Efficient and Provable Effective Resistance Computation on Large Graphs: an Index-based Approach

Meihao Liao[†], Junjie Zhou[†], Rong-Hua Li[†], Qiangqiang Dai[†], Hongyang Chen[‡], Guoren Wang[†]
[†]Beijing Institute of Technology, China [‡]Zhejiang Lab, China
mhliao@bit.edu.cn,zjjyyyk@bit.edu.cn,lironghuabit@126.com,qiangd66@gmail.com
dr.h.chen@ieee.org,wanggrbit@gmail.com

ABSTRACT

Effective resistance (ER) is a fundamental metric for measuring node similarities in a graph, and it finds applications in various domains including graph clustering, recommendation systems, link prediction, and graph neural networks. The state-of-the-art algorithm for computing effective resistance relies on a landmark technique, which involves selecting a node that is easy to reach by all the other nodes as a landmark. The performance of this technique heavily depends on the chosen landmark node. However, in many real-life graphs, it is not always possible to find an easily reachable landmark node, which can significantly hinder the algorithm's efficiency. To overcome this problem, we propose a novel multiple landmarks technique which involves selecting a set of landmark nodes $\mathcal{V}_l$ such that the other nodes in the graph can easily reach any one of a landmark node in $\mathcal{V}_l$. Specifically, we first propose several new formulas to compute ER with multiple landmarks, utilizing the concept of Schur complement. These new formulas allow us to pre-compute and maintain several small-sized matrices related to $\mathcal{V}_l$ as a compact index. With this powerful index technique, we demonstrate that both single-pair and single-source ER queries can be efficiently answered using a newly-developed $\mathcal{V}_l$-absorbed random walk sampling or $\mathcal{V}_l$-absorbed push technique. Comprehensive theoretical analysis shows that all proposed index-based algorithms achieve provable performance guarantees for both single-pair and single-source ER queries. Extensive experiments on 5 real-life datasets demonstrate the high efficiency of our multiple landmarks-based index techniques. For instance, our algorithms, with a 1.5 GB index size, can be up to 4 orders of magnitude faster than the state-of-the-art algorithms while achieving the same accuracy on a large road network.

ACM Reference Format:
Meihao Liao[†], Junjie Zhou[†], Rong-Hua Li[†], Qiangqiang Dai[†], Hongyang Chen[‡], Guoren Wang[†] . 2022. Efficient and Provable Effective Resistance Computation on Large Graphs: an Index-based Approach. In *Proceedings of SIGMOD '24: International Conference on Management of Data (SIGMOD '24).* ACM, New York, NY, USA, 25 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

SIGMOD '24, June 09–15, 2024, Santiago, Chile

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Effective resistance [56] is a fundamental metric to measure node similarities of a graph. Effective resistance is a fundamental metric to measure node similarities of a graph. Given an undirected graph $\mathcal{G}$ and two nodes s, t, the effective resistance between s and t, denoted by $r(s,t)$, is equivalent to (up to a constant factor) the expected number of steps taken by a random walk starting from s, visiting t, and coming back to s [56]. Intuitively, a small $r(s,t)$ indicates a high similarity between s and t. This is because a small effective resistance implies that it is relatively easy for a random walk starting from s to reach t, and vice versa. In other words, the nodes s and t are more likely to be well-connected or have similar neighbors in the graph, suggesting a high degree of similarity between them. Compared to the classic shortest-path distance metric, effective resistance is often more robust to noise, such as the deletion or insertion of a small number of edges in a graph, as it considers all possible paths [31, 56].

The effective resistance metric has found wide applications in graph data management and mining, including clustering in geo-social networks [51], long-tail recommendation systems [69], link prediction in social networks [49], and anomaly detection in time-varying graphs [55]. Recently, it has also been employed to analyze the over-squashing problem in graph neural networks [34, 57]. These applications leverage the power of effective resistance to capture and quantify node similarities in various graph-based scenarios.

Despite many efforts have been made to compute effective resistance in various studies [31, 45, 54, 67], there is still a lack of an efficient and provable algorithm for computing effective resistance on large graphs. Recently, Peng et al. [45] proposed several efficient algorithms to compute single-pair effective resistance using an L-step random-walk sampling technique. They proved that their algorithms achieves sub-linear time complexity with respect to the size of the graph. Building upon this, Yang et al. [67] further improved the algorithm by reducing the required length of L while maintaining accuracy. However, a major limitation of these algorithms is that when the effective resistance $r(s,t)$ is large, the random walk length L can still be very large. Additionally, these algorithms can only compute single-pair effective resistance and would need to be invoked $n-1$ times to compute the single-source effective resistance, resulting in significant computational cost.

In addition to the work by Yang et al. [67], Liao et al. [31] proposed an alterative random walk sampling algorithm for effective resistance computation, which utilizes a landmark node. This algorithm is considered as the state-of-the-art for both single-pair and single-source effective resistance caculations on large real-life graphs, as demonstrated in our experiments. The key idea behind their algorithm is as follows: rather than performing random walks

from the source node s to the target node t, random walks are conducted from both s and t towards an easily reachable node (e.g., the highest-degree node), referred to as the landmark node. By terminating the random walk once it hits the landmark node, the random walk sampling process can be executed quickly. The estimated value of $r(s,t)$ is then derived based on these random walks. Additionally, the landmark idea is also extended to develop a random spanning forest sampling technique to compute both single-pair and single-source effective resistance. A notable advantage of their algorithm for computing single-pair effective resistance is that it only requires exploration of a small portion of the graph, resulting in high efficiency. However, a limitation of this approach is that its performance heavily relies on the selection of an appropriate landmark node. In cases where finding an easily reachable landmark node is challenging, such as in road networks, their algorithm may yield poor performance, as observed in our experiments.

To overcome this problem, in this work, we propose a novel multiple landmarks technique based on several newly-developed effective resistance formulas. This technique allows us to strategically select a set of multiple landmark nodes, denoted by $\mathcal{V}_l$, which in turn enables faster random walk sampling. By leveraging the multiple landmarks technique, we develop a novel, efficient and provable index-based approach to efficiently compute both the single-pair and single-source effective resistance queries. More specifically, we first propose three new effective resistance formulas with multiple landmarks, using a classic concept of Schur complement [12]. Based on these new formulas, we can pre-compute and maintain several small-sized matrices, which are related the landmark node set $\mathcal{V}_l$, as a compact index. To construct the index, we develop two efficient and provable Monte Carlo algorithms based on several interesting and newly-discovered probability interpretations of the Schur complement. Armed with our indexing technique, we propose new and provable $\mathcal{V}_l$-absorbed random walk and $\mathcal{V}_l$-absorbed push algorithms to efficiently answer both the single-pair and single-source effective resistance queries. We show that the state-of-the-art algorithm based on a single landmark node is a special case of our algorithms. In addition, we also present comprehensive theoretical analysis of our algorithms, and the results demonstrate that all our algorithms can achieve an ϵ-absolute error guarantee while using only sub-linear time. Finally, we conduct extensive experiments using 5 large real-life graphs to evaluate our algorithms, and the results show that our algorithms are substantially faster than the state-of-the-art algorithms while maintaining the same accuracy level. To summarize, the main contributions of this paper are as follows.

New theoretical results. We propose three new formulas to compute the effective resistance with multiple landmarks $\mathcal{V}_l$. We propose novel and interesting probability interpretations of the Schur complement of a graph w.r.t. the landmark nodes set $\mathcal{V}_l$. We believe that these novel probability interpretations could be of independent interest. In addition, we also establish several novel connections among effective resistance and three new concepts called $\mathcal{V}_l$-absorbed random walk, $\mathcal{V}_l$-absorbed push, and $\mathcal{V}_l$-rooted random spanning forest. We present detailed theoretical analysis for all our algorithms and the results show that all of them can achieve provable accuracy guarantee and take sub-linear running time.

Novel index-based algorithms. We propose novel index-based approaches to both single-pair and single-source effective resistance computations. Specifically, we first propose several novel techniques to construct the index, including $\mathcal{V}_l$-absorbed random walk sampling, $\mathcal{V}_l$-rooted random spanning forest sampling, and loop-erased random walk sampling. Then, armed with our index, we develop two new $\mathcal{V}_l$-absorbed random walk and $\mathcal{V}_l$-absorbed push algorithms to efficiently answer both the single-pair and single-source effective resistance queries. We show that the time complexity of our query processing algorithms is strictly lower than the state-of-the-art (SOTA) algorithms.

Extensive experiments. We conduct comprehensive experiments on 5 large real-life graphs to evaluate our algorithms. The results show that by selecting a small set of landmark nodes, our algorithms can be up to 4 orders magnitude faster than the SOTA algorithms for both single-pair and single-source effective resistance computations, when achieve the same accuracy. For example, on the Road-PA dataset (1.09 million nodes, 1.54 millon edges), our algorithms takes 0.19 and 30 seconds to achieve a relative error 0.05 for single-pair and single-source queries respectively, while the SOTA algorithm consumes 184 and 2×10^4 seconds respectively. Additionally, the results also show that our index can be constructed very efficient and uses acceptable space. For instance, on Road-PA dataset, our index can be built within 788 seconds using 1493 MB spaces. For reproducible purpose, the source code of the paper can be found at https://anonymous.4open.science/r/ML-EED5.

2 PRELIMINARIES

Notations and definitions. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with $|\mathcal{V}| = n$ nodes and $|\mathcal{E}| = m$ edges. For a node u, denote by $\mathcal{N}(u)$ the set of neighbor nodes of u. Let $\mathbf{A}$ be the adjacency matrix of $\mathcal{G}$ and $\mathbf{D}$ be the diagonal degree matrix with $\mathbf{D}_{uu} = d(u) = |\mathcal{N}(u)|$. Let $\mathbf{L}$ be the Laplacian matrix of $\mathcal{G}$ defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$. Without loss of generality, we assume that the graph is connected (if the graph is disconnected, we can consider each connected component separately). For any n-nodes connected graph, the Laplacian matrix $\mathbf{L}$ has a rank $n-1$, thus the inverse of $\mathbf{L}$ does not exist. Let $0 = \mu_1 \leq \cdots \leq \mu_n$ be the eigenvalues of $\mathbf{L}$, and $\mathbf{u}_1, \cdots, \mathbf{u}_n$ be the corresponding eigenvectors. The eigen-decomposition of $\mathbf{L}$ is $\mathbf{L} = \sum_{i=2}^{n} \mu_i \mathbf{u}_i \mathbf{u}_i^T$, because $\mu_1 = 0$. Based on this, the classic Moore-Penrose pseudo-inverse of $\mathbf{L}$ can be defined as $\mathbf{L}^\dagger = \sum_{i=2}^{n} \frac{1}{\mu_i} \mathbf{u}_i \mathbf{u}_i^T$. Given a graph $\mathcal{G}$ and two nodes s, t, the effective resistance (ER) between s and t is defined as

$$r(s,t) = (\mathbf{L}^\dagger)_{ss} + (\mathbf{L}^\dagger)_{tt} - 2(\mathbf{L}^\dagger)_{st}. \tag{1}$$

Eq. (1) is often used for computing the ER. However, as shown in [12], the ER can also be computed by the so-called g-inverse of $\mathbf{L}$. Specifically, the g-inverse of $\mathbf{L}$, denoted by $\mathbf{H}$, is a symmetric matrix that satisfies $\mathbf{LHL} = \mathbf{L}$. Then, $r(s,t)$ can be determined by the following formula:

$$r(s,t) = (\mathbf{H})_{ss} + (\mathbf{H})_{tt} - 2(\mathbf{H})_{st}. \tag{2}$$

Notice that the g-inverse of $\mathbf{L}$ is not unique. Thus, we can compute effective resistance once we construct any g-inverse of $\mathbf{L}$. Random walk is a random process on graphs. In each step, a random surfer jumps to the neighbor of the current node u with probability $\frac{1}{d(u)}$.

Table 1: Frequently used notations.

Notation	Description
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	An undirected graph $\mathcal{G}$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$
$\mathbf{D}, \mathbf{A}, \mathbf{L}$	The degree matrix, adjacency matrix and Laplacian matrix of $\mathcal{G}$, respectively
$\mathbf{P}$	The probability transition matrix of $\mathcal{G}$
$\bar{d}, \Delta_{\mathcal{G}}$	The average degree and the diameter of $\mathcal{G}$
$\mathcal{V}_l, \mathcal{U}$	The landmark node set $\mathcal{V}_l$ and the remaining node set $\mathcal{U}$
$\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$ ($\mathbf{P}_{\mathcal{U}\mathcal{V}_l}$)	The sub-matrix of $\mathbf{L}$ ($\mathbf{P}$) with rows indexed by $\mathcal{U}$ and columns indexed by $\mathcal{V}_l$
$\mathbf{L}/\mathcal{V}_l, \mathbf{L}_{\mathcal{H}}$	The real and estimated Schur complement of the landmark node set $\mathcal{V}_l$
$\mathcal{G}/\mathcal{V}_l, \mathcal{H}$	The real and estimated Schur complement graph
$\mathcal{G}\backslash\mathcal{V}_l$	The remaining graph by regarding $\mathcal{V}_l$ as an extended node
$\mathbf{P}_r$ ($\mathbf{P}_f$)	The random walk (spanning forest) probability matrix
$\mathbf{P}_u$	The u-th row of the matrix $\mathbf{P}_r$ ($\mathbf{P}_f$)
λ	The spectral radius of the matrix $\mathbf{P}$
$\lambda_{\mathcal{V}_l}$	The spectral radius of the matrix $\mathbf{P}_{\mathcal{U}\mathcal{U}}$
ϵ	The absolute error threshold

The probability transition matrix is defined as $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$. The commute time $c(s,t)$ between two nodes s, t is the expected number of steps of the random walk starting from s, visiting t, and then coming back to s. It is well known that the commute time is closely related to ER, i.e., $c(s,t) = 2m \times r(s,t)$ [56]. The hitting time $h(s,t)$ between two nodes s, t is the expected number of steps of the random walk starting from s and visiting t for the first time. Similarly, the hitting time $h(s, \mathcal{V}_l)$ is the expected number of steps of the random walk starting from s and visiting a node set $\mathcal{V}_l$ for the first time.

Let $\mathbf{L}_v$ be a sub-matrix of $\mathbf{L}$ which is obtained by deleting the v-th row and the v-th column of $\mathbf{L}$. Unlike $\mathbf{L}$, the inverse of $\mathbf{L}_v$ exists for any node v. Recently, Liao et al. [31] shows that ER can also be computed based on the matrix $\mathbf{L}_v$:

$$r(s,t) = (\mathbf{L}_v^{-1})_{ss} + (\mathbf{L}_v^{-1})_{tt} - 2(\mathbf{L}_v^{-1})_{st}, \quad u_1, u_2 \neq v; \tag{3}$$

$$r(u,v) = (\mathbf{L}_v^{-1})_{uu}. \tag{4}$$

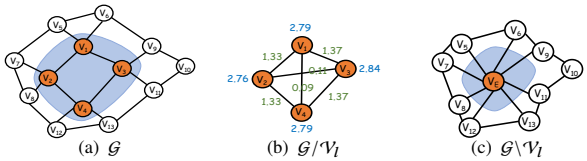
Note that both Eq. (3) and Eq. (4) always hold for any node v. This is because it is easy to verify that $\begin{bmatrix} \mathbf{L}_v^{-1} & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix}$ is a g-inverse of $\mathbf{L}$. Substituting the g-inverse into Eq. (2), we can obtain Eq. (3) and Eq. (4). Moreover, similar to the shortest path distance, effective resistance is also a distance metric [12]. Notably, given three distinct nodes s, t, v, the effective resistance satisfies $r(s,v) + r(v,t) > r(s,t)$. Based on Eq. (3) and Eq. (4), it is easy to derive that the difference is exactly $2(\mathbf{L}_v^{-1})_{st}$. Thus, to compute $r(s,t)$, the existing method [31] first compute the effective resistance distance from s, t to the landmark node v. Then, efficient approximate methods are designed to approximate the difference $2(\mathbf{L}_v^{-1})_{st}$. Table 1 lists the notations that are frequently used in this paper.

