximate methods. As shown in Figs. 6 and 7, once the index has been constructed, BD-Index, although computing exact results, achieves at least an order-of-magnitude speedup over the fastest competing approximate method on all datasets. RP can only handle small graphs. STW, SWF, and Push+ are competitive on social graphs but their performance deteriorates sharply on road networks, where BD-Index is up to two additional orders of magnitude faster. For example, on the Amazon dataset, our method answers queries in about 6 seconds with exact accuracy, whereas other methods require at least 10^2 seconds to reach

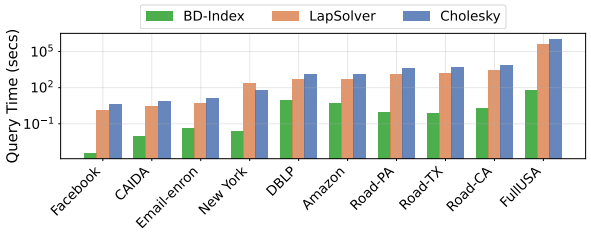


Figure 4: Query time compared with exact methods

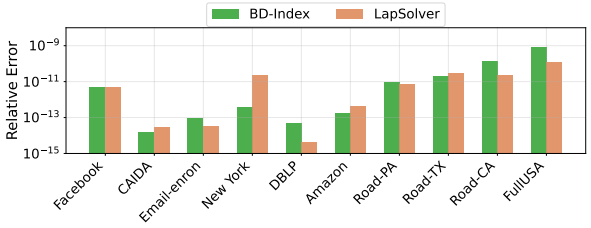


Figure 5: Relative error of BD-Index **and** LapSolver

a relative error of 10^{-4}. Overall, existing approximate methods incur very large errors on road networks and often fail to produce usable results, this is because they all rely on fast random walk mixing—precisely the bottleneck that our divide-and-conquer indexing strategy is easy to overcome.

5.3 Index analysis

The significant improvement of BD-Index over existing approaches is due to the sacrifice of index building time and index size. In this experiment, we report the index building time and index size on 10 datasets. The results can be found in Table 2.

Index size. Recall that the index size can be bounded by $O(n \cdot h)$. As can be seen, the indices are compact when the hierarchy tree height h is modest. As h increases on social networks, the stored vectors become much larger. On large road networks, despite very large n, the indices remain within a single-machine budget when h stays moderate. For example, the whole index of Full-USA takes 167, 597 MB with $h = 1,622$, because Full-USA is a large road network. For social graphs significantly larger than those in Table 2, the index size may exceed the available memory on our machine, thus the build cannot complete. In conclusion, our BD-Index is highly efficient for large graphs with small h (e.g., road networks). However, for graphs with large h, the $O(n \cdot h)$ storage cost remains a limitation, indicating a clear direction for future optimization.

Index building time. The index building time follows the same pattern. Most road networks with moderate tree hierarchy h complete index construction in minutes. When h is large, social/information graphs incur substantial offline time. For example, Amazon takes 78,957 seconds and DBLP takes 368,690 seconds. However, Road-PA takes only 198 seconds even though this graph is an order of magnitude larger than Amazon and DBLP. These results are consistent the $n \cdot h \cdot (h + d_{max})$ dependence for the current implementation.

Table 2: Index Performance on all datasets

Datasets	Graph Size (MB)	Hierarchy tree height h	Index Size (MB)	Construction Time (secs)
Facebook	0.8	401	5	0.55
CAIDA	0.5	265	37	44
Email-enron	1.8	2,455	299	1,549
NewYork	4.6	295	352	2.34
DBLP	13	18,466	33,470	368,690
Amazon	12	11,805	21,358	78,957
Road-PA	21	817	5,496	198
Road-TX	26	607	4,318	77
Road-CA	40	857	9,318	253
FullUSA	470	1,622	167,597	16,023

BD-Index achieves theoretically exact answers and enables single-pair queries that are one to three orders of magnitude faster than the fastest approximate method, and two to four orders of magnitude faster than existing exact solvers, at the cost of an offline index whose memory footprint scales with n and h. In practice, it is efficient on very large graphs with small h (e.g., road networks) and on medium-scale graphs with large h (e.g., social networks).