Problem formulation. Based on the definition of ER, we formulate two ER computation problems as follows. Note that we do not consider the problem of all-pairs ER computation, since it requires $O(n^2)$ space to store the results, which is clearly intractable for large graphs.

Definition 2.1 (Single-pair ER computation). Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and two nodes s, t with $s \neq t$, the single-pair ER computation problem is to calculate $r(s,t)$.

Definition 2.2 (Single-source ER computation). Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a source node s, the single-source ER computation problem is to calculate $r(s,t)$ for every $t \in \mathcal{V}$.

Note that by definition, computing the exact effective resistance requires to calculate the matrix inverse (or pseudo-inverse) which is typically intractable for large graphs. As a result, existing solutions often focus on deriving an ϵ-estimation of the effective resistance [31, 45, 67], i.e., outputting an approximate effective resistance $\hat{r}(s,t)$ that satisfies $|\hat{r}(s,t) - r(s,t)| \leq \epsilon$ with a high probability (e.g., the failure probability is smaller than 0.01). In this work, we also focus on developing efficient ϵ-estimation algorithms for both single-pair and single-source ER computation. Below, we first establish several new effective resistance formulas with multiple landmark nodes, based on which we will develop a powerful index-based approach to compute single-pair and single-source ER queries.



(a) $\mathcal{G}$ (b) $\mathcal{G}/\mathcal{V}_l$ (c) $\mathcal{G}\backslash\mathcal{V}_l$

Figure 1: An illustrative example of a graph and its Schur complement. (a) An example graph $\mathcal{G}$ with Laplacian matrix $\mathbf{L}$, a landmark node set $\mathcal{V}_l = \{v_1, v_2, v_3, v_4\}$ is chosen; (b) The Schur complement graph $\mathcal{G}/\mathcal{V}_l$ with Laplacian matrix $\mathbf{L}/\mathcal{V}_l = \mathbf{L}_{\mathcal{V}_l\mathcal{V}_l} - \mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$; (c) The remaining graph $\mathcal{G}\backslash\mathcal{V}_l$ which is obtained by regarding $\mathcal{V}_l$ as an extended node v_E, that $\mathbf{L}_{\mathcal{U}\mathcal{U}}$ is matrix obtained by removing the row and column indexed by v_E from the Laplacian matrix of $\mathcal{G}\backslash\mathcal{V}_l$.

3 NEW EFFECTIVE RESISTANCE FORMULAS

Recall that Liao et al. [31] proposed an effective resistance (ER) formula based on the matrix $\mathbf{L}_v$ that involves only one landmark node v. A natural question arises: can we extend their formula to handle scenarios with multiple landmark nodes? Note that solving this problem is quite nontrivial as it needs to incorporate the concept of Schur complement. Specifically, let $\mathcal{V}_l \subset \mathcal{V}$ be a small set of *landmark* nodes (e.g., $|\mathcal{V}_l| \leq 100$), and $\mathcal{U} = \mathcal{V} \setminus \mathcal{V}_l$ be the set of remaining nodes. Denote by $\mathbf{L}_{\mathcal{U}\mathcal{U}}$ the sub-matrix obtained by deleting the rows and columns indexed by the landmark node set $\mathcal{V}_l$. Then, $\mathbf{L}$ can be represented as $\mathbf{L} = \begin{bmatrix} \mathbf{L}_{\mathcal{U}\mathcal{U}} & \mathbf{L}_{\mathcal{U}\mathcal{V}_l} \\ \mathbf{L}_{\mathcal{V}_l\mathcal{U}} & \mathbf{L}_{\mathcal{V}_l\mathcal{V}_l} \end{bmatrix}$. Based on this block representation, we formally define the Schur complement of the landmark node set $\mathcal{V}_l$, denoted by $\mathbf{L}/\mathcal{V}_l$, as follows.

Definition 3.1. For a node set $\mathcal{V}_l$, the Schur complement of $\mathcal{V}_l$ is $\mathbf{L}/\mathcal{V}_l \triangleq \mathbf{L}_{\mathcal{V}_l\mathcal{V}_l} - \mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$.

New ER formulas. It is well-known that the Schur complement $\mathbf{L}/\mathcal{V}_l$ is also the Laplacian matrix of a weighted graph $\mathcal{G}/\mathcal{V}_l$ [14]. Here we give an illustrative example. As shown in Fig. 1(a), $\mathcal{G}$ is an example graph with 13 nodes. The Laplacian matrix of $\mathcal{G}$ is $\mathbf{L}$. By selecting a landmark node set $\mathcal{V}_l = \{v_1, v_2, v_3, v_4\}$, and suppose that $\mathcal{U}$ is the remaining node set, we can compute the Schur complement $\mathbf{L}/\mathcal{V}_l$ respectively. Then, $\mathbf{L}/\mathcal{V}_l$ is the Laplacian matrix of a weighted graph $\mathcal{G}/\mathcal{V}_l$ which is illustrated in Fig. 1(b), whose node set is exactly $\mathcal{V}_l$. By considering $\mathcal{V}_l$ as an extended node v_E, we also obtain a graph $\mathcal{G}\backslash\mathcal{V}_l$ (Fig. 1(c)), where $\mathbf{L}_{\mathcal{U}\mathcal{U}}$ is the matrix obtained by removing the row and column indexed by v_E from the Laplacian matrix of $\mathcal{G}\backslash\mathcal{V}_l$. Let $(\mathbf{L}/\mathcal{V}_l)^{\dagger}$ be the pseudo-inverse of $\mathbf{L}/\mathcal{V}_l$. Similar to $\mathbf{L}_v$, the inverse of $\mathbf{L}_{\mathcal{U}\mathcal{U}}$ also exists. Below, we construct a new

g-inverse of $\mathbf{L}$ with the pseudo-inverse of the Schur complement $\mathbf{L}/\mathcal{V}_l$.

LEMMA 3.2. *Given a landmark node set $\mathcal{V}_l$ and the remaining node set $\mathcal{U} = \mathcal{V} \setminus \mathcal{V}_l$, a g-inverse of $\mathbf{L}$, denoted by H, can be constructed as:*

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} & \mathbf{0} \\ \mathbf{0} & (\mathbf{L}/\mathcal{V}_l)^{\dagger} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} + \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}(\mathbf{L}/\mathcal{V}_l)^{\dagger}\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} & -\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}(\mathbf{L}/\mathcal{V}_l)^{\dagger} \\ -(\mathbf{L}/\mathcal{V}_l)^{\dagger}\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} & (\mathbf{L}/\mathcal{V}_l)^{\dagger} \end{bmatrix}. \tag{5}$$

PROOF. By definition, if a symmetric matrix $\mathbf{H}$ satisfies the property $\mathbf{LHL} = \mathbf{L}$, $\mathbf{H}$ is a g-inverse of $\mathbf{L}$. Clearly, H defined in Eq. (5) is symmetric. Since $\begin{bmatrix} \mathbf{I} & \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} = \mathbf{I}$, and $\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} & \mathbf{I} \end{bmatrix} = \mathbf{I}$, we have:

$$\begin{aligned} \mathbf{L}\mathbf{L}^{\dagger}\mathbf{L} &= \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{L}_{\mathcal{U}\mathcal{U}} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}/\mathcal{V}_l \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \\ &\quad \begin{bmatrix} \mathbf{I} & -\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} & \mathbf{0} \\ \mathbf{0} & (\mathbf{L}/\mathcal{V}_l)^{\dagger} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} & \mathbf{I} \end{bmatrix} \\ &\quad \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{L}_{\mathcal{U}\mathcal{U}} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}/\mathcal{V}_l \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{L}_{\mathcal{U}\mathcal{U}} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}/\mathcal{V}_l \end{bmatrix} \begin{bmatrix} \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} & \mathbf{0} \\ \mathbf{0} & (\mathbf{L}/\mathcal{V}_l)^{\dagger} \end{bmatrix} \\ &\quad \begin{bmatrix} \mathbf{L}_{\mathcal{U}\mathcal{U}} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}/\mathcal{V}_l \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{L}_{\mathcal{U}\mathcal{U}} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}/\mathcal{V}_l \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \\ &= \mathbf{L}, \end{aligned}$$

Thus, the Lemma is established. □

Based on Lemma 3.2 and Eq. (1), we present new formulas for ER computation as follows:

LEMMA 3.3. *(New effective resistance formulas) Given a landmark node set $\mathcal{V}_l$ and the remaining node set $\mathcal{U} = \mathcal{V} \setminus \mathcal{V}_l$, the effective resistance between any two nodes can be computed by the following formulas.*

(1) For $u_1, u_2 \in \mathcal{U}$, we have

$$\begin{aligned} r(u_1,u_2) = {} & (\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{u_1u_1} + (\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{u_2u_2} - 2(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{u_1u_2} \\ & + (\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}(\mathbf{L}/\mathcal{V}_l)^{\dagger}\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{u_1u_1} \\ & + (\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}(\mathbf{L}/\mathcal{V}_l)^{\dagger}\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{u_2u_2} \\ & - 2(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}(\mathbf{L}/\mathcal{V}_l)^{\dagger}\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{u_1u_2}; \end{aligned} \tag{6}$$

(2) For $u \in \mathcal{U}, v \in \mathcal{V}_l$, we have

$$\begin{aligned} r(u,v) = {} & (\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{uu} + ((\mathbf{L}/\mathcal{V}_l)^{\dagger})_{vv} \\ & + (\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}(\mathbf{L}/\mathcal{V}_l)^{\dagger}\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{uu} \\ & - 2(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}(\mathbf{L}/\mathcal{V}_l)^{\dagger})_{uv}; \end{aligned} \tag{7}$$

(3) For $v_1, v_2 \in \mathcal{V}_l$, we have

$$\begin{aligned} r(v_1,v_2) = {} & ((\mathbf{L}/\mathcal{V}_l)^{\dagger})_{v_1v_1} + ((\mathbf{L}/\mathcal{V}_l)^{\dagger})_{v_2v_2} \\ & - 2((\mathbf{L}/\mathcal{V}_l)^{\dagger})_{v_1v_2}. \end{aligned} \tag{8}$$

PROOF. Based on Lemma 3.2, since $r(s,t) = (H)_{ss} + (H)_{tt} - 2(H)_{st} = (\mathbf{e}_s - \mathbf{e}_t)^T H(\mathbf{e}_s - \mathbf{e}_t)$, the formulas can be derived in three cases: (1) For $u_1, u_2 \in \mathcal{U}$, we can derive $r(u_1,u_2) = (\mathbf{e}_{u_1} - \mathbf{e}_{u_2})^T\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}(\mathbf{e}_{u_1} - \mathbf{e}_{u_2}) + (\mathbf{e}_{u_1} - \mathbf{e}_{u_2})^T\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}(\mathbf{L}/\mathcal{V}_l)^{\dagger}\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}(\mathbf{e}_{u_1} - \mathbf{e}_{u_2})$ by Eq. (5). By expanding the bilinear form, we can derive Eq. (6); (2) For $u \in \mathcal{U}, v \in \mathcal{V}_l$, we have $r(u,v) = (\mathbf{e}_u - \mathbf{e}_v)^T H(\mathbf{e}_u - \mathbf{e}_v)$, by Eq. (5), we can derive Eq. (7); (3) For $v_1, v_2 \in \mathcal{V}_l$, we have $r(v_1,v_2) = (\mathbf{e}_{v_1} - \mathbf{e}_{v_2})^T(\mathbf{L}/\mathcal{V}_l)^{\dagger}(\mathbf{e}_{v_1} - \mathbf{e}_{v_2})$. By expanding the bilinear form, we can obtain Eq. (8). The proof is completed. □

By Lemma 3.3, to compute the effective resistance for any two nodes, it is sufficient to calculate (i) the matrix $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$ (note that it is no need to repeatedly compute $\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$, because $\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} = (\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l})^T$); (ii) the pseudo-inverse of the Schur complement: $(\mathbf{L}/\mathcal{V}_l)^{\dagger}$; and (iii) the inverse of the Laplacian submatrix: $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$. By Definition 3.1, the Schur complement $\mathbf{L}/\mathcal{V}_l$ is closely related to the matrix $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$. Note the Schur complement is a $|\mathcal{V}_l| \times |\mathcal{V}_l|$ matrix and $|\mathcal{V}_l|$ is often small (e.g., $|\mathcal{V}_l| \leq 100$), thus $(\mathbf{L}/\mathcal{V}_l)^{\dagger}$ can be easily computed after obtaining the matrix $\mathbf{L}/\mathcal{V}_l$. As a consequence, for both (i) and (ii), the key is to determine the matrix $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$. Below, we propose several interesting probability interpretations of the matrix $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$, based on which we will develop several novel and efficient sampling techniques to estimate such a matrix.

Novel probability interpretations of $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$. We first give a random walk interpretation for each element of the matrix $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$. For a given set of landmark nodes $\mathcal{V}_l$, we define a $\mathcal{V}_l$-absorbed random walk as a random walk that terminates once it reaches any node in the set $\mathcal{V}_l$. That is to say, for a $\mathcal{V}_l$-absorbed random walk starting from a node $u \in \mathcal{U}$, it must stop at a node $v \in \mathcal{V}_l$. Let $\tau_{\mathcal{V}_l}[u_1,u_2]$ be the expected number of visits to node u_2 in a $\mathcal{V}_l$-absorbed walk starting from node u_1. Denote by $\mathbf{P}_{\mathcal{U}\mathcal{U}}$ a sub-matrix of the probability transition matrix $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ obtained by removing the rows and columns indexed by the set $\mathcal{V}_l$. Then, by definition, we have $\tau_{\mathcal{V}_l}[u_1,u_2] = \sum_{k=0}^{\infty}(\mathbf{P}_{\mathcal{U}\mathcal{U}}^k)_{u_1u_2}$. Specifically, $(\mathbf{P}_{\mathcal{U}\mathcal{U}}^k)_{u_1u_2}$ is the probability that a $\mathcal{V}_l$-random walk starting from u_1 visits u_2 at the k-th step. Based on this, we have the following results.

LEMMA 3.4. $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{u_1u_2} = \frac{\tau_{\mathcal{V}_l}[u_1,u_2]}{d(u_2)}$.

PROOF. Let $\mathbf{D}_{\mathcal{U}\mathcal{U}}$ be a sub-matrix of $\mathbf{D}$ obtained by removing the rows and columns indexed by the set $\mathcal{V}_l$. Since $\mathbf{D}_{\mathcal{U}\mathcal{U}}(\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}}) = \mathbf{L}_{\mathcal{U}\mathcal{U}}$, we have $(\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\mathbf{D}_{\mathcal{U}\mathcal{U}}^{-1} = \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$. Note that $\tau_{\mathcal{V}_l}[u_1,u_2] = \sum_{k=0}^{\infty}(\mathbf{P}_{\mathcal{U}\mathcal{U}}^k)_{u_1u_2} = (\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})_{u_1u_2}^{-1}$. Thus, the lemma is established. □

Then, by Lemma 3.4, we can derive an interesting probability interpretation of the matrix $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$ as follows.

LEMMA 3.5. *($\mathcal{V}_l$-absorbed random walk interpretation) Given a node $u \in \mathcal{U}$ and a node $v \in \mathcal{V}_l$, the u,v-th element of the matrix $-(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l})_{uv}$ is the probability p_{uv}^r that a $\mathcal{V}_l$-absorbed random walk starts from u and terminates at the node $v \in \mathcal{V}_l$.*

PROOF. Recall that $\tau_{\mathcal{V}_l}[u,w] = \sum_{k=0}^{\infty}(\mathbf{P}_{\mathcal{U}\mathcal{U}}^k)_{uw}$, where $(\mathbf{P}_{\mathcal{U}\mathcal{U}}^k)_{uw}$ is the probability that a $\mathcal{V}_l$-random walk starting from u visits w at the k-th step. If $w \in \mathcal{N}(v)$, then with probability $\frac{1}{d(w)}$, the random

walk will reach $v \in \mathcal{V}_l$ in the next step (i.e., it will terminates at v). Thus, by adding up all such probabilities, we have

$$p^r_{uv} = \sum_{w \in \mathcal{N}(v) \cap \mathcal{U}} \frac{\sum_{k=0}^{\infty} (\mathbf{P}^k_{\mathcal{U}\mathcal{U}})_{uw}}{d_w} = \sum_{w \in \mathcal{N}(v) \cap \mathcal{U}} \frac{\tau_{\mathcal{V}_l}[u,w]}{d_w}$$
$$= \sum_{w \in \mathcal{N}(v) \cap \mathcal{U}} (\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uw}.$$

The last equality is due to Lemma 3.4. Since $\mathbf{D}_{\mathcal{U}\mathcal{V}_l}$ is a zero matrix by definition, we have $\mathbf{L}_{\mathcal{U}\mathcal{V}_l} = -\mathbf{A}_{\mathcal{U}\mathcal{V}_l}$. Therefore, $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{V}_l} = -\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{A}_{\mathcal{U}\mathcal{V}_l}$. As a consequence, $p^r_{uv} = \sum_{w \in \mathcal{N}(v) \cap \mathcal{U}} (\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uw} = \mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{A}_{\mathcal{U}\mathcal{V}_l} = -(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{V}_l})_{uv}$. This completes the proof. □

Lemma 3.5 indicates that the matrix $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$ can be efficiently estimated by using $\mathcal{V}_l$-random walk sampling technique. Below, we further propose a different probability interpretation for $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$ based on a concept of random spanning forest.

A spanning forest is a subgraph of $\mathcal{G}$ with no cycles. For each connected component of a spanning forest, we specify a node as the root node in a rooted spanning forest. The set of all root nodes is called the root set $\mathcal{R}$. For convenience, we let the root set be the landmark node set $\mathcal{V}_l$, i.e., $\mathcal{R} = \mathcal{V}_l$. For a tree $\mathcal{T}$ with a root node v, we call that "u is rooted at v" for each node $u \in \mathcal{T}$. A random spanning forest with a prescribed root set $\mathcal{V}_l$ is a spanning forest uniformly sampled from all spanning forests of $\mathcal{G}$ with the root set $\mathcal{V}_l$. With this concepts, we can derive the following results.

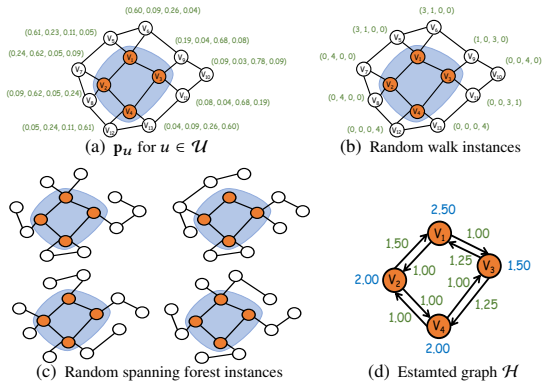
LEMMA 3.6. *($\mathcal{V}_l$-rooted random spanning forest interpretation) Given a node $u \in \mathcal{U}$ and a node $v \in \mathcal{V}_l$, the u,v-th element $-(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{V}_l})_{uv}$ is the probability p^f_{uv} that in a random spanning forest $\mathcal{F}$ with root set $\mathcal{V}_l$, u is rooted at v in a tree of $\mathcal{F}$.*

PROOF. According to the all-minors matrix forest theorem [17], we have $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uw} = \frac{|\mathcal{F}_{\mathcal{V}_l|u,w}|}{|\mathcal{F}_{\mathcal{V}_l}|}$, where $\mathcal{F}_{\mathcal{V}_l}$ is the set of spanning forests with root set $\mathcal{V}_l$, $\mathcal{F}_{\mathcal{V}_l|u,w}$ is the set of spanning forests with root set $\mathcal{V}_l \cup \{u\}$, and w belongs to the tree component with root u. Let $\mathcal{F}_{\mathcal{V}_l:u\rightsquigarrow v}$ be the set of spanning forests with root set $\mathcal{V}_l$, u is rooted at v. There is a bijection between $\mathcal{F}_{\mathcal{V}_l:u\rightsquigarrow v}$ and $\bigcup_{w \in \mathcal{N}(v) \cap \mathcal{U}} \mathcal{F}_{\mathcal{V}_l|u,w}$. To see this, if we remove the last edge on the path from u to $\mathcal{V}_l$ in each spanning forest in $\mathcal{F}_{\mathcal{V}_l:u\rightsquigarrow v}$, we will exactly obtain a spanning forest in $\bigcup_{w \in \mathcal{N}(v) \cap \mathcal{U}} \mathcal{F}_{\mathcal{V}_l|u,w}$. And it is easy to verify that the reverse is also true. Thus, we have:

$$p^f_{uv} = -\frac{\sum_{w \in \mathcal{N}(v) \cap \mathcal{U}} |\mathcal{F}_{\mathcal{V}_l|u,w}|}{|\mathcal{F}_{\mathcal{V}_l}|} = \sum_{w \in \mathcal{N}(v) \cap \mathcal{U}} (\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uw}$$
$$= -(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{V}_l})_{uv}.$$

As a result, the element $-(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{V}_l})_{uv}$ is exactly the probability that in a random spanning forest with root set $\mathcal{V}_l$, the node u is rooted at v. □

Note that each row of the matrix $-\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$ sums up to 1, this is because a random walk starts from a node $u \in \mathcal{U}$ must stop when hitting a node $v \in \mathcal{V}_l$, thus the probability $\sum_{v \in \mathcal{V}_l} p^r_{uv} = 1$. Similarly, in a random spanning forest with root set $\mathcal{V}_l$, each node $u \in \mathcal{U}$ must be root at a certain landmark node $v \in \mathcal{V}_l$, the probability $\sum_{v \in \mathcal{V}_l} p^f_{uv} = 1$. We use $\mathbf{P}_r \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{V}_l|}$ to denote the random walk probability matrix where $(\mathbf{P}_r \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{V}_l|})_{uv} = p^r_{uv}$, and $\mathbf{P}_f \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{V}_l|}$ to denote the random spanning forest probability matrix where $(\mathbf{P}_f \in \mathbb{R}^{|\mathcal{U}| \times |\mathcal{V}_l|})_{uv} = p^f_{uv}$. Therefore, we have $\mathbf{P}_r = \mathbf{P}_f = -\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$ by Lemma 3.5 and Lemma 3.6. Then, we can further obtain three new ER formulas based on the matrices $\mathbf{P}_r$ and $\mathbf{P}_f$.



(a) $\mathbf{p}_u$ for $u \in \mathcal{U}$ (b) Random walk instances (c) Random spanning forest instances (d) Estamted graph $\mathcal{H}$

Figure 2: Illustrative of several important concepts. (a) The probability vector $\mathbf{p}_u$ for $u \in \mathcal{U}$. (b) 4 possible random walk instances. Suppose that we sample 4 random walks, the $[3,1,0,0]^T$ vector depicted near v_5 means that 3 (1, 0, 0) of them hits $\mathcal{V}_l$ by v_1 (v_2, v_3, v_4), respectively. (c) 4 possible random spanning forests instances. (d) Based on the random walk and random spanning forests instances, we can obtain the same estimated graph $\mathcal{H}$ according to Lemma 3.8.

LEMMA 3.7. *Let $\mathbf{p}_u$ be the u-th row of the matrix $\mathbf{P}_r$ ($\mathbf{P}_f$) for $u \in \mathcal{U}$, where $\mathbf{p}_u(v)$ is the probability that a random walk from u hits $v \in \mathcal{V}_l$ (the probability that in a random spanning forest with root set $\mathcal{V}_l$, u is rooted at v). Let $\mathbf{e}_u$ be a one-hot vector such that the element indexed by u is 1 and other elements are 0. Then, we have:*

(1) For $u_1, u_2 \in \mathcal{U}$, we have

$$r(u_1,u_2) = (\mathbf{e}_{u_1} - \mathbf{e}_{u_2})^T(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})(\mathbf{e}_{u_1} - \mathbf{e}_{u_2}) + (\mathbf{p}_{u_1} - \mathbf{p}_{u_2})^T(\mathbf{L}/\mathcal{V}_l)^{\dagger}(\mathbf{p}_{u_1} - \mathbf{p}_{u_2}); \quad (9)$$

(2) For $u \in \mathcal{U}, v \in \mathcal{V}_l$, we have

$$r(u,v) = \mathbf{e}_u^T\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{e}_u + (\mathbf{p}_u - \mathbf{e}_v)^T(\mathbf{L}/\mathcal{V}_l)^{\dagger}(\mathbf{p}_u - \mathbf{e}_v); \quad (10)$$

(3) For $v_1, v_2 \in \mathcal{V}_l$, we have

$$r(v_1,v_2) = (\mathbf{e}_{v_1} - \mathbf{e}_{v_2})^T(\mathbf{L}/\mathcal{V}_l)^{\dagger}(\mathbf{e}_{v_1} - \mathbf{e}_{v_2}). \quad (11)$$

PROOF. Based on Lemma 3.3 and Lemma 3.5, Lemma 3.6, the u-th row of $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$ is a probability vector, which represents the probability that a $\mathcal{V}_l$-absorbed walk starts from u and hits the node $v \in \mathcal{V}_l$ (the probability that in a random spanning forest with root set $\mathcal{V}_l$, u is rooted at v). We also have $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{V}_l})^T = \mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}$. Thus, by substituting the matrix $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$ by $\mathbf{P}_r$ ($\mathbf{P}_f$) and combining the terms into bilinear forms, the proof is established. □

Thus, to compute the effective resistance, suppose that $v_1, v_2 \in \mathcal{V}_l$, $r(v_1,v_2)$ is exactly the effective resistance between v_1 and v_2 in the Schur complement graph $\mathcal{G}/\mathcal{V}_l$ according to Eq. (11). When $u \in \mathcal{U}$, $v \in \mathcal{V}_l$, $r(u,v)$ is the effective resistance between u and the

extended node v_E in $\mathcal{G}\backslash\mathcal{V}_l$, added by a linear combination of the effective resistances in $\mathcal{G}/\mathcal{V}_l$ according to Eq. (10). When $u_1, u_2 \in \mathcal{U}$, $r(u_1, u_2)$ is the effective resistance between u_1 and u_2 in $\mathcal{G}\backslash\mathcal{V}_l$, added by a linear combination of the effective resistances in $\mathcal{G}/\mathcal{V}_l$ according to Eq. (9).

A running example. Fig. 2(a) illustrates the probability vector $\mathbf{p}_u$ for $u \in \mathcal{U}$. For example, $\mathbf{p}_{v_5} = [0.61, 0.23, 0.11, 0.05]^T$ means that a $\mathcal{V}_l$-absorbed random walk from v_5 has probability 0.61 (0.23, 0.11, 0.05) to hit $\mathcal{V}_l$ by v_1 (v_2, v_3, v_4). Also, in a uniformly sampled random spanning forest with root set $\mathcal{V}_l$, the probability that v_5 is rooted at v_1 (v_2, v_3, v_4) is 0.61 (0.23, 0.11, 0.05). This is intuitive because v_1 is nearer to v_5 than other landmark nodes.

Fig. 2(a) and Fig. 2(b) illustrate the random walk and random spanning forests sampling techniques described above. Specifically, when sampling 4 random walks from each node $u \in \mathcal{U}$, suppose that the random walk starts from v_5 hits $\mathcal{V}_l$ by v_1 (v_2, v_3, v_4) for 3 (1, 0, 0) times (Fig. 2(b)), then $[0.75, 0.25, 0, 0]^T$ is an approximate of $\mathbf{p}_{v_5}$. Similarly, in the 4 random spanning forests sampled in Fig. 2(b), in 3 (1, 0, 0) of them v_5 is rooted at v_1 (v_2, v_3, v_4). Thus, we can also obtain an approximate $\mathbf{p}_{v_5}$ ($[0.75, 0.25, 0, 0]^T$) from the random spanning forest instances.

Novel probability interpretations of the Schur complement $\mathbf{L}/\mathcal{V}_l$. Based on Lemma 3.5 and Lemma 3.6, we can derive probability interpretations for the Schur complement $\mathbf{L}/\mathcal{V}_l = \mathbf{L}_{\mathcal{V}_l\mathcal{V}_l} - \mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$. Note that $\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$ is a $|\mathcal{V}_l| \times |\mathcal{V}_l|$ matrix. Then, we have $(\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l})_{v_iv_j} = -(\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{P}_r)_{v_iv_j} = \sum_{u\in\mathcal{U},\, u\in\mathcal{N}(v_i)} -p^r_{uv_j}$ for any $v_i, v_j \in \mathcal{V}_l$. In other words, the v_i, v_j-th element of the matrix $-\mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$ is the sum of the probability of a $\mathcal{V}_l$-absorbed random walk starting from u with $u \in \mathcal{U}$ and $u \in \mathcal{N}(v_i)$ that hits the landmark node $v_j \in \mathcal{V}_l$. Since $\sum_{v\in\mathcal{V}_l} p^r_{uv} = 1$ for $u \in \mathcal{U}$, we can derive that the v_i-th row of the matrix sums up to $-d_{\mathcal{V}_l}(v_i)$, where $d_{\mathcal{V}_l}(u) = d(u) - d_{\mathcal{U}}(u)$ denotes the number of nodes u that satisfy $u \in \mathcal{V}_l$ and $u \in \mathcal{N}(v_i)$, and $d_{\mathcal{U}}(u)$ denotes the number of nodes u that meet $u \in \mathcal{U}$ and $u \in \mathcal{N}(v_i)$. The Schur complement $\mathbf{L}/\mathcal{V}_l$ is also the Laplacian matrix of a Schur complement graph, which we denote by $\mathcal{G}/\mathcal{V}_l$. Similarly, the same statement also holds for the spanning forest probability p^r_{uv}. Thus, together with the matrix $\mathbf{L}_{\mathcal{V}_l\mathcal{V}_l}$, we can derive probability interpretations of the Schur complement $\mathbf{L}/\mathcal{V}_l$.

LEMMA 3.8. *(Probability interpretations of Schur complement) The Schur complement $\mathbf{L}/\mathcal{V}_l$ is the Laplacian matrix of a weighted graph $\mathcal{G}/\mathcal{V}_l$ with node set $\mathcal{V}_l$, the degree of each node $v \in \mathcal{V}_l$ is $d(v) - \sum_{u\in\mathcal{N}(v)\cap\mathcal{U}} p^r_{uv}$, the weight of each edge (v_i, v_j) for $v_i, v_j \in \mathcal{V}_l$ is $\mathbf{1}\{v_i \sim v_j\} + \sum_{u\in\mathcal{N}(v_i)\cap\mathcal{U}} p^r_{uv_j}$, where $\mathbf{1}\{v_i \sim v_j\}$ is an indicator variable that equals 1 when v_i and v_j are connected in the original graph, equals 0 otherwise. The statement still holds when we replace p^r_{uv} with p^f_{uv}.*

PROOF. Since $\mathbf{L}/\mathcal{V}_l = \mathbf{L}_{\mathcal{V}_l\mathcal{V}_l} - \mathbf{L}_{\mathcal{V}_l\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$, based on the interpretations presented in Lemma 3.5 and Lemma 3.6, the Lemma can be established. □

Note that Lemma 3.8 shows that we can easily construct the Schur complement $\mathbf{L}/\mathcal{V}_l$ based on the probability matrix $\mathbf{P}_r$ (or $\mathbf{P}_f$). In the following section, we will apply such probability interpretations to design an efficient sampling algorithm to compute the Schur complement. For example, based on the random walk and random spanning forest samples illustrated in Fig. 2(b) and Fig. 2(c), we can obtain a weighted, directed graph $\mathcal{H}$ as an estimated graph of $\mathcal{G}/\mathcal{V}_l$ according to Lemma 3.8, as illustrated in Fig. 2(d).