5.4 Comparison of different tree hierarchy $\mathcal{H}$

In this experiment, we compare different tree hierarchy $\mathcal{H}$ to cut the graph into pieces. Since both the time and space complexity of our algorithm depend on the tree height h, the structure of the tree hierarchy $\mathcal{H}$ has a direct impact on the overall efficiency. In Section 4.3, we introduced two heuristic strategies for constructing $\mathcal{H}$: *minimum cut-based* hierarchy construction and *minimum degree-based* hierarchy construction. Figures 8, 9, and 10 report the query time, index construction time, and index size, respectively, under these two hierarchy constructions across all datasets. Across the board, the minimum cut-based hierarchy yields shorter query time, lower construction time, and a smaller index on most datasets.

Table 3 further compares the structural properties of the resulting hierarchies in terms of the tree height h and average label size s, which is the average number of non-zero elements stored in our index. The minimum cut-based strategy typically produces hierarchies with smaller tree height h and lower average label size s. This structurally more compact hierarchy directly translates into the superior query and indexing performance observed in our experiments.

6 CASE STUDIES

In this section, we conduct two case studies to demonstrate the effectiveness of BD in two graph mining tasks.

6.1 Critical link identification on road networks

We apply BD to identify *critical links* in a road network [80]. For each edge (s, t), $b(s, t)$ acts as an *edge centrality* measure: larger $b(s, t)$ indicate links that are more critical for preserving network connectivity. We evaluate this interpretation on a 3×3 km road network of Philadelphia extracted from OpenStreetMap, which contains real urban infrastructure such as highways, local streets, and bridges. We keep the largest connected component and treat bridges and river-crossing segments as weak ground-truth critical

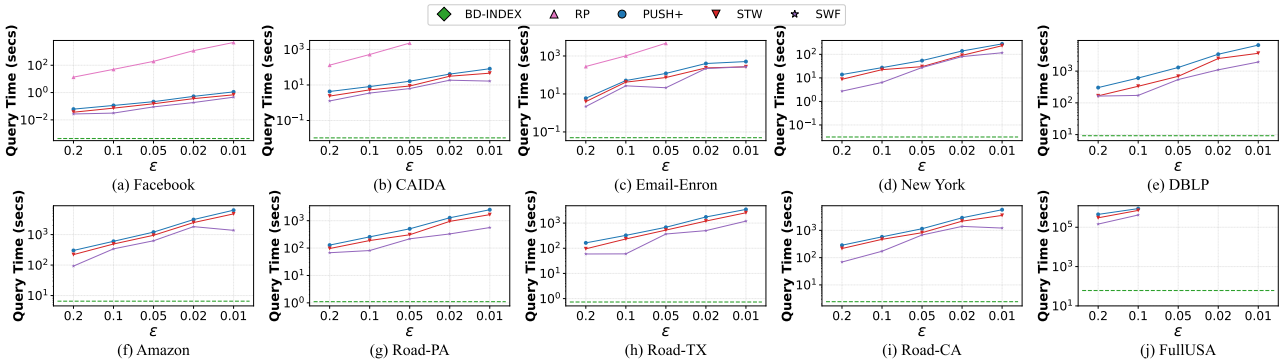


Figure 6: Query time compared with different approximate methods

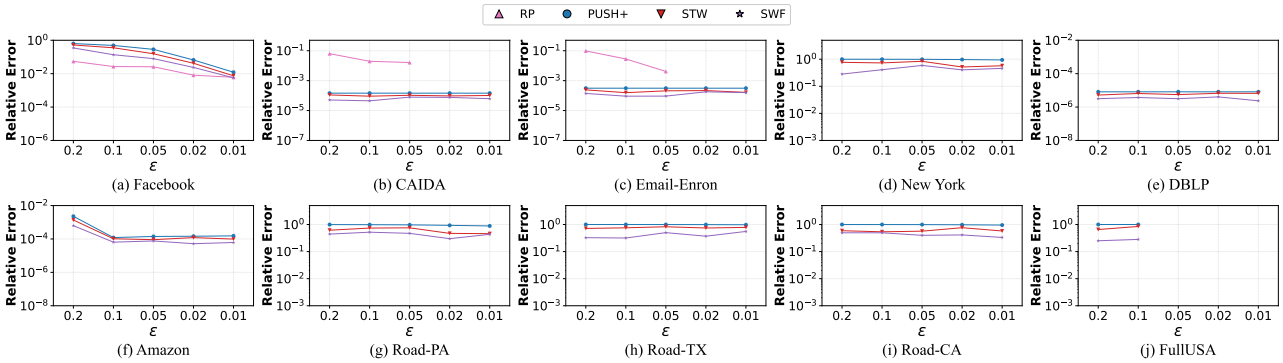


Figure 7: Query relative error of different approximate methods

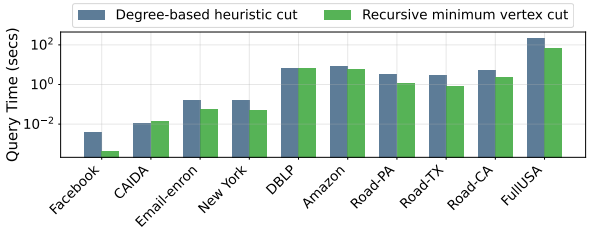


Figure 8: Query time with different tree hierarchy $\mathcal{H}$

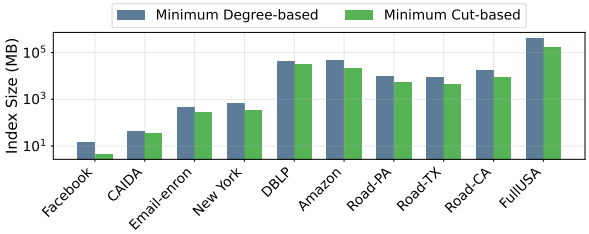


Figure 10: Index size with different tree hierarchy $\mathcal{H}$

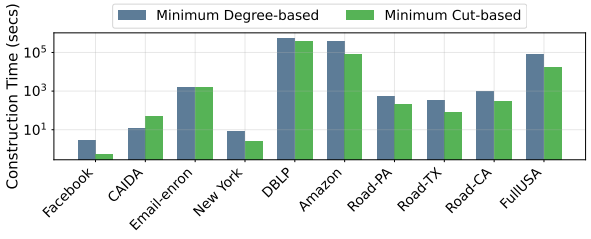


Figure 9: Index building time with different tree hierarchy $\mathcal{H}$

links. We compare BD against three alternative edge centralities: ER (effective resistance using inverse road length), EB (edge betweenness with road length as impedance), and PR (an edge score derived from node PageRank computed on inverse length). In Fig. 11, edges are visualized from blue (low centrality) to red (high centrality). BD highlights all bridge segments while remaining low on internal roads; ER also highlights bridges but assigns similarly high scores to many adjacent streets, showing limited contrast; and both EB and PR fail to consistently detect all bridges. Overall, BD most cleanly isolates the visually critical links in the network.

To further quantify this effect, we iteratively remove top-ranked edges under each centrality until the total removed length reaches 5% or 10% of the network, and measure the impact on connectivity using three metrics: LCC, the fraction of nodes in the largest

Table 3: Comparison of hierarchy tree height h and average label size s under different tree hierarchy $\mathcal{H}$

Dataset	Hierarchy tree height h		Average label size s	
	Minimum cut-based	Minimum degree-based	Minimum cut-based	Minimum degree-based
Facebook	401	741	154	486
CAIDA	265	289	181	222
Email-Enron	2,455	2,397	1,166	1,895
New York	295	767	174	346
DBLP	18,466	20,109	13,836	17,582
Amazon	11,805	21,394	8,360	18,770
Road-PA	817	2,034	662	1,146
Road-TX	607	1,530	419	867
Road-CA	857	1,715	624	1,167
FullUSA	1,622	3,976	917	2,268

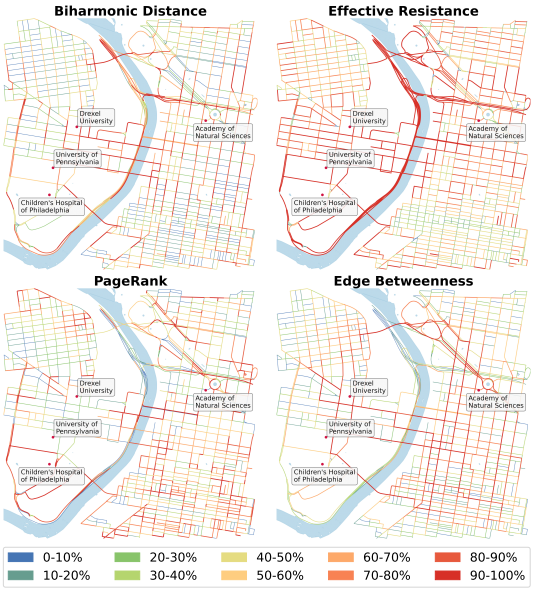


Figure 11: Comparison of different edge centralities on the Philadelphia network, edges are marked with colors from blue (low centrality value) to red (high centrality value).