Discussions. If there is only one landmark node v in $\mathcal{V}_l$, then since for each random walk that hits $\mathcal{V}_l$, it must hit v, the probability $p^r_{uv} = 1$. According to Lemma 3.8, the Schur complement $\mathbf{L}/\mathcal{V}_l$ is 0. Therefore, the pseudo-inverse $(\mathbf{L}/\mathcal{V}_l)^\dagger$ is also 0. As a result, Eq. (5) degrades to:

$$\mathbf{H} = \begin{bmatrix} \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix}. \tag{12}$$

Then, the ER formula Eq. (3) and Eq. (4) can be recovered easily by Eq. (12), indicating that the result established in [31] is a special case our results.

4 SINGLE-PAIR ER COMPUTATION

In this section, we propose a novel index-based approach to compute single-pair ER based on the theoretical results established in Section 3. Recall that the key to compute the ER is to compute three matrices: (i) $\mathbf{P}_r$ (or $\mathbf{P}_f$), (ii) $(\mathbf{L}/\mathcal{V}_l)^\dagger$, and (iii) $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$. The basic idea of our approach is as follows. First, we pre-compute both $\mathbf{P}_r$ (or $\mathbf{P}_f$) and $(\mathbf{L}/\mathcal{V}_l)^\dagger$ and maintain these two matrices as an index. Clearly, such an index consumes $O(|\mathcal{V}_l|n)$ space. Since $|\mathcal{V}_l|$ is often a small constant, the index only requires a small amount of additional space. Second, armed with the index, we can efficiently process any single-pair ER query (e.g., $r(s, t)$) by computing at most three elements of $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$ (e.g., $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{ss}$, $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{tt}$, and $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{st}$) based on Lemma 3.7. Below, we first propose two efficient index construction algorithms, followed by the query processing algorithms.

4.1 Index Construction Algorithms

Our index includes two matrices $\mathbf{P}_r$ (or $\mathbf{P}_f$) and $(\mathbf{L}/\mathcal{V}_l)^\dagger$. To compute the matrix $\mathbf{P}_r$ ($\mathbf{P}_f$), it is identical to calculate $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}$ by Lemma 3.5 and Lemma 3.6. Let $\mathbf{p}_{v_i}$ be the v_i-th column of the matrix $\mathbf{P}_r$, in which an element denotes the probability that a $\mathcal{V}_l$-absorbed random walk starting from a node $u_i \in \mathcal{U}$ hits $v_i \in \mathcal{V}_l$. We can compute $\mathbf{p}_{v_i}$ by solving a linear system $\mathbf{L}_{\mathcal{U}\mathcal{U}}\mathbf{p}_{v_i} = \mathbf{L}_{\mathcal{U}\mathcal{V}_l}\mathbf{e}_{v_i}$, where $\mathbf{e}_{v_i}$ is a $|\mathcal{V}_l|$-dimensional one-hot vector such that the element indexed by v_i is 1 and other elements are 0. Thus, a basic method to compute $\mathbf{P}_r$ ($\mathbf{P}_f$) requires solving $|\mathcal{V}_l|$ linear systems. After determining $\mathbf{P}_r$, we can construct the Schur complement $\mathbf{L}/\mathcal{V}_l$ by Definition 3.1, and then compute its pseduo-inverse by eigen-decomposition. Clearly, such a basic method is costly for large graphs, because it needs to solve $|\mathcal{V}_l|$ linear systems. Below, we propose two new and efficient Monte Carlo approaches to estimate the probability matrix $\mathbf{P}_r$ ($\mathbf{P}_f$).

Index building by $\mathcal{V}_l$-absorbed random walk sampling. Based on the $\mathcal{V}_l$-absorbed random walk interpretation (Lemma 3.5), we can construct the index matrices $\mathbf{P}_r$ and $(\mathbf{L}/\mathcal{V}_l)^\dagger$ by $\mathcal{V}_l$-absorbed random walk sampling. The detailed algorithm is outlined in Algorithm 1. Specifically, Algorithm 1 samples a number of $\mathcal{V}_l$-absorbed random walk from each node $u \in \mathcal{U}$, and uses the proportion of walks that hit $v \in \mathcal{V}_l$ as an estimation of $(\mathbf{P}_r)_{uv}$ (Lines 4-6). Clearly, such an estimation is unbiased by Lemma 3.5. Similarly, by Lemma 3.8, Algorithm 1 constructs an unbiased estimator for each element of the Schur complement $\mathbf{L}/\mathcal{V}_l$ (Lines 7-11), resulting in an estimated

Laplacian matrix $\mathbf{L}_{\mathcal{H}}$. After that, the algorithm computes the pseudo-inverse of the matrix $\mathbf{L}_{\mathcal{H}}$ as the final estimation of $(\mathbf{L}/\mathcal{V}_l)^{\dagger}$. Note that although $\mathbf{L}_{\mathcal{H}}$ is an unbiased estimator of $\mathbf{L}/\mathcal{V}_l$, $\mathbf{L}_{\mathcal{H}}^{\dagger}$ is not necessarily an unbiased estimator of $(\mathbf{L}/\mathcal{V}_l)^{\dagger}$. However, this does not affect the theoretical guarantee of our algorithm as analyzed below.

Theoretical analysis of Algorithm 1. To analyze the accuracy and complexity of our algorithm, we need an additional concept, called ϵ-spectral sparsifier, which is defined as follows:

Definition 4.1. (ϵ-spectral sparsifier) Given two undirected, unweighted graphs $\mathcal{G}$, $\mathcal{H}$ with the same number of nodes, let $\mathbf{L}_{\mathcal{H}}$ and $\mathbf{L}_{\mathcal{G}}$ be the Laplacian matrix of $\mathcal{G}$ and $\mathcal{H}$. We call $\mathcal{H}$ an ϵ-spectral sparsifier if for any vector $\mathbf{x}$, $(1-\epsilon)\mathbf{x}^T\mathbf{L}_{\mathcal{G}}\mathbf{x} \leq \mathbf{x}^T\mathbf{L}_{\mathcal{H}}\mathbf{x} \leq (1+\epsilon)\mathbf{x}^T\mathbf{L}_{\mathcal{G}}\mathbf{x}$.

We first analyze the sample size needed for Algorithm 1 to construct $\mathcal{H}$ as an ϵ-spectral sparsifier of the Schur complement graph $\mathcal{G}/\mathcal{V}_l$, i.e. the matrix $\mathbf{L}_{\mathcal{H}}$ satisfies $(1-\epsilon)\mathbf{x}^T\mathbf{L}_{\mathcal{G}}\mathbf{x} \leq \mathbf{x}^T\mathbf{L}_{\mathcal{H}}\mathbf{x} \leq (1+\epsilon)\mathbf{x}^T\mathbf{L}_{\mathcal{G}}\mathbf{x}$. We need a variant of matrix Chernoff bound presented in [54].

THEOREM 4.2. *[54] Let $\mathcal{H}_1^{(m)}, \cdots, \mathcal{H}_{\omega}^{(m)}$ be random multi-graphs consisting random multi-edges (a multi-edge can appear in a multi-graph more than one time) over the graph $\mathcal{G}$. Let $\mathcal{H}_1, \cdots, \mathcal{H}_{\omega}$ be the weighted graphs that the weight of an edge (e_1, e_2), $w_{\mathcal{H}}$ is the sum of the weight of all multi-edges (e_1, e_2). Suppose that the following properties is satisfied: (i) The expectation sums to the graph $\mathcal{G}$, i.e. $\sum_{i=1}^{\omega} E[\mathcal{H}_i] = \mathcal{G}$; (ii) For each $\mathcal{H}_i$, any edge $e = (e_1, e_2) \in \mathcal{H}_i$ has low effective resistance in $\mathcal{G}$, i.e. $w_{\mathcal{H}}(e) r_{\mathcal{G}}(e_1, e_2) \leq \frac{\epsilon^2}{O(\log n)}$. Then, with high probability, $\mathcal{H} = \sum_{i=1}^{\omega} \mathcal{H}_i$ is an ϵ-spectral sparsifier of $\mathcal{G}$.*

Algorithm 1 constructs the approximated Schur complement graph $\mathcal{H}$ by adding edges to the induced graph with node set $\mathcal{V}_l$ when the $\mathcal{V}_l$-absorbed random walk sampled from a node $u \in \mathcal{U}$ and u is a neighbor of a node $v \in \mathcal{V}_l$. According to Lemma 3.8, $\mathcal{H}$ is an unbiased estimator of $\mathcal{G}/\mathcal{V}_l$. Thus, it remains to prove that each random edge added has low effective resistance in the graph $\mathcal{G}/\mathcal{V}_l$ that we want to approximate. We have the following Lemma:

LEMMA 4.3. *Let $\Delta_{\mathcal{G}}$ be the diameter of the graph $\mathcal{G}$. When the sample size ω is larger than $O(\frac{\Delta_{\mathcal{G}}|\mathcal{V}_l|\log n}{\epsilon^2})$, the approximated Schur complement graph $\mathcal{H}$ in Algorithm 1 is an ϵ-spectral sparsifier of $\mathcal{G}/\mathcal{V}_l$.*

PROOF. Let $\mathcal{H}_i$ denote the random multi-graph (i.e. graph consisting several random edges that can be repeated) including the edges as follows: in the i-th round of random walks from all $u \in \mathcal{U}$, for any $v_1, v_2 \in \mathcal{V}_l$, suppose that the random walk starts from the node $u \in \mathcal{N}(v_i) \cap \mathcal{U}$ hits $\mathcal{V}_l$ by v_2, then there is an edge (v_1, v_2) in $\mathcal{H}_i$ with weight $\frac{1}{\omega}$. The rounds of random walks are independent of each other. Since the weight of the edge (v_i, v_j) for $v_i, v_j \in \mathcal{V}_l$ in the Schur complement $\mathcal{G}/\mathcal{V}_l$ is $\sum_{u \in \mathcal{N}(v_i) \cap \mathcal{U}} p^r_{uv_j}$, suppose that $\mathcal{H} = \sum_{i=1}^{\omega} \mathcal{H}_i$, we have $E[\mathcal{H}] = \mathcal{G}/\mathcal{V}_l$ according to Lemma 3.8. The condition (i) in Theorem 4.2 is satisfied. Moreover, for an arbitrary node u, it is the neighbor of at most $|\mathcal{V}_l|$ nodes $v_i \in \mathcal{V}_l$. Thus, the weight of an edge (when summing up all the repeated multi-edges) in a random subgraph that u contributes is at most $\frac{|\mathcal{V}_l|}{\omega}$, when all the nodes in $\mathcal{V}_l$ have the neighbor u. As a result, for each edge in the approximate graph $\mathcal{H}_i$, the weight of the edge $w_{\mathcal{H}}(e)$ is at most $\frac{|\mathcal{V}_l|}{\omega}$, we have $w_{\mathcal{H}}(e) r_{\mathcal{G}}(e_1, e_2) \leq \frac{|\mathcal{V}_l|\Delta_{\mathcal{G}}}{\omega}$, since Δ_G is a natural bound of the effective resistance in $\mathcal{G}$ [12]. Thus, when $\omega \geq O(\frac{|\mathcal{V}_l|\Delta_G \log n}{\epsilon^2})$, the condition (ii) in Theorem 4.2 is satisfied. According to Theorem 4.2, $\mathcal{H}$ is an ϵ-spectral sparsifier of $\mathcal{G}$. □

Algorithm 1: RwIndex

Input: Graph $\mathcal{G}$, landmark node set $\mathcal{V}_l$, sample size ω
Output: The estimated index matrices $\widetilde{\mathbf{P}}_r$ and $\mathbf{L}_{\mathcal{H}}^{\dagger}$

$\widetilde{\mathbf{P}}_r \leftarrow \mathbf{0}$; $\mathcal{U} \leftarrow \mathcal{V} \backslash \mathcal{V}_l$;
Let $\mathcal{H}$ be a subgraph of $\mathcal{G}$ induced by the node set $\mathcal{V}_l$;
For each edge $(u, v) \in \mathcal{H}$, let $w(u, v) = 1$ be the weight of (u, v);
for $i = 1 : \omega$ **do**
  for *each node* $u \in \mathcal{U}$ **do**
    Sample a $\mathcal{V}_l$-absorbed random walk from u; Suppose the terminated node is $v \in \mathcal{V}_l$, then $(\widetilde{\mathbf{P}}_r)_{uv} \leftarrow (\widetilde{\mathbf{P}}_r)_{uv} + \frac{1}{\omega}$;
  for *each node* $v \in \mathcal{V}_l$ **do**
    for *each neighbor node* $u \in \mathcal{N}(v) \cap \mathcal{U}$ **do**
      if *the $\mathcal{V}_l$-absorbed random walk sampled from u terminates at v* **then**
        If the edge $(u, v) \in \mathcal{H}$, $w(u, v) \leftarrow w(u, v) + \frac{1}{\omega}$; Otherwise, create a new edge (u, v) with weight $w(u, v) \leftarrow \frac{1}{\omega}$;
Let $\mathbf{L}_{\mathcal{H}}$ be the Laplacian matrix of the weighted graph $\mathcal{H}$;
Compute the pseudo-inverse $\mathbf{L}_{\mathcal{H}}^{\dagger}$ by eigen-decomposition;
return $\widetilde{\mathbf{P}}_r, \mathbf{L}_{\mathcal{H}}^{\dagger}$

Based on Lemma 4.3, when the sample size $\omega \geq O(\frac{\Delta_{\mathcal{G}}|\mathcal{V}_l|\log n}{\epsilon^2})$, $\mathcal{H}$ is an ϵ-spectral sparsifier of $\mathcal{G}/\mathcal{V}_l$. We will show later in Section 4.2 that the computed $\mathbf{P}_r$ and $\mathcal{L}_{\mathcal{H}}^{\dagger}$ is sufficient to support an ϵ-estimation single-pair query efficiently if $\mathcal{H}$ is an ϵ-spectral sparsifier of $\mathcal{G}/\mathcal{V}_l$. Thus, we can obtain the overall time complexity of Algorithm 1.

LEMMA 4.4. *The expected time complexity of Algorithm 1 to obtain an ϵ-spectral sparsifier of $\mathcal{G}/\mathcal{V}_l$ is $O(\frac{\vec{1}^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}\Delta_{\mathcal{G}}|\mathcal{V}_l|\log n}{\epsilon^2})$.*

PROOF. Since each sample of random walks from all nodes $u \in \mathcal{U}$ in Algorithm 1 costs expected time $\sum_{u \in \mathcal{U}} h(u, \mathcal{V}_l) = \vec{1}^T(\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}$. The equation holds because $h(u, \mathcal{V}_l)$, the hitting time from node u to the set $\mathcal{V}_l$ can be written as $\mathbf{e}_u^T(\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}$ [56], the expected running time for sampling $\mathcal{V}_l$-absorbed random walks separately from $u \in \mathcal{U}$ is the sum of all such hitting times, which is $\vec{1}^T(\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}$. According to Lemma 4.3, to obtain an ϵ-spectral sparsifier of $\mathcal{G}/\mathcal{V}_l$, the sample size should be larger than $O(\frac{\Delta_{\mathcal{G}}|\mathcal{V}_l|\log n}{\epsilon^2})$. Thus, the expected total time complexity is $\omega \sum_{u \in \mathcal{U}} h(u, \mathcal{V}_l) = O(\frac{\vec{1}^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}\Delta_{\mathcal{G}}|\mathcal{V}_l|\log n}{\epsilon^2})$, which establishes the Theorem. □

In Lemma 4.4, $\vec{1}^T(\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}$ is the expected running time of random walks from $\mathcal{U}$ to hit $\mathcal{V}_l$. This quantity is not very large when $\mathcal{V}_l$ is properly selected. For example, on a real-life road network PowerGrid [27], $\vec{1}^T(\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}$ is $71 \times n$ when choosing the 10 highest degree nodes as $\mathcal{V}_l$. In practice, $|\mathcal{V}_l|$ is often selected as a small number (i.e., $|\mathcal{V}_l| \leq 100$). The diameter of the graph $\mathcal{G}$ is also often not very large in real-world graphs. Thus, Algorithm 1 is efficient in practice.