connected component; #Comp, the number of connected components; and Reach, the fraction of reachable origin–destination pairs among 10^3 randomly sampled pairs. Lower values of LCC and Reach, together with higher #Comp, indicate stronger disruption. As reported in Table 4, under both removal budgets, BD yields the lowest LCC and Reach and the highest #Comp, confirming that it focuses on true structural bottlenecks. PR performs second best, while ER and EB have the weakest impact on network connectivity. These results demonstrate the effectiveness of BD for identifying critical links in real road networks.

6.2 Over-squashing mitigation in GNN

Graph Neural Networks (GNNs) often suffer from *over-squashing* [3, 11, 70], where information from many distant nodes is forced through a few narrow connections, weakening message passing and degrading accuracy. Prior work [11] has shown that BD can be

Table 4: Comparison of network connectivity (LCC, #Comp, Reach) after removing the top 5% / 10% of road length ranked by different edge centrality measures

Budget	Method	LCC	#Comp	Reach
5%	BD	**0.675**	12	**0.528**
	ER	0.994	6	0.991
	EB	0.993	4	0.983
	PR	0.988	12	0.975
10%	BD	**0.664**	29	**0.481**
	ER	0.986	14	0.981
	EB	0.975	12	0.944
	PR	0.971	27	0.940

Table 5: BD-guided over-squashing mitigation in GNNs on Chameleon [64] node classification

Graph	#Edges	Δ#Edges (%)	TotalER ($\times 10^6$)	ΔTotalER (%)	Acc.	ΔAcc (%)
Baseline	23370	+0.0	7.52	+0.00	62.28	+0.00
BD-guided rewired	23603	+1.0	7.23	-3.93	62.50	+0.35
	23837	+2.0	7.22	-4.06	62.72	+0.70
	24071	+3.0	7.18	-4.47	62.75	+0.75

used to identify and alleviate such bottlenecks via graph rewiring, thereby mitigating over-squashing and improving GNN performance. Here, we perform a simple replication-style study on the Chameleon [64] dataset (a standard benchmark constructed from Wikipedia pages on diverse topics), applying the same BD-guided rewiring strategy as in [11]. Our goal is not to design a new GNN architecture, but to validate the role of BD in reducing over-squashing.

A common theoretical indicator of over-squashing is the graph's *total effective resistance* (TotalER): higher TotalER implies more constrained information flow. We therefore use TotalER as a measure for the severity of over-squashing. Starting from the original graph (Baseline), we iteratively add a small number of new edges in up to three rounds. In each round, we select node pairs with the largest BD values from a pool of long-range pairs and add 1% new edges. After each step, we record both TotalER and test accuracy.

As shown in Table 5, adding only 1% of BD-guided edges already reduces TotalER by about 3.9% while increasing accuracy by roughly 0.35 percentage points. Increasing the addition to 2–3% of BD-guided edges further decreases TotalER by up to 4.5% and yields accuracy gains of around 0.70–0.75 percentage points overall. The absolute improvement is modest because our setup deliberately uses a very simple two-layer GNN with basic training (rather than sophisticated architectures or heavy hyperparameter tuning), and GNN design is not the main focus of this paper. Nevertheless, these results demonstrate that BD provides an effective signal for rewiring to mitigate over-squashing: selecting a small fraction of additional edges according to BD already relieves communication bottlenecks (lower TotalER) and improves predictive performance.