Index building by $\mathcal{V}_l$-rooted random spanning forest sampling. Likewise, based on the $\mathcal{V}_l$-rooted random spanning forest interpretation (Lemma 3.6), we can derive the index matrices $\mathbf{P}_f$ and $(\mathbf{L}/\mathcal{V}_l)^\dagger$ by $\mathcal{V}_l$-rooted random spanning forest sampling. The detailed implementation is shown in Algorithm 2. First, Algorithm 2 uniformly draws a set of $\mathcal{V}_l$-rooted random spanning forests using the classic Wilson algorithm [65] (Lines 4-5). Note that here we need to slightly modify the original Wilson algorithm to obtain the $\mathcal{V}_l$-rooted random spanning forests. The Wilson algorithm is based on the concept of loop-erased random walk (LERW), in which all loops in the random walk trajectory are removed. Specifically, the Wilson algorithm first fixes a node ordering. Then, the algorithm initializes a tree $\mathcal{T} = \{v\}$ with a root node v and performs an LERW from the first node until it hits the tree $\mathcal{T}$ [31]. Once the LERW hits any node in $\mathcal{T}$, the LERW trajectory will be added into $\mathcal{T}$. The algorithm processes the nodes following the fixed node ordering, until all nodes are visited, a rooted spanning tree $\mathcal{T}$ is generated uniformly [6, 31, 50, 65]. To generate a $\mathcal{V}_l$-rooted random spanning forest, we only need to initialize the tree as $\mathcal{T} = \mathcal{V}_l$ and set the root set as $\mathcal{V}_l$, and then invoke the same LERW procedure as the Wilson algorithm. By the results established in [4, 10, 46], such a modified Wilson algorithm can generate a uniformly $\mathcal{V}_l$-rooted random spanning forest.

Second, for each $u \in \mathcal{U}$, Algorithm 2 utilizes the proportion of $\mathcal{V}_l$-rooted random spanning forests in which u is rooted at v as an unbiased estimator of $(\mathbf{P}_f)_{uv}$ based on Lemma 3.6 (Lines 4-6). Similar to Algorithm 1, Algorithm 2 creates an unbiased estimator for each element of the Schur complement $\mathbf{L}/\mathcal{V}_l$ based on Lemma 3.8 (Lines 7-11), and then computes the pseudo-inverse as the estimator of $(\mathbf{L}/\mathcal{V}_l)^\dagger$ using eigen-decomposition (Line 12). In the following, we analyze the accuracy guarantee and time complexity of Algorithm 2.

Theoretical analysis of Algorithm 2. First, similar to Algorithm 1, the sample size for Algorithm 2 to obtain an ϵ-spectral sparsifier can also be derived based on Theorem 4.2.

LEMMA 4.5. *Let $\Delta_{\mathcal{G}}$ be the diameter of the graph $\mathcal{G}$, let $d^{max}_{\mathcal{V}_l}$ be the maximum degree of the nodes in $\mathcal{V}_l$ in the original graph $\mathcal{G}$. When the sample size ω is larger than $O(\frac{\Delta_{\mathcal{G}} d^{max}_{\mathcal{V}_l} \log n}{\epsilon^2})$, the approximate Schur complement graph $\mathcal{H}$ in Algorithm 2 is an ϵ-spectral sparsifier of $\mathcal{G}/\mathcal{V}_l$.*

PROOF. Let $\mathcal{H}_i$ denote the random multi-graph including the edges as follows: in the i-th spanning forest instance F_i, for any $v_1, v_2 \in \mathcal{V}_l$, suppose that $n(\mathcal{N}(v_1), v_2)$ is the number of the nodes $u \in \mathcal{N}(v_i) \cap \mathcal{U}$ which are rooted in v_2, then there is an edge (v_1, v_2) in $\mathcal{H}_i$ with weight $\frac{n(\mathcal{N}(v_1), v_2)}{\omega}$. The spanning forests are independent of each other. Since the weight of the edge (v_i, v_j) for $v_i, v_j \in \mathcal{V}_l$ in the Schur complement $\mathcal{G}/\mathcal{V}_l$ is $\sum_{u \in \mathcal{N}(v_i) \cap \mathcal{U}} p_{uv_j}$, suppose that $\mathcal{H} = \sum_{i=1}^{\omega} \mathcal{H}_i$, we have $E[\mathcal{H}] = \mathcal{G}/\mathcal{V}_l$. Thus, the condition (i) in Theorem 4.2 is satisfied. Moreover, the weight of an edge in a random subgraph that u contributes is at most $\frac{d^{max}_{\mathcal{V}_l}}{\omega}$, when all the neighbors of the node v_1 have the root v_2. As a result, for each edge in the approximate graph $\mathcal{H}_i$, the weight of the edge $w_{\mathcal{H}}(e)$ is at most $\frac{d^{max}_{\mathcal{V}_l}}{\omega}$, we have $w_{\mathcal{H}}(e) r_{\mathcal{G}}(e_1, e_2) \leq \frac{d^{max}_{\mathcal{V}_l} \Delta_{\mathcal{G}}}{\omega}$, since Δ_G is a natural bound of the effective resistance in $\mathcal{G}$ [12]. Thus, when $\omega \geq O(\frac{d^{max}_{\mathcal{V}_l} \Delta_G \log n}{\epsilon^2})$, the condition (ii) in Theorem 4.2 is satisfied. The proof is completed. □

Similar to Algorithm 1, Algorithm 2 can also support an ϵ-estimation query of ER if it constructs an ϵ-spectral sparsifier, which we will discuss later. We can give the overall time complexity as follows:

LEMMA 4.6. *The expected time complexity of Algorithm 2 to obtain an ϵ-spectral sparsifier of $O(\frac{\mathrm{Tr}((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}) \Delta_{\mathcal{G}} d^{max}_{\mathcal{V}_l} \log n}{\epsilon^2})$.*

PROOF. Based on the results in [39, 47], the expected running time of the loop-erased walk procedure in Algorithm 2 is $\mathrm{Tr}((\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})$. According to Lemma 3.5, $\tau_{\mathcal{V}_l}[u,u] = ((\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})_{uu}$. Thus, $\mathrm{Tr}((\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}) = \sum_{u \in \mathcal{U}} \tau_{\mathcal{V}_l}[u,u]$, which is strictly smaller than $\vec{1}^T(\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1} = \sum_{u \in \mathcal{U}} h(u, \mathcal{V}_l)$. In other words, the expected running time of a loop-erased walk with root set $\mathcal{V}_l$ is the sum of all expected number of passes to a node in a $\mathcal{V}_l$-absorbed random walk starts from u for all $u \in \mathcal{U}$. Thus, the total time complexity is $\omega \sum_{u \in \mathcal{U}} \tau_{\mathcal{V}_l}[u,u] = O(\frac{\mathrm{Tr}((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}) \Delta_{\mathcal{G}} d^{max}_{\mathcal{V}_l} \log n}{\epsilon^2})$. □

In Lemma 4.6, $t^{le}_{\mathcal{V}_l} = \mathrm{Tr}((\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})$ is the expected running time of a loop-erased walk with root set $\mathcal{V}_l$, which is strictly smaller than the expected running time of $|\mathcal{U}|$ random walks ($\vec{1}^T(\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}$) in Algorithm 1. According to Lemma 3.5, $\tau_{\mathcal{V}_l}[u,u] = ((\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})_{uu}$. Thus, $t^{le}_{\mathcal{V}_l}$ can also be written as $\sum_{u \in \mathcal{U}} \tau_{\mathcal{V}_l}[u,u]$. Since $\tau_{\mathcal{V}_l}[u,u]$ is the expected number of passes to a node u in a random walk from u before hitting $\mathcal{V}_l$. In real-world networks, if $\mathcal{V}_l$ is properly chosen, there is little probability that a node will pass itself twice. As a result, $t^{le}_{\mathcal{V}_l}$ is nearly $O(n)$. For example, if we choose $\mathcal{V}_l$ as the top-100 highest degree nodes, $t^{le}_{\mathcal{V}_l} = 3.2 \times n$ in a real-life graph PowerGrid [27]. Thus, sampling a spanning forest is much faster than sampling random walks. For the sample size, it is hard to compare the bound of Algorithm 1 ($\frac{|\mathcal{V}_l| \Delta_{\mathcal{G}} \log n}{\epsilon^2}$) and the bound of Algorithm 2 ($\frac{d^{max}_{\mathcal{V}_l} \Delta_{\mathcal{G}} \log n}{\epsilon^2}$). In general, $d^{max}_{\mathcal{V}_l}$ is larger than $|\mathcal{V}_l|$, since we prefer a small set of nodes with high degrees. However, these bounds are worst-case theoretical bounds, we find that in practice, Algorithm 2 can also obtain comparable index quality compared to Algorithm 1 when the sample size is the same. For the overall performance, we will show in experiments that Algorithm 2 is significantly better than Algorithm 1.

4.2 Query processing algorithms

In this section, we propose three novel algorithms to efficiently process any single-pair ER query based on our index. Note that equipped with our index, it is sufficient to compute at most three elements of the matrix $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}$ (i.e., $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{ss}$, $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{tt}$, $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$) to process a ER query $r(s,t)$ based on Lemma 3.7. Below, we first propose three new algorithms to estimate an element of $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}$, followed by the query processing algorithm.

Estimating $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$ by $\mathcal{V}_l$-absorbed random walk. By Lemma 3.4, we have already shown that each element of $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}$ can be determined by the probability of a $\mathcal{V}_l$-absorbed random walk. Based on this, we can easily devise a $\mathcal{V}_l$-absorbed random walk sampling algorithm to estimate the element of $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}$. Specifically, to estimate $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$

Algorithm 2: RsfIndex

Input: Graph $\mathcal{G}$, landmark node set $\mathcal{V}_l$, sample size ω
Output: The estimated index matrices $\widetilde{\mathbf{P}}_f$ and $\mathbf{L}^{\dagger}_{\mathcal{H}}$

$\widetilde{\mathbf{P}}_f \leftarrow \mathbf{0}$; $\mathcal{U} \leftarrow \mathcal{V} \backslash \mathcal{V}_l$;
Let $\mathcal{H}$ be a subgraph of $\mathcal{G}$ induced by the node set $\mathcal{V}_l$;
For each edge $(u, v) \in \mathcal{H}$, let $w(u, v) = 1$ be the weight of (u, v);
for $i = 1 : \omega$ **do**
  Sample a $\mathcal{V}_l$-rooted random spanning forest $F \in \mathcal{F}_{\mathcal{V}_l}$ using the Wilson algorithm (with an initial root set $\mathcal{V}_l$); **for** *each node* $u \in \mathcal{U}$ **do**
    Suppose $v \in \mathcal{V}_l$ is the root of u, $(\widetilde{\mathbf{P}}_f)_{uv} \leftarrow (\widetilde{\mathbf{P}}_f)_{uv} + \frac{1}{\omega}$;
  for *each node* $v \in \mathcal{V}_l$ **do**
    for *each neighbor node* $u \in \mathcal{N}(v) \cap \mathcal{U}$ **do**
      if $v \in \mathcal{V}_l$ *is the root of* u *in* F **then**
        If the edge $(u, v) \in \mathcal{H}$, $w(u, v) \leftarrow w(u, v) + \frac{1}{\omega}$; Otherwise, create a new edge (u, v) with weight $w(u, v) \leftarrow \frac{1}{\omega}$;
Let $\mathbf{L}_{\mathcal{H}}$ be the Laplacian matrix of the weighted graph $\mathcal{H}$;
Compute the pseudo-inverse $\mathbf{L}^{\dagger}_{\mathcal{H}}$ by eigen-decomposition;
return $\widetilde{\mathbf{P}}_f, \mathbf{L}^{\dagger}_{\mathcal{H}}$

($(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{ss}$ and $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{tt}$ can be estimated similarly), we first simulate a number of $\mathcal{V}_l$-absorbed random walks from s, and then take the average number of visits to the node t as an estimator of $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$. Clearly, by Lemma 3.4, such an estimator is unbiased. Note that such an algorithm can also simultaneously estimate $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$ for each node $t \in \mathcal{U}$ by simulating $\mathcal{V}_l$-absorbed random walks from s. This is because for each $\mathcal{V}_l$-absorbed random walk, we can simultaneously count the number of visits for all nodes in this walk. Thus, for each node, we can derive an unbiased estimator by taking the average visits across all $\mathcal{V}_l$-absorbed random walks. Such a $\mathcal{V}_l$-absorbed random walk sampling approach that simultaneously estimates $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$ for every node $t \in \mathcal{U}$ will be used in computing the single-source ER query.

Additionally, it is worth mention that if the landmark node set $\mathcal{V}_l$ contains only one node v, then our algorithm is exactly equivalent to the v-absorbed random walk algorithm proposed in [31]. In this sense, the v-absorbed random walk algorithm [31] is a special case of our algorithm. However, compared to the v-absorbed random walk algorithm, our algorithm is often much more efficient, because the $\mathcal{V}_l$-absorbed random walk (with multiple landmark nodes) typically terminates much faster than v-absorbed random walk (with only one landmark node).

Estimating $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$ by $\mathcal{V}_l$-absorbed push. Push is a powerful technique to compute the personalized PageRank vector [7, 8, 13]. Recently, Liao et al. [31] propose a *push-style* algorithm, called v-absorbed push, to compute the effective resistance. However, their push algorithm only relies on a single landmark nodes. Here, we propose a new *push-style* algorithm to compute the elements of $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}$, called $\mathcal{V}_l$-absorbed push, by extending their technique with multiple landmark nodes. Such a $\mathcal{V}_l$-absorbed push can be regarded as the deterministic version of our $\mathcal{V}_l$-absorbed random walk algorithm. Algorithm 3 details our $\mathcal{V}_l$-absorbed push algorithm.

Let $\boldsymbol{\tau}_s$ be the s-th row of $(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}$. Since $(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\mathbf{D}^{-1}_{\mathcal{U}\mathcal{U}} = \mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}$, we can easily obtain that the t-th element of $\boldsymbol{\tau}_s$ is $\boldsymbol{\tau}_s(t) = \tau_{\mathcal{V}_l}[s, t]$ by Lemma 3.4.

As described in Algorithm 3, a $\mathcal{V}_l$-absorbed push procedure maintains two $|\mathcal{U}|$-dimensional vectors: (i) $\widetilde{\boldsymbol{\tau}}_s$, which is the estimated vector of $\boldsymbol{\tau}_s$; and (ii) "**res**", which is the residual vector. Initially, $\boldsymbol{\tau}_s$

Algorithm 3: The $\mathcal{V}_l$-absorbed push algorithm

Input: Graph $\mathcal{G}$, landmark node set $\mathcal{V}_l$, a source node s, a threshold $r_{\max}$
Output: $\widetilde{\boldsymbol{\tau}}_s$ as an approximation of $\boldsymbol{\tau}_s$

1 $\widetilde{\boldsymbol{\tau}}_s \leftarrow \mathbf{0}$, $\mathbf{res} \leftarrow \mathbf{e}_s$;
2 **while** $\exists u \in \mathcal{U}$ *such that* $\mathbf{res}(u) > r_{\max}$ **do**
3 $\widetilde{\boldsymbol{\tau}}_s(u) \leftarrow \widetilde{\boldsymbol{\tau}}_s(u) + \mathbf{res}(u)$;
4 **for** *each* $w \in \mathcal{N}(u) \cap \mathcal{U}$ **do**
5 $\mathbf{res}(w) \leftarrow \mathbf{res}(w) + \frac{\mathbf{res}(u)}{d(w)}$;
6 $\mathbf{res}(u) \leftarrow 0$;
7 **return** $\widetilde{\boldsymbol{\tau}}_s$;

is set as $\mathbf{0}$ and "**res**" is set as $\mathbf{e}_s$ (Line 1). Then, every time there exists a node $u \in \mathcal{U}$ such that $\mathbf{res}(u) > r_{\max}$, we will conduct push operation on the node u (Lines 2-6). Specifically, we add the residual of u to the estimated value of u (Line 3), and uniformly distribute the residual of u to its neighbors (Lines 4-6). If the neighbor node belongs to $\mathcal{V}_l$, the residual will vanish. It is easy to show (by induction) that the following invariant holds during the whole push process:

$$\tau_{\mathcal{V}_l}[s, u] = \widetilde{\boldsymbol{\tau}}_s(u) + \sum_{w \in \mathcal{U}} \tau_{\mathcal{V}_l}[w, u]\mathbf{res}(w). \tag{13}$$

This is because each push operation in Algorithm 3 (Lines 3-6) does not violate the invariant equation. Note that when the landmark set $\mathcal{V}_l$ contains only one node v, Algorithm 3 degrades to the v-absorbed push proposed by Liao et al. [31]. Likewise, our $\mathcal{V}_l$-absorbed push is often much faster the v-absorbed push, as the residuals can be vanished faster with multiple landmark nodes compared to the case that has only one landmark node. Intuitively, by Eq. (13), the vector $\widetilde{\boldsymbol{\tau}}_s$ output by Algorithm 3 is a good approximation of the *true* vector $\boldsymbol{\tau}_s$ if the threshold $r_{\max}$ is small, because the final residual vector **res** satisfies $\mathbf{res}(w) \leq r_{\max}$ when the algorithm terminates. As a result, by Lemma 3.4, we can easily obtain an approximation of $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$ for all $t \in \mathcal{U}$. Later, we will present detailed error guarantee and time complexity analysis of the $\mathcal{V}_l$-absorbed push algorithm.