7 RELATED WORK

Several works have examined the mathematical properties of the Laplacian pseudoinverse [8, 16, 24, 36, 41, 79]. Early studies revealed

its close connection to the *resistance distance* (ER) [41] and the theory of random walks on graphs [24, 47]. Later analyses explored its spectral interpretation, linking resistance measures to Laplacian eigenvalues and network coherence [6, 16, 73, 74, 79]. Quantities derived from the Laplacian pseudoinverse, including the effective resistance (ER), personalized PageRank (PR), and biharmonic distance (BD), are widely used in graph learning, network analysis, and spectral algorithms [9, 55, 65, 71, 72, 78]. ER measures connectivity and robustness; its total value reflects how strongly a network is connected, and has been used for robustness optimization, sparsification, and resistance-based embeddings [11, 48, 51, 54, 61, 67]. PR quantifies the influence of a node relative to a source and underlies many ranking, recommendation, and diffusion methods [35, 56, 66, 77]. BD has been applied in clustering, centrality, and geometric processing, with recent studies demonstrating its advantages over ER [9, 38, 52, 53, 80–82]. Physical and dynamical systems perspectives linked these quantities to synchronization and control in large networks [73, 74]. These studies focus on applications, whereas our work emphasizes computation.

Many recent studies employ random walk–based methods to compute quantities derived from the Laplacian pseudoinverse [7, 12, 45, 47, 69]. The computation of ER can be interpreted as counting the expected number of times a random walk starting at one node visits another [51, 54, 60], while PR corresponds to a random walk with probabilities [32, 35, 75]. These works have developed efficient random walk algorithms for such quantities. Beyond purely probabilistic estimators, spectral and randomized projection techniques such as the Johnson–Lindenstrauss transform [2, 57] and nearly-linear Laplacian solvers [21, 30, 68] have also been adopted. However, such methods cannot be directly applied to the computation of BD, since BD measures the distance between the distributions of two random walks. This form makes the problem substantially more challenging than ER or PR computations.

Another related direction studies index-based shortest-path distance querying. Beyond classical online algorithms such as Dijkstra, bidirectional search, and A* search [23, 33], many methods precompute hopsets and distance labelings so that, after adding a sparse set of shortcuts, all pairs are reachable within a bounded number of hops while (approximate) distances are preserved [19, 25, 31]. This leads to distance oracles and 2-hop labels with strong guarantees, including bounds on general graphs and on graphs with small treewidth, highway dimension, or skeleton dimension [1, 5, 20, 22, 28, 34, 43, 49]. Theoretical advances have been instantiated in practical systems such as TEDI [76], pruned landmark and independent-set–based labelings [4, 29], hop-doubling and hierarchical 2-hop schemes [14, 37, 58], projected vertex separators and hierarchical cut labellings [17, 26, 27], as well as dynamic and learning-based indexes [42, 59, 83, 84]. Our goal is also to build an index for fast pairwise distance queries, but BD is a Laplacian-based distance between random-walk distributions. It does not satisfy the cut and bounded-hop properties that these indexes exploit, so hopset and labeling techniques cannot be reused; we instead design new hierarchical decompositions and labels tailored to encoding biharmonic distances via the Laplacian pseudoinverse.

8 CONCLUSION

In this work, we revisited the biharmonic distance (BD) from an algorithmic perspective and addressed the challenge of efficient single-pair BD queries on large graphs. We introduced BD-Index, a divide-and-conquer index structure that leverages small graph separators to decompose BD computation into independent local problems. The index can be built in $O(n \cdot h \cdot (h + d_{max}))$ time, requires $O(n \cdot h)$ space, and answers each query in $O(n \cdot h)$ time, where h is the height of the hierarchy tree—typically much smaller than n. Extensive experiments on large-scale datasets demonstrate that BD-Index achieves order-of-magnitude speedups over state-of-the-art approximate methods while achieving theoretically exact accuracy. Beyond efficiency, we also showcased the utility of BD in downstream applications, including identifying critical links in road networks and mitigating the over-squashing problem in graph neural networks.

REFERENCES

[1] Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. 2010. Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 782–793.

[2] Dimitris Achlioptas. 2003. Database-Friendly Random Projections: Johnson–Lindenstrauss with Binary Coins. *J. Comput. System Sci.* 66, 4 (2003), 671–687.

[3] S. Akansha. 2025. Over-squashing in Graph Neural Networks: A comprehensive survey. *Neurocomputing* 642 (2025), 130389.

[4] 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 2013 ACM SIGMOD International Conference on Management of Data*. 349–360.

[5] Stephen Alstrup, Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Ely Porat. 2016. Sublinear Distance Labeling. In *Proceedings of the 24th Annual European Symposium on Algorithms (ESA)*.