Combining $\mathcal{V}_l$-absorbed random walk and push. To further improve the efficiency, we propose a bidirectional algorithm by combining the techniques of $\mathcal{V}_l$-absorbed random walk and $\mathcal{V}_l$-absorbed push based on the invariant equation Eq. (13). Specifically, the bidirectional algorithm first invokes a $\mathcal{V}_l$-absorbed push with a *relatively-large* parameter $r_{\max}$ to obtain a *rough* estimation of $\widetilde{\boldsymbol{\tau}}_s$. Then, by Eq. (13), we can easily show that the additive error of the estimation is bounded by $\sum_{w \in \mathcal{U}} \tau_{\mathcal{V}_l}[w, u]\mathbf{res}(w)$. Here the "**res**" vector is the residual vector output by $\mathcal{V}_l$-absorbed push. Subsequently, to make the estimation more accurate, we can apply $\mathcal{V}_l$-absorbed random walk to estimate the second term of the right hand side of Eq. (13). To achieve this, we draw a source node s' from the probability distribution $\frac{\mathbf{res}}{\|\mathbf{res}\|_1}$, and sample a $\mathcal{V}_l$-absorbed random walk from s'. For each node w the $\mathcal{V}_l$-absorbed random walk visits, we add $\|\mathbf{res}\|_1$ to the estimation $\widetilde{\boldsymbol{\tau}}_s(w)$. It can be shown that this is an unbiased estimation of the second term of the right hand side of Eq. (13). By properly setting the threshold $r_{\max}$ and the sample size ω, we can obtain an algorithm that is better than both of $\mathcal{V}_l$-absorbed random walk and $\mathcal{V}_l$-absorbed push. We will present a detailed analysis of this bidirectional algorithm in the following.

The single-pair ER query processing algorithm. Equipped with the above three different techniques to estimate the elements of

Algorithm 4: The single-pair ER query processing algorithm

Input: Graph $\mathcal{G}$, landmark node set $\mathcal{V}_l$, a source node s, a target node t, two indexed matrices $\mathbf{P}_r$ and $\mathbf{L}^{\dagger}_{\mathcal{H}}$
Output: $\tilde{r}(s,t)$ as an estimation of $r(s,t)$

1 **if** $s,t \in \mathcal{V}_l$ **then**
2 $\quad \tilde{r}(s,t) \leftarrow (\mathbf{e}_s-\mathbf{e}_t)^T\mathbf{L}^{\dagger}_{\mathcal{H}}(\mathbf{e}_s-\mathbf{e}_t)$;
3 **else if** $s \in \mathcal{U}$, $t \in \mathcal{V}_l$ **then**
4 $\quad$ Invoking an algorithm to obtain an estimation $(\widetilde{\mathbf{L}}^{-1}_{\mathcal{U}\mathcal{U}})_{ss}$ of $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{ss}$;
5 $\quad \tilde{r}(s,t) \leftarrow (\widetilde{\mathbf{L}}^{-1}_{\mathcal{U}\mathcal{U}})_{ss} + (\mathbf{p}_s-\mathbf{e}_t)^T\mathcal{L}^{\dagger}_{\mathcal{H}}(\mathbf{p}_s-\mathbf{e}_t)$;
6 **else if** $s \in \mathcal{V}_l$, $t \in \mathcal{U}$ **then**
7 $\quad$ Invoking an algorithm to obtain an estimation $(\widetilde{\mathbf{L}}^{-1}_{\mathcal{U}\mathcal{U}})_{tt}$ of $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{tt}$;
8 $\quad \tilde{r}(s,t) \leftarrow (\widetilde{\mathbf{L}}^{-1}_{\mathcal{U}\mathcal{U}})_{tt} + (\mathbf{p}_t-\mathbf{e}_s)^T\mathcal{L}^{\dagger}_{\mathcal{H}}(\mathbf{p}_t-\mathbf{e}_s)$;
9 **else if** $s,t \in \mathcal{U}$ **then**
10 $\quad$ Invoking an algorithm to obtain an estimation $(\widetilde{\mathbf{L}}^{-1}_{\mathcal{U}\mathcal{U}})_{ss}$, $(\widetilde{\mathbf{L}}^{-1}_{\mathcal{U}\mathcal{U}})_{tt}$ and $(\widetilde{\mathbf{L}}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$ of $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{ss}$, $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{tt}$ and $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$ respectively;
11 $\quad \tilde{r}(s,t) \leftarrow (\mathbf{e}_s-\mathbf{e}_t)^T\widetilde{\mathbf{L}}^{-1}_{\mathcal{U}\mathcal{U}}(\mathbf{e}_s-\mathbf{e}_t) + (\mathbf{p}_s-\mathbf{p}_t)^T\mathbf{L}^{\dagger}_{\mathcal{H}}(\mathbf{p}_s-\mathbf{p}_t)$;
12 **return** $\tilde{r}(s,t)$

$\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}$, we propose three algorithms for processing the single-pair ER query based on Lemma 3.7.

The detailed description of our algorithm is shown in Algorithm 4. Algorithm 4 takes two indexed matrices $\mathbf{P}_r$ ($\mathbf{P}_f$) and $\mathbf{L}^{\dagger}_{\mathcal{H}}$ as input. Let $\mathbf{p}_{u_i}$ be the u_i-th row of $\mathbf{P}_r$. By Lemma 3.7, there are four cases depending on which set s and t belongs to. When $s,t \in \mathcal{V}_l$, i.e., both of them belong to the landmark node set, the algorithm computes an estimator $\tilde{r}(s,t)$ using the indexed matrix $\mathbf{L}^{\dagger}_{\mathcal{H}}$ within $O(1)$ time (Lines 1-2). When one node (s or t) belongs to $\mathcal{U}$, the other node is located in $\mathcal{V}_l$, Algorithm 4 first invokes an algorithm devised above (e.g., $\mathcal{V}_l$-absorbed random walk, or $\mathcal{V}_l$-absorbed push, or the bidirectional algorithm) to estimate only one element of $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}$, and then compute $\tilde{r}(s,t)$ by Lemma 3.7 (Lines 3-8). When $s,t \in \mathcal{U}$, the situation becomes more complicated. Specifically, in this case, Algorithm 4 needs to estimate three elements of $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}$ by Lemma 3.7 (Lines 9-11). For convenience, we refer to Algorithm 4 that is equipped with $\mathcal{V}_l$-absorbed random walk, $\mathcal{V}_l$-absorbed push, and the bidirectional algorithm as RW, Push and Bipush respectively. Below, we present comprehensive theoretical analyses of these algorithms.

Theoretical analysis of the query processing algorithms. First, we give the time complexity for the three techniques to achieve an ϵ-estimation of $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$. We will utilize the following Hoeffding lemma in our analysis:

LEMMA 4.7. *(Hoeffding lemma [18]) Let $X_1,\cdots,X_\omega$ be random variables with expectation $E[X]$. Suppose that X can be bounded by $a \le X \le b$. Then, with high probability (e.g., the failure probability is smaller than 0.01), when $\omega \ge \frac{(a-b)^2}{\epsilon^2}$, we have $|\frac{1}{\omega}\sum_{i=1}^{\omega} - E[X]| \le \epsilon$.*

Notice that for our $\mathcal{V}_l$-absorbed sampling algorithm, although it usually terminates very fast, in the worst case the random walk length can be infinite. Thus, it is hard to directly analyze the algorithm. However, if we truncate the random walks with length σ, we can give a rigorous analysis of the modified algorithm, by separately analyzing the sampling error and the truncated error.

LEMMA 4.8. *Let $\lambda_{\mathcal{V}_l}$ be the spectral radius of the probability transition matrix $\mathbf{P}_{\mathcal{U}\mathcal{U}}$, $\sigma_{\mathcal{V}_l} = O(\frac{\log(1/(\epsilon(1-\lambda_{\mathcal{V}_l})))}{\log(1/\lambda_{\mathcal{V}_l})})$, the $\mathcal{V}_l$-absorbed random walk sampling algorithm can estimate $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$ with ϵ absolute error in time $O(\frac{\sigma^3_{\mathcal{V}_l}}{\epsilon^2})$.*

PROOF. For an arbitrary element $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$, recall that $\mathbf{L}_{\mathcal{U}\mathcal{U}} = (\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\mathbf{D}_{\mathcal{U}\mathcal{U}}$. We have $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st} = \frac{((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})_{st}}{d_t}$. By sampling random walks with length at most l from s to hit $\mathcal{V}_l$, suppose that X is the random variable that represents the number of passes to the node t, we have $E[\frac{X}{d_t}] = (\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$. The total error consists two parts: (i) the sampling error ϵ_1 and (ii) the truncation error ϵ_2. For the (i) sampling error, since $X \le \sigma$, according to Hoffeding bound, when $\omega \ge O(\frac{\sigma^2}{\epsilon_1^2})$, with high probability, the first part error is less than ϵ_1. For the (ii) truncated error, we bound the error $\epsilon_2 = \mathbf{e}_s^T\sum_{k=\sigma+1}^{\infty}\mathbf{P}^k_{\mathcal{U}\mathcal{U}}\mathbf{e}_{e_t}$. We have $\|\mathbf{e}_s^T\sum_{k=\sigma+1}^{\infty}\mathbf{P}^k_{\mathcal{U}\mathcal{U}}\mathbf{e}_{e_t}\|_2 \le \|\mathbf{e}_s\|_2\frac{\lambda^{\sigma+1}_{\mathcal{V}_l}}{1-\lambda_{\mathcal{V}_l}}\|\mathbf{e}_t\|_2 = \frac{\lambda^{\sigma+1}_{\mathcal{V}_l}}{1-\lambda_{\mathcal{V}_l}}$. Thus, when $\sigma \ge O(\frac{\log(1/(\epsilon_2(1-\lambda_{\mathcal{V}_l})))}{\log(1/\lambda_{\mathcal{V}_l})})$, the second part error can be bound by ϵ_2. Let $\epsilon_1 = \epsilon_2 = \frac{\epsilon}{2}$, the total time complexity for the $\mathcal{V}_l$-absorbed random walk sampling algorithm to achieve an ϵ absolute error of $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$ is $\omega\sigma = O(\frac{\log^3(1/(\epsilon(1-\lambda_{\mathcal{V}_l})))}{\epsilon^2\log^3(1/\lambda_{\mathcal{V}_l})}) = O(\frac{\sigma^3_{\mathcal{V}_l}}{\epsilon^2})$, which establishes the Lemma. □

Note that $\lambda_{\mathcal{V}_l}$ is often near to 1 [45]. Thus, by Lemma 4.8, $\sigma_{\mathcal{V}_l}$ is a small number, which results in an efficient sublinear algorithm. Recall that the v-absorbed walk sampling algorithm proposed in [31] is a special case of the $\mathcal{V}_l$-absorbed walk sampling algorithm when $\mathcal{V}_l = \{v\}$. Let λ_v denote the spectral radius of $\mathbf{P}_v$, $\sigma_v = O(\frac{\log(1/\epsilon(1-\lambda_v))}{\log(1/\lambda_v)})$. The time complexity for the v-absorbed walk sampling algorithm is $O(\frac{\sigma^3_v}{\epsilon^2})$. According to the Cauchy Interlacing theorem [12], $\lambda_{\mathcal{V}_l^{(1)}} \le \lambda_{\mathcal{V}_l^{(2)}}$ if $\mathcal{V}_l^{(1)} \subseteq \mathcal{V}_l^{(2)}$. Thus, we have $\sigma_{\mathcal{V}_l} \le \sigma_v$, the theoretical bound of the algorithm is strictly better than [31]. For example, on a real-life road network PowerGrid, $\sigma_v = 1.5\times10^5$ while $\sigma_{\mathcal{V}_l} = 1321$ if $\mathcal{V}_l$ is the top-100 highest degree nodes. Next, we derive a theoretical bound for the $\mathcal{V}_l$-absorbed push algorithm.

LEMMA 4.9. *Let $\bar{d} = \frac{2m}{n}$ be the average degree, the $\mathcal{V}_l$-absorbed push algorithm can estimate $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$ with ϵ absolute error in time $O(\frac{\bar{d}h(s,\mathcal{V}_l)h(t,\mathcal{V}_l)}{\epsilon})$.*

PROOF. For an arbitrary element $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{st}$, the $\mathcal{V}_l$-absorbed push algorithm with source node s estimates the s-th row of the matrix $(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}$, then divided the elements by degree. In each push operation, the algorithm will add at most r_{max} to the estimation vector $\boldsymbol{\tau}$. During the whole $\mathcal{V}_l$-absorbed push algorithm, it will add at most $\mathbf{e}_s^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}$ to the vector $\boldsymbol{\tau}$. Thus, there are at most $O(\frac{\mathbf{e}_s^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}}{r_{max}})$ push operations. The amortized time of each push operation is $\bar{d}$ [36], which results in the time complexity $O(\frac{\bar{d}\mathbf{e}_s^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}}{r_{max}}) = O(\frac{\bar{d}h(s,\mathcal{V}_l)}{r_{max}})$, where $h(s,\mathcal{V}_l)$ is the hitting time from s to $\mathcal{V}_l$. At the same time, according to Eq. (13), the additive error of the t-th element is bounded by $r_{max}\vec{1}^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\mathbf{e}_t$. Since

we have $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{st} = \frac{((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})_{st}}{d_t}$, the additive error of $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{st}$ can be bounded by $r_{max}\frac{\vec{1}^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\mathbf{e}_t}{d_t} \leq r_{max}\mathbf{e}_t^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1} = r_{max}h(t,\mathcal{V}_l)$. By setting $r_{max} = \frac{\epsilon}{h(t,\mathcal{V}_l)}$, the total time complexity to achieve an ϵ absolute error is $O(\frac{\bar{d}h(s,\mathcal{V}_l)h(t,\mathcal{V}_l)}{\epsilon})$ for estimating $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{st}$. Thus, the lemma is established. □

Based on the results of Lemma 4.9, the time complexity of the bidirectional algorithm is closely related to the hitting time from the query node to $\mathcal{V}_l$. Notice that the push algorithm proposed in [31] is $\frac{\bar{d}h(s,v)h(t,v)}{\epsilon}$ based on our results. The hitting time $h(u,\mathcal{V}_l)$ is intuitively smaller than $h(u,v)$ for $u \in \mathcal{U}$, since there are more nodes to hit. Thus, the time complexity of our algorithm is strictly lower than that of the algorithm in [31]. For example, on a real-life network PowerGrid, $\bar{h}_v = \sum_{u\in\mathcal{V}\setminus\{v\}} h(u,v)$ is 1.3×10^4 while $\bar{h}_{\mathcal{V}_l} = \sum_{u\in\mathcal{U}} h(u,\mathcal{V}_l)$ is 71 if $\mathcal{V}_l$ is the top-100 highest degree nodes. Next, we present theoretical analysis for the bidirectional algorithm.