[6] B. Bamieh, M. Jovanović, P. Mitra, and S. Patterson. 2012. Coherence in Large-Scale Networks: Dimension-Dependent Limitations of Local Feedback. *IEEE Trans. Automat. Control* 57, 9 (2012), 2235–2249.

[7] Siddhartha Banerjee and Peter Lofgren. 2015. Fast bidirectional probability estimation in Markov models. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 28. 1423–1431.

[8] R. B. Bapat. 2014. *Graphs and Matrices (2nd Edition)*. Springer.

[9] Matthew Black, Florian Dörfler, and Claudia Gratton. 2024. Biharmonic Distance of Graphs and Its Higher-Order Variants: Analytical Properties with Applications to Centrality and Clustering. In *Proceedings of the 2024 International Conference on Complex Networks*.

[10] 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 (ICML'24)*. Article 165, 26 pages.

[11] 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 (ICML)*. PMLR, 2528–2547.

[12] Marco Bressan, Enoch Peserico, and Luca Pretto. 2018. Sublinear algorithms for local graph centrality estimation. In *59th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 709–718.

[13] Thang Bui and Curt Jones. 1992. Finding good approximate vertex and edge partitions is NP-hard. *Inform. Process. Lett.* 42, 3 (1992), 153–159.

[14] Lijun Chang, Jeffrey Xu Yu, Lu Qin, Hong Cheng, and Miao Qiao. 2012. The Exact Distance to Destination in Undirected World. *VLDB Journal* 21, 6 (2012), 869–888.

[15] Chao Chen, Tianyu Liang, and George Biros. 2021. RCHOL: Randomized Cholesky Factorization for Solving SDD Linear Systems. *SIAM Journal on Scientific Computing* 43, 6 (2021), C411–C438.

[16] Guandong Chen and Zhongzhi Zhang. 2007. Resistance Distance and the Normalized Laplacian Spectrum. *Physica A* 385, 2 (2007), 761–772.

[17] Zitong Chen, Ada Wai-Chee Fu, Minhao Jiang, Eric Lo, and Pengfei Zhang. 2021. P2H: Efficient Distance Querying on Road Networks by Projected Vertex Separators. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*. 313–325.

[18] Fan R. K. Chung. 1997. *Spectral Graph Theory*. CBMS Regional Conference Series in Mathematics, Vol. 92. American Mathematical Society. https://doi.org/10.

1090/cbms/092
[19] Edith Cohen. 1994. Polylog-time and Near-linear Work Approximation Scheme for Undirected Shortest Paths. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*. 16–26.
[20] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2002. Reachability and Distance Queries via 2-hop Labels. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 937–946.
[21] Michael B. Cohen and Jonah Sherman. 2014. Solving SDD Linear Systems in Nearly m sqrtlog n Time. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC)*. 343–352.
[22] Vincent Cohen-Addad, Søren Dahlgaard, and Christian Wulff-Nilsen. 2017. Fast and Compact Exact Distance Oracle for Planar Graphs. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 962–973.
[23] Dennis de Champeaux and Lenie Sint. 1977. An Optimality Theorem for a Bi-Directional Heuristic Search Algorithm. In *The Computer Journal*, Vol. 20. 148–150.
[24] Peter G. Doyle and J. Laurie Snell. 1984. *Random Walks and Electric Networks*. Mathematical Association of America.
[25] Michael Elkin and Ofer Neiman. 2016. Hopsets with Constant Hopbound, and Applications to Approximate Shortest Paths. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 128–137.
[26] Muhammad Farhan, Henning Koehler, and Qing Wang. 2023. Hierarchical Cut Labelling: Scaling Up Distance Queries on Road Networks. *Proceedings of the ACM on Management of Data* 1, 4 (2023), 244:1–244:25.
[27] Muhammad Farhan, Henning Koehler, and Qing Wang. 2025. Dual-Hierarchy Labelling: Scaling Up Distance Queries on Dynamic Road Networks. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 35:1–35:25.
[28] Arash Farzan and Shahin Kamali. 2011. Compact Navigation and Distance Oracles for Graphs with Small Treewidth. In *Proceedings of the 38th International Colloquium on Automata, Languages, and Programming (ICALP)*. 268–280.
[29] Ada Wai-Chee Fu, Huanhuan Wu, James Cheng, and Raymond Chi-Wing Wong. 2013. IS-LABEL: An Independent-Set Based Labeling Scheme for Point-to-Point Distance Querying. *VLDB Journal* 22, 4 (2013), 457–468.
[30] Yuan Gao, Rasmus Kyng, and Daniel A. Spielman. 2023. Robust and Practical Solution of Laplacian Equations by Approximate Elimination. *arXiv preprint arXiv:2307.05911* (2023).
[31] Cyril Gavoille, David Peleg, Stephane Perennes, and Ran Raz. 2001. Distance Labeling in Graphs. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 210–219.
[32] David F. Gleich. 2015. PageRank Beyond the Web. *SIAM Rev.* (2015), 321–363.
[33] Andrew V. Goldberg and Chris Harrelson. 2005. Computing the Shortest Path: A Search Meets Graph Theory. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 156–165.
[34] Siddharth Gupta, Adrian Kosowski, and Laurent Viennot. 2019. Exploiting Hopsets: Improved Distance Oracles for Graphs of Constant Highway Dimension and Beyond. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*.
[35] Taher H. Haveliwala and Sepandar D. Kamvar. 2003. *The Second Eigenvalue of the Google Matrix*. Technical Report. Stanford InfoLab.
[36] Roger A. Horn and Charles R. Johnson. 2013. *Matrix Analysis (2nd ed.)*. Cambridge University Press.
[37] Minhao Jiang, Ada Wai-Chee Fu, Raymond Chi-Wing Wong, and Yanyan Xu. 2014. Hop Doubling Label Indexing for Point-to-Point Distance Querying on Scale-Free Networks. *VLDB Journal* 7, 12 (2014), 1203–1214.
[38] 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 (ICDM)*. 339–348.
[39] George Karypis and Vipin Kumar. 1997. METIS—A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes and Computing Fill-Reducing Ordering of Sparse Matrices. (1997).
[40] George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing* (1998), 359–392.
[41] D. J. Klein and M. Randić. 1993. Resistance Distance. *Journal of Mathematical Chemistry* 12, 1 (1993), 81–95.
[42] Henning Koehler, Muhammad Farhan, and Qing Wang. 2025. Stable Tree Labelling for Accelerating Distance Queries on Dynamic Road Networks. In *Proceedings of the 2025 International Conference on Extending Database Technology (EDBT)*. 477–489.
[43] Adrian Kosowski and Laurent Viennot. 2017. Beyond Highway Dimension: Small Distance Labels Using Tree Skeletons. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1462–1478.
[44] Rasmus Kyng and Sushant Sachdeva. 2016. Approximate Gaussian Elimination for Laplacians: Fast, Sparse, and Simple. In *Proceedings of the 57th IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE. https://doi.org/10.1109/FOCS.2016.68
[45] Christina E. Lee, Asuman Ozdaglar, and Devavrat Shah. 2013. Computing the stationary distribution locally. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 26. 1376–1384.
[46] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection.
[47] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. 2017. *Markov Chains and Mixing Times (2nd ed.)*. American Mathematical Society.
[48] Lawrence Li and Sushant Sachdeva. 2023. A New Approach to Estimating Effective Resistances and Counting Spanning Trees in Expander Graphs. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2728–2745.
[49] Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2020. Scaling Up Distance Labeling on Graphs with Core-Periphery Properties. *Proceedings of the ACM on Management of Data* 1, 1 (2020), 1367–1381.
[50] 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. *SIGMOD* (2023).
[51] 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. *Proc. ACM Manag. Data* (2024).
[52] Yaron Lipman, Raif M. Rustamov, and Thomas A. Funkhouser. 2010. Biharmonic distance. *ACM Transactions on Graphics* 29, 1 (2010), 1–11.
[53] 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 (KDD '24)*. 1887–1897.
[54] Zhiqiang Liu and Wenjian Yu. 2023. Computing Effective Resistances on Large Graphs Based on Approximate Inverse of Cholesky Factor. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1–6.
[55] Linyuan Lü and Tao Zhou. 2011. Link Prediction in Complex Networks: A Survey. *Physica A* 390, 6 (2011), 1150–1170.
[56] Michael W. Mahoney. 2012. A Local Spectral Method for Graphs: With Applications to Semi-Supervised Learning and Partitioning. *Journal of Machine Learning Research (JMLR)* 13 (2012), 2339–2364.
[57] Rajeev Motwani and Prabhakar Raghavan. 1995. Randomized Algorithms. *SIGACT News* 26, 3 (1995), 48–50.
[58] 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 2018 ACM SIGMOD International Conference on Management of Data*. 709–724.
[59] Dian Ouyang, Long Yuan, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. 2020. Efficient Shortest Path Index Maintenance on Dynamic Road Networks with Theoretical Guarantees. *Proceedings of the VLDB Endowment* 13, 5 (2020), 602–615.
[60] 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 and Data Mining (KDD)*. ACM, 1329–1338.
[61] M. Predari et al. 2023. Greedy optimization of resistance-based graph robustness. *Social Network Analysis and Mining* 13, 1 (2023).
[62] Neil Robertson and P.D Seymour. 1991. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B* (1991), 153–190.
[63] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*.
[64] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2021. Multi-scale Attributed Node Embedding. *Journal of Complex Networks* (2021), cnab014.
[65] Yutaka Shimada, Yoshito Hirata, Tohru Ikeguchi, and Kazuyuki Aihara. 2016. Graph Distance for Complex Networks. *Scientific Reports* 6 (2016), 34944.
[66] Disha Shur, Yufan Huang, and David F. Gleich. 2023. A flexible PageRank-based graph embedding framework. *Applied and Computational Topology* 7, 1 (2023).
[67] Daniel A. Spielman and Nikhil Srivastava. 2008. Graph Sparsification by Effective Resistances. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*. 563–568.
[68] 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.
[69] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. In *Proceedings of the Sixth International Conference on Data Mining (ICDM)*. IEEE, 613–622.
[70] Jake Topping, Francesco Di Giovanni, Benjamin Chamberlain, Xiaowen Dong, and Michael Bronstein. 2022. Understanding over-squashing and bottlenecks on graphs via curvature.
[71] Thanh Tran, Xinyue Liu, Kyumin Lee, and Xiangnan Kong. 2019. Signed Distance-based Deep Memory Recommender. In *Proceedings of The Web Conference (WWW)*. 1841–1852.
[72] 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 (WWW)*. 2697–2703.
[73] Melvyn Tyloo, Tommaso Coletta, and Philippe Jacquod. 2018. Robustness of Synchrony in Complex Networks and Generalized Kirchhoff Indices. *Physical Review Letters* 120, 8 (2018), 084101.