LEMMA 4.10. *The bidirectional algorithm can estimate* $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{st}$ *with* ϵ *absolute error in time* $O(\frac{(n\bar{d}h(s,\mathcal{V}_l))^{\frac{2}{3}}\sigma_{\mathcal{V}_l}}{\epsilon^{\frac{2}{3}}})$.

PROOF. The bidirectional algorithm first applies a $\mathcal{V}_l$-absorbed push procedure, then performs random walks. Similar to Lemma 4.8, we can analyze the algorithm when the random walks are truncated by a length σ, and the total error of the algorithm is the sum of the sampling error and the truncated error. When the push procedure terminates, all the elements of the residual vector "**res**" are less than r_{max}. According to Eq. (13), let X be the random variable that a V_l-absorbed walk starts from a node s' sampled from the distribution $\frac{\mathbf{res}}{\|\mathbf{res}\|_1}$ passes node t for X times, we have $E[\|\mathbf{res}\|_1 X] = \tau_{\mathcal{V}_l}[s,t] - \widetilde{\tau}_s(t)$. Since there are less than n elements in the vector "**res**", $\|\mathbf{res}\|_1$ can be bounded by nr_{max}. Similar to the proof of Lemma 4.8, X can be bounded by σ, $\|\mathbf{res}\|_1 X$ can be bounded by $nr_{max}\sigma$, we have that when $\omega \geq \frac{n^2r_{max}^2\sigma^2}{\epsilon_1^2}$, the sampling error is smaller than ϵ_1. Likewise, the truncated error $\epsilon_2 = (\frac{\mathbf{res}}{\|\mathbf{res}\|_1})^T\sum_{k=\sigma+1}^{\infty}\mathbf{P}_{\mathcal{U}\mathcal{U}}^k\mathbf{e}_t$. We have $\|(\frac{\mathbf{res}}{\|\mathbf{res}\|_1})^T\sum_{k=\sigma+1}^{\infty}\mathbf{P}_{\mathcal{U}\mathcal{U}}^k\mathbf{e}_t\|_2 \leq \|(\frac{\mathbf{res}}{\|\mathbf{res}\|_1})\|_2\frac{\lambda_{\mathcal{V}_l}^{\sigma+1}}{1-\lambda_{\mathcal{V}_l}}\|\mathbf{e}_t\|_2 = \frac{\lambda_{\mathcal{V}_l}^{\sigma+1}}{1-\lambda_{\mathcal{V}_l}}$. Thus, when $\sigma \geq O(\frac{\log(1/(\epsilon_2(1-\lambda_{\mathcal{V}_l})))}{\log(1/\lambda_{\mathcal{V}_l})}) = \sigma_{\mathcal{V}_l}$, the second part error can be bound by ϵ_2. Let $\epsilon_1 = \epsilon_2 = \frac{\epsilon}{2}$, the total absolute error is smaller than ϵ with high probability. According to Lemma 4.9, the push procedure costs time $O(\frac{\bar{d}h(s,\mathcal{V}_l)}{r_{max}})$. The sampling procedure costs time $O(\frac{n^2r_{max}^2\sigma_{\mathcal{V}_l}^3}{\epsilon^2})$. Thus, the total time complexity is minimized when $r_{max} = \sqrt[3]{\frac{\bar{d}h(s,\mathcal{V}_l)}{2n^2r_{max}^2\sigma_{\mathcal{V}_l}^3}}$, the total time complexity is thereby $O(\frac{(n\bar{d}h(s,\mathcal{V}_l))^{\frac{2}{3}}\sigma_{\mathcal{V}_l}}{\epsilon^{\frac{2}{3}}})$. The proof is completed. □

As the bidirectional algorithm is a combination of the $\mathcal{V}_l$-absorbed walk sampling algorithm and the $\mathcal{V}_l$-absorbed push algorithm, the dependency of the bound on $\sigma_{\mathcal{V}_l}$, $h(u,\mathcal{V}_l)$ and ϵ is better than both of the algorithms. Although there is an additional n term, the bound presented in Lemma 4.10 is only a worst-case theoretical bound. We find that in experiments, the bidirectional algorithm performs much better than both $\mathcal{V}_l$-absorbed random walk sampling and $\mathcal{V}_l$-absorbed push. Moreover, as discussed above, the time complexity bound is also strictly better than the Bipush algorithm proposed in [31], since $\sigma_{\mathcal{V}_l}$, $h(u,\mathcal{V}_l)$ are both smaller.

Recall that in Algorithm 4, the error of the single-pair query algorithm includes both the error of building index and the error of estimating $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$. Next, we first show the time complexity for Algorithm 1 and Algorithm 2 to achieve an ϵ-absolute error of index building. Then, we combine the error of estimating $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$ to give the total error of the query processing algorithms. Here, the error of index building is different in four cases, depending on the query node s and t: (i) If $s,t \in \mathcal{V}_l$, it is the error of estimating $(\mathbf{e}_s-\mathbf{e}_t)^T\mathbf{L}_{\mathcal{H}}^{\dagger}(\mathbf{e}_s-\mathbf{e}_t)$; (ii) If $s \in \mathcal{U}$, $t \in \mathcal{V}_l$, it is the error of estimating $(\mathbf{p}_s-\mathbf{e}_t)^T\mathcal{L}_{\mathcal{H}}^{\dagger}(\mathbf{p}_s-\mathbf{e}_t)$; (iii) If $s \in \mathcal{V}_l$, $t \in \mathcal{U}$, it is the error of estimating $(\mathbf{p}_t-\mathbf{e}_s)^T\mathcal{L}_{\mathcal{H}}^{\dagger}(\mathbf{p}_t-\mathbf{e}_s)$; (iv) If $s,t \in \mathcal{V}_l$, is the error of estimating $(\mathbf{p}_s-\mathbf{p}_t)^T\mathcal{L}_{\mathcal{H}}^{\dagger}(\mathbf{p}_s-\mathbf{p}_t)$. Since all the four cases deal with the bilinear form of the pseudo-inverse of the Schur complement $\mathbf{L}/\mathcal{V}_l$, and we have shown the time complexity for Algorithm 2 and Algorithm 2 to output an ϵ-spectral sparsifier, we will utilize the following lemma proposed in [23].

LEMMA 4.11. *(Property of an* ϵ*-spectral sparsifier [23]) If* $\mathcal{H}$ *is an* ϵ*-spectral sparsifier of* $\mathcal{G}$*, let* $\mathbf{L}_{\mathcal{G}}$ *and* $\mathbf{L}_{\mathcal{H}}$ *be the Laplacian matrix of* $\mathcal{G}$ *and* $\mathcal{H}$*, respectively. Then, for any vector* $\mathbf{x}$*, we have:* $(1-\epsilon)\mathbf{x}^T\mathbf{L}_{\mathcal{G}}^{\dagger}\mathbf{x} \leq \mathbf{x}^T\mathbf{L}_{\mathcal{H}}^{\dagger}\mathbf{x} \leq (1+\epsilon)\mathbf{x}^T\mathbf{L}_{\mathcal{G}}^{\dagger}\mathbf{x}$.

Notice that Lemma 4.11 is not enough for the error of index building, since in cases (ii)-(iv), the probability vector $\mathbf{p}_u$ for $u \in \mathcal{U}$ is also an estimated value. Below, we show the time complexity for Algorithm 1 and Algorithm 2 to achieve an ϵ-absolute error of index building.

LEMMA 4.12. *Algorithm 1 can achieve an* ϵ*-absolute error for index building in time* $O(\frac{\vec{1}^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}\Delta_{\mathcal{G}}^3|\mathcal{V}_l|\log n}{\epsilon^2})$.

PROOF. As we have proved in Lemma 4.4, Algorithm 1 can output an $\frac{\epsilon}{\Delta_{\mathcal{G}}}$-spectral sparsifier $\mathcal{H}$ in time $O(\frac{\vec{1}^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}\Delta_{\mathcal{G}}^3|\mathcal{V}_l|\log n}{\epsilon^2})$. Based on Theorem 4.11, for any $\mathbf{x}$, we have: $(1-\epsilon)\mathbf{x}^T(\mathbf{L}/\mathcal{V}_l)^{\dagger}\mathbf{x} \leq \mathbf{x}^T\mathbf{L}_{\mathcal{H}}^{\dagger}\mathbf{x} \leq (1+\epsilon)\mathbf{x}^T(\mathbf{L}/\mathcal{V}_l)^{\dagger}\mathbf{x}$. Below, we discuss the four cases of the error index building. (i) If $s,t \in \mathcal{V}_l$, the error of index building is $(\mathbf{e}_s-\mathbf{e}_t)^T\mathbf{L}_{\mathcal{H}}^{\dagger}(\mathbf{e}_s-\mathbf{e}_t)$. The output of the ER value in Algorithm 4 is also $(\mathbf{e}_s-\mathbf{e}_t)^T\mathbf{L}_{\mathcal{H}}^{\dagger}(\mathbf{e}_s-\mathbf{e}_t)$. Thus, let $\mathbf{x} = \mathbf{e}_s-\mathbf{e}_t$, it directly holds that $|\hat{r}(s,t)-r(s,t)| \leq \frac{\epsilon}{\Delta_{\mathcal{G}}}r(s,t) \leq \frac{\epsilon}{\Delta_{\mathcal{G}}}\Delta_{\mathcal{G}} = \epsilon$. Thus, the absolute error of index building in case (i) is smaller than ϵ. (ii) When $s \in \mathcal{U}$, $t \in \mathcal{V}_l$, $r(s,t) = (\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{ss} + (\mathbf{e}_t-\mathbf{p}_s)^T\mathbf{L}_{\mathcal{H}}^{\dagger}(\mathbf{e}_t-\mathbf{p}_s)$. Since $\mathbf{p}_s$ is the $|\mathcal{V}_l|$-dimensional probability vector that the v-th element represents the probability of a random walk starts from s hits $\mathcal{V}_l$ by v. Thus, suppose that the random walk starts from s hits $\mathcal{V}_l$ by v, let X_v denote the random variable that $X_v = (\mathbf{e}_t-\mathbf{e}_v)^T\mathbf{L}_{\mathcal{H}}^{\dagger}(\mathbf{e}_t-\mathbf{e}_v) = r_{\mathcal{H}}(t,v)$, then we have $E[X_v] = \sum_{v\in\mathcal{V}_l}\mathbf{p}(v)r_{\mathcal{H}}(t,v) = \sum_{v\in\mathcal{V}_l}\mathbf{p}(v)r(t,v) = (\mathbf{e}_t-\mathbf{p}_s)^T\mathbf{L}_{\mathcal{H}}^{\dagger}(\mathbf{e}_t-\mathbf{p}_s)$. Since $\mathcal{H}$ is an ϵ-spectral sparsifier of $\mathcal{G}/\mathcal{V}_l$, we have $r_{\mathcal{H}}(t,v) = (\mathbf{e}_t-\mathbf{e}_v)^T\mathbf{L}_{\mathcal{H}}^{\dagger}(\mathbf{e}_t-\mathbf{e}_v) \leq (1+\epsilon)r(t,v) \leq (1+\epsilon)\Delta_{\mathcal{G}}$. In other words, the random variable X_v is bounded by $(1+\epsilon)\Delta_{\mathcal{G}}$. Thus, according to the Hoeffding lemma, with high probability, when

the sampling size in Algorithm 1 satisfies $\omega \geq (1+\epsilon)^2\Delta_{\mathcal{G}}^2\frac{\log n}{\epsilon^2}$, we have $|(\mathbf{e}_t-\mathbf{p}_s)^T\mathbf{L}_{\mathcal{H}}^{\dagger}(\mathbf{e}_t-\mathbf{p}_s)-(\mathbf{e}_t-\mathbf{p}_s)^T\mathbf{L}/\mathcal{V}_l^{\dagger}(\mathbf{e}_t-\mathbf{p}_s)| \leq \epsilon$. Thus, for case (ii), we can also achieve an ϵ-absolute error in the desired complexity. (iii) When $s \in \mathcal{V}_l$, $t \in \mathcal{U}$, the situation is similar to (ii). (iv) When $s, t \in \mathcal{U}$, $r(s,t) = (\mathbf{e}_s-\mathbf{e}_t)^T\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}(\mathbf{e}_s-\mathbf{e}_t)+(\mathbf{p}_s-\mathbf{p}_t)^T\mathbf{L}_{\mathcal{H}}^{\dagger}(\mathbf{p}_s-\mathbf{p}_t)$. Again, suppose that the random walk starts from s hits $\mathcal{V}_l$ by v_1, the random walk starts from t hits $\mathcal{V}_l$ by v_2, let X denote the random variable that $X = (\mathbf{e}_{v_1} - \mathbf{e}_{v_2})^T\mathbf{L}_{\mathcal{H}}^{\dagger}(\mathbf{e}_{v_1} - \mathbf{e}_{v_2}) = r_{\mathcal{H}}(v_1, v_2)$, then we have $E[X_v] = \sum_{v_1,v_2\in\mathcal{V}_l}\mathbf{p}(v_1)\mathbf{p}(v_2)r_{\mathcal{H}}(v_1,v_2) = (\mathbf{p}_{v_1} - \mathbf{p}_{v_2})^T(\mathcal{G}/\mathcal{V}_l)^{\dagger}(\mathbf{p}_{v_1} - \mathbf{p}_{v_2})$. With high probability, $r_{\mathcal{H}}(v_1,v_2) \leq (1+\epsilon)r(v_1,v_2) \leq (1+\epsilon)\Delta_{\mathcal{G}}$. The random variable X can also be bounded by $(1+\epsilon)\Delta_{\mathcal{G}}$. Thus, according to the Hoeffding lemma, with high probability, when $\omega \geq (1+\epsilon)^2\Delta_{\mathcal{G}}^2\frac{\log n}{\epsilon^2} = O(\Delta_{\mathcal{G}}^2\frac{\log n}{\epsilon^2})$, we have $|(\mathbf{p}_s - \mathbf{p}_t)^T\mathbf{L}_{\mathcal{H}}^{\dagger}(\mathbf{p}_s - \mathbf{p}_t) - (\mathbf{e}_s - \mathbf{p}_t)^T\mathbf{L}/\mathcal{V}_l^{\dagger}(\mathbf{p}_s - \mathbf{p}_t)| \leq \epsilon$. Thus, for case (iv), we can also achieve an ϵ-absolute error in the desired complexity. The proof is completed. □

LEMMA 4.13. *Algorithm 2 can achieve an ϵ-absolute error for index building in time $O(\frac{\mathrm{Tr}((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})d_{\mathcal{V}_l}^{max}\Delta_{\mathcal{G}}^2|\mathcal{V}_l|\log n}{\epsilon^2})$.*

PROOF. According to Lemma 4.6, Algorithm 2 can output an ϵ-spectral sparsifier in time $O(\frac{\mathrm{Tr}((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})d_{\mathcal{V}_l}^{max}|\mathcal{V}_l|\log n}{\epsilon^2})$. Similar to the proof of Lemma 4.12, Algorithm 2 can achieve an ϵ-absolute error of index building in time $O(\frac{\mathrm{Tr}((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})d_{\mathcal{V}_l}^{max}\Delta_{\mathcal{G}}^2|\mathcal{V}_l|\log n}{\epsilon^2})$. The proof is completed. □

Based on Lemma 4.12 and Lemma 4.13, we can give the query time complexity for all the three algorithms to achieve an ϵ-absolute error of $r(s,t)$ based on the fact that we have built the index with an ϵ-absolute error.