[74] M. Tyloo, L. Pagnier, and P. Jacquod. 2019. The Key Player Problem in Complex Oscillator Networks and Electric Power Grids: Resistance Centralities Identify Local Vulnerabilities. *Science Advances* 5, 11 (2019), eaaw8359.

[75] 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 (KDD)*. ACM, 505–514.

[76] Fang Wei. 2010. TEDI: Efficient Shortest Path Query Answering on Graphs. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. 99–110.

[77] W. Xie and K. Lang. 2015. Edge-Weighted Personalized PageRank: Breaking A Decade of Barriers. In *Proceedings of the 24th International Conference on World Wide Web (WWW)*. 727–737.

[78] Renchi Yang. 2022. Efficient and Effective Similarity Search over Bipartite Graphs. In *Proceedings of The Web Conference (WWW)*. 308–318.

[79] Yujun Yang and Douglas J. Klein. 2013. A Recursion Formula for Resistance Distances and Its Applications. *Discrete Applied Mathematics* 161, 16-17 (2013), 2702–2715.

[80] 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 (IJCAI-18)*. 3620–3626.

[81] Yuhao Yi, Bingjia Yang, Zhongzhi Zhang, and Stacy Patterson. 2018. Biharmonic distance and performance of second-order consensus networks with stochastic disturbances. In *Proceedings of the American Control Conference (ACC)*. 4943–4950.

[82] 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.

[83] Yikai Zhang and Jeffrey Xu Yu. 2022. Relative Subboundedness of Contraction Hierarchy and Hierarchical 2-Hop Index in Dynamic Road Networks. In *Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*. 1992–2005.

[84] Bolong Zheng, Yong Ma, Jingyi Wan, Yongyong Gao, Kai Huang, Xiaofang Zhou, and Christian S. Jensen. 2023. Reinforcement Learning Based Tree Decomposition for Distance Querying in Road Networks. In *Proceedings of the 2023 IEEE 39th International Conference on Data Engineering (ICDE)*. 1678–1690.