LEMMA 4.14. *The query complexity of* RW *to achieve an ϵ-absolute error for $r(s,t)$ is $O(\frac{\sigma_{\mathcal{V}_l}^3}{\epsilon^2})$.*

PROOF. Based on Lemma 4.8, RW can output an estimation of $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{st}$ with ϵ-absolute error in time $O(\frac{\sigma_{\mathcal{V}_l}^3}{\epsilon^2})$. Since there are at most four elements of $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$ to estimate, the query complexity of RW is $O(\frac{\sigma_{\mathcal{V}_l}^3}{\epsilon^2})$ if we use the $O(\cdot)$ notation to ignore the constant terms. □

LEMMA 4.15. *Let $h_{\mathcal{V}_l} = \max\{h(s,\mathcal{V}_l), h(t,\mathcal{V}_l)\}$. The query complexity of* Push *to achieve an ϵ-absolute error for $r(s,t)$ is $O(\frac{\bar{d}h_{\mathcal{V}_l}^2}{\epsilon})$.*

PROOF. Based on Lemma 4.9, Push can output an estimation of $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{st}$ with ϵ-absolute error in time $O(\frac{\bar{d}h(s,\mathcal{V}_l)h(t,\mathcal{V}_l)}{\epsilon})$. Since there are at most four elements of $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$ to estimate, the total time complexity can also be bounded by $O(\frac{\bar{d}h_{\mathcal{V}_l}^2}{\epsilon})$ if we use the $O(\cdot)$ notation to ignore the constant terms. □

LEMMA 4.16. *The query complexity of* Bipush *algorithm to achieve an ϵ-absolute error for $r(s,t)$ is $O(\frac{(n\bar{d}h_{\mathcal{V}_l})^{\frac{2}{3}}\sigma_{\mathcal{V}_l}}{\epsilon^{\frac{2}{3}}})$.*

PROOF. Based on Lemma 4.9, Push can output an estimation of $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{st}$ with ϵ-absolute error in time $O(\frac{(n\bar{d}h(s,\mathcal{V}_l))^{\frac{2}{3}}\sigma_{\mathcal{V}_l}}{\epsilon^{\frac{2}{3}}})$. Since there are at most four elements of $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$ to estimate, the total time complexity is $O(\frac{(n\bar{d}h_{\mathcal{V}_l})^{\frac{2}{3}}\sigma_{\mathcal{V}_l}}{\epsilon^{\frac{2}{3}}})$ if we use the $O(\cdot)$ notation to ignore the constant terms. □

As discussed above, the bounds of the three query algorithms are strictly better than the RW, Push and Bipush algorithm proposed in [31], which are all special cases of our algorithms. Moreover, the bound is also better than the other SOTA method GEER [67]. Let $\lambda_1 \leq \cdots \leq \lambda_n = 1$ be the eigenvalues of the probability transition matrix $\mathbf{P}$, $\lambda = \max\{|\lambda_1|, \lambda_{n-1}\}$. Then, suppose that $\sigma = O(\frac{\log(1/\epsilon(1-\lambda))}{\log(1/\lambda)})$, the time complexity bound for GEER to achieve an ϵ-absolute error is $O(\frac{\sigma^3}{\epsilon^2})$, which is similar to the bound $O(\frac{\sigma_{\mathcal{V}_l}^3}{\epsilon^2})$ of RW. Although it is hard to compare these bounds in theory. We find that in practice, when $\mathcal{V}_l$ is properly selected, $\sigma_{\mathcal{V}_l}$ can be significantly smaller than σ. For example, on a real-life road graph PowerGrid [27], $\sigma_{\mathcal{V}_l} = 1321$ if we select $\mathcal{V}_l$ as the top-100 highest degree nodes, while σ is as large as 3.3×10^4. Thus, the theoretical bound of our query processing algorithms RW to achieve an ϵ-estimation of effective resistance is strictly better than the SOTA methods. Push and Bipush can be better than RW further as we have discussed. We will show in experiments that the empirical performance of our query algorithms are significantly better than the SOTA methods.

Discussions. In the above analysis, we give time complexities based on quantities defined by specific graph parameters. Next, we state that we can also derive bounds related to the graph size n. In the following analysis, we assume that real-world graphs are always scale-free (i.e., $m = O(n\log n)$) and small-world (i.e., $\Delta_{\mathcal{G}} = O(\log n)$) graphs. These assumptions are widely adopted by previous studies [9, 32, 61, 63, 66]. At the same time, we assume that real-world graphs are always rapid-mixing [43]. Here, the mixing time of a graph is defined as the minimal length of the random walk to reach the stationary distribution. A typical upper bound of the mixing time is $\frac{1}{1-\lambda}$, where λ is the spectral radius of the probability transition matrix $\mathbf{P}$. It is well-known that $\frac{1}{1-\lambda} = O(\log n)$ in real-world graphs [43]. As we have discussed before, $\lambda_{\mathcal{V}_l}$ is smaller than λ after properly selecting landmarks, thus we also assume that $\frac{1}{1-\lambda_{\mathcal{V}_l}} = O(\log n)$. Notice that similar assumptions are widely adopted by previous studies related to effective resistance computation [45, 67], and a wide range of very recent theoretical papers [16, 21, 29]. Let $\bar{h}(\mathcal{V}_l) = \sum_{u\in\mathcal{U}}\frac{1}{|\mathcal{U}|}h(u,\mathcal{V}_l)$ be the average hitting time from all nodes in $\mathcal{U}$ to $\mathcal{V}_l$, we first show that under such assumptions, several quantities in the above analysis can be simplified as $O(\log n)$.

LEMMA 4.17. *When the underlying graph is fast mixing after selecting a landmark node set $\mathcal{V}_l$, i.e. $\frac{1}{1-\lambda_{\mathcal{V}_l}} = O(\log n)$, we can derive that $\sigma_{\mathcal{V}_l} = O(\log n)$ and $\bar{h}(\mathcal{V}_l) = O(\log n)$.*

PROOF. By setting $\epsilon = 1$ in Lemma 4.8, when $\lambda_{\mathcal{V}_l} = 1 - \frac{1}{O(\log n)}$, the random walk length $\sigma_{\mathcal{V}_l} = O(\frac{\log(1/(1-\lambda_{\mathcal{V}_l}))}{1/\lambda_{\mathcal{V}_l}}) = O(\log n)$. For

the average hitting time, we have $\bar{h}(\mathcal{V}_l) = \frac{\vec{1}^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}}{|\mathcal{U}|}$. Since $\mathcal{V}_l$ is often selected as a small set, we can directly consider $\frac{\vec{1}^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}}{n}$. We have:

$$\begin{aligned}
&\frac{\vec{1}^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}}{n}\\
&= (\frac{\vec{1}}{\sqrt{n}})^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\frac{\vec{1}}{\sqrt{n}}\\
&= (\frac{\vec{1}}{\sqrt{n}})^T\sum_{k=0}^{\infty}\mathbf{P}^k_{\mathcal{U}\mathcal{U}}\frac{\vec{1}}{\sqrt{n}}\\
&= (\frac{\vec{1}}{\sqrt{n}})^T\sum_{k=0}^{\sigma_{\mathcal{V}_l}}\mathbf{P}^k_{\mathcal{U}\mathcal{U}}\frac{\vec{1}}{\sqrt{n}} + (\frac{\vec{1}}{\sqrt{n}})^T\sum_{k=\sigma_{\mathcal{V}_l}+1}^{\infty}\mathbf{P}^k_{\mathcal{U}\mathcal{U}}\frac{\vec{1}}{\sqrt{n}},
\end{aligned}$$

For the first term, for any $k \in [0, \sigma_{\mathcal{V}_l}]$, we have $(\frac{1}{\sqrt{n}})^T\mathbf{P}^k_{\mathcal{U}\mathcal{U}}\frac{1}{\sqrt{n}} \leq 1$, since $\mathbf{P}_{\mathcal{U}\mathcal{U}}$ is a submatrix of a probability transition matrix. Thus, the first term is $O(\log n)$. For the second term, since $\sigma_{\mathcal{V}_l} = O(\frac{\log(1/(1-\lambda_{\mathcal{V}_l}))}{1/\lambda_{\mathcal{V}_l}})$, we have:

$$\begin{aligned}
(\frac{\vec{1}}{\sqrt{n}})^T\sum_{k=\sigma_{\mathcal{V}_l}+1}^{\infty}\mathbf{P}^k_{\mathcal{U}\mathcal{U}}\frac{\vec{1}}{\sqrt{n}} &= \|(\frac{\vec{1}}{\sqrt{n}})^T\sum_{k=\sigma+1}^{\infty}\mathbf{P}^k_{\mathcal{U}\mathcal{U}}(\frac{\vec{1}}{\sqrt{n}})\|_2\\
&\leq \|(\frac{1}{\sqrt{n}})\|_2\frac{\lambda^{\sigma+1}_{\mathcal{V}_l}}{1-\lambda_{\mathcal{V}_l}}\|(\frac{1}{\sqrt{n}})\|_2\\
&= \frac{\lambda^{\sigma+1}_{\mathcal{V}_l}}{1-\lambda_{\mathcal{V}_l}} = O(1),
\end{aligned}$$

Thus, we can derive that $\bar{h}(\mathcal{V}_l) = O(\log n)$. □

Then, equipped with Lemma 4.17, we have the following results:

LEMMA 4.18. *Suppose that $m = O(n\log n)$, $\Delta_{\mathcal{G}} = O(\log n)$ and $\frac{1}{1-\lambda_{\mathcal{V}_l}} = O(\log n)$, the index building algorithms* RwIndex *(Algorithm 1) and* RsfIndex *(Algorithm 2) can both achieve a near-linear time complexity $\widetilde{O}(n)$*[1].

PROOF. According to Lemma 4.12, Algorithm 1 can achieve an ϵ-absolute error for index building in time $O(\frac{\vec{1}^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1}\Delta^3_{\mathcal{G}}|\mathcal{V}_l|\log n}{\epsilon^2})$. Under our assumptions, $\vec{1}^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1} = O(n\log n)$, the diamater $\Delta_{\mathcal{G}}$ also satisfies $\Delta_{\mathcal{G}} = O(\log n)$. $|\mathcal{V}_l|$ is often set as a small number. Thus, the time complexity of Algorithm 1 is $O(\frac{n\log^5 n}{\epsilon^2}) = \widetilde{O}(n)$. According to Lemma 4.13, Algorithm 2 can achieve an ϵ-absolute error for index building in time $O(\frac{\mathrm{Tr}((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})d^{max}_{\mathcal{V}_l}\Delta^2_{\mathcal{G}}|\mathcal{V}_l|\log n}{\epsilon^2})$. Similar to Algorithm 1, under our assumptions, $\mathrm{Tr}((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}) < \vec{1}^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1} = O(n\log n)$. Since in a scale-free graph, $m = O(n\log n)$, we can choose $\mathcal{V}_l$ such that $d^{max}_{\mathcal{V}_l} = O(\log n)$. Combined with $\Delta_{\mathcal{G}} = O(\log n)$, we can obtain that Algorithm 2 has a time complexity $O(\frac{n\log^4 n}{\epsilon^2}) = \widetilde{O}(n)$. This shows that the two index building algorithms can both achieve a near-linear time complexity on a real-world scale-free graph. □

[1] $\widetilde{O}(\cdot) = O(\cdot * \mathrm{poly}(\log n))$

LEMMA 4.19. *Suppose that $m = O(n\log n)$, $\frac{1}{1-\lambda_{\mathcal{V}_l}} = O(\log n)$, the query processing algorithms can achieve a sub-linear time complexity. Specifically,* RW *and* Push *have a query complexity* $O(\mathrm{poly}(\log n))$. *For* Bipush*, it is $\widetilde{O}(n^{\frac{2}{3}})$.*

PROOF. According to Lemma 4.14, the query complexity of RW to achieve an ϵ-absolute error for $r(s,t)$ is $O(\frac{\sigma^3_{\mathcal{V}_l}}{\epsilon^2})$. Based on our assumptions, we can derive that $\sigma_{\mathcal{V}_l} = O(\log n)$. Thus, the time complexity can be simplified as $O(\frac{\log^3 n}{\epsilon^2}) = O(\mathrm{poly}(\log n))$. According to Lemma 4.15, the query complexity of Push to achieve an ϵ-absolute error for $r(s,t)$ is $O(\frac{\bar{d}h^2_{\mathcal{V}_l}}{\epsilon})$. Consider the case that s,t are selected randomly, $h_{\mathcal{V}_l} = \bar{h}(\mathcal{V}_l) = O(\log n)$. On a scale-free graph where $m = O(n\log n)$, $\bar{d} = O(\log n)$. Thus, we can derive that the time complexity of Push is $O(\frac{\log^3 n}{\epsilon})$. Finally, according to Lemma 4.16, the query complexity of Bipush to achieve an ϵ-absolute error for $r(s,t)$ is $O(\frac{(n\bar{d}h_{\mathcal{V}_l})^{\frac{2}{3}}\sigma_{\mathcal{V}_l}}{\epsilon^{\frac{2}{3}}})$. Similarly, it can be simplified as $O(\frac{n^{\frac{2}{3}}(\log n)^{\frac{7}{3}}}{\epsilon^{\frac{2}{3}}}) = \widetilde{O}(n^{\frac{2}{3}})$. This shows that all the query algorithms can achieve a sublinear time complexity under our assumptions. □

Notice that although the proposed algorithms are sub-linear under our assumptions, these bounds are just worst-case theoretical bounds. The experimental evaluation (See Section 4.2) shows that they have better empirical performance than the existing algorithms [45, 67] which also can achieve a $O(\mathrm{poly}(\log n))$ bound under similar assumptions. Moreover, the experimental evaluation also shows that Bipush is much better than the other two algorithms RW and Push.

Heuristic landmark nodes set selection. Recall that the efficiency of our algorithms is closely related to the landmark set nodes. However, given a number k, selecting k landmarks lacks a unique criterion, as the performance of the proposed algorithms depends on different specific parameters. For example, as the time complexity of the single-pair index-building algorithm RsfIndex is related to the expected running time of the Wilson algorithm $O(\mathrm{Tr}(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})$. Thus, one possible objective is to minimize $O(\mathrm{Tr}(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})$ by choosing k nodes $\mathcal{V}_l$, thus minimizing the time of building index. This problem is known to be NP-hard, as shown in [28]. For another example, the query processing algorithm Push takes time complexity which is related to the hitting time from the query node to the landmark set $\mathcal{V}_l$. Thus, another possible objective is to choose k nodes $\mathcal{V}_l$ that minimize the average hitting time from all other nodes to the node set $\mathcal{V}_l$. This problem is also an NP-hard problem, as shown in [3]. However, solving these problems is not a necessity for our approach. Instead, we employ several heuristic selection strategies and observe that simple heuristics yield good performance. Intuitively, if every node in $\mathcal{U}$ can easily hit any one landmark node in $\mathcal{V}_l$ by a random walk, then our algorithms can be very efficient, because both the $\mathcal{V}_l$-absorbed random walk and $\mathcal{V}_l$-absorbed push can terminate very quickly in this case. Based on this intuition, we present two heuristic landmark node selection methods as follows. The first approach degree+ is based on the degree of the nodes. Specifically, we first sort the nodes in an non-increasing ordering by their degrees. Then, we add the highest-degree node into $\mathcal{V}_l$ and

delete all the neighbors of this selected node from the graph. In the remaining graph, we iteratively select the highest-degree nodes according this rule, until we get $|\mathcal{V}_l|$ nodes. Since we always select the highest-degree nodes, this method can often obtain a good landmark node set such that other nodes can easily hit the landmark nodes. Instead of using the degrees, our second method pagerank+ is based on the PageRank values. The selection procedure is same as the degree-based method, i.e., it iteratively selects the highest PageRank node from the remaining graph (when a node is selected, all of its neighbors will be deleted). Our experiments in Section 6.5 confirm that these methods are very effective in practice.

5 SINGLE-SOURCE ER COMPUTATION

A straightforward single-source ER computation approach is to invoke the single-pair algorithm for n times. However, even equipped with the index proposed in Section 4.1, such a straightforward method may still need to compute $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{ss}$, $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{st}$, $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{tt}$ for all $t \in \mathcal{U}$, which is very costly for large graphs.

Recall that for a given source node s, we can simultaneously compute $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{st}$ for all $t \in \mathcal{U}$ by invoking the $\mathcal{V}_l$-absorbed random walk or $\mathcal{V}_l$-absorbed push algorithms proposed in Section 4.2. Thus, for the single-source ER computation problem, the most challenging step is to compute $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{tt}$ for all $t \in \mathcal{U}$, i.e., the diagonal elements of the matrix $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$. Since the diagonal elements of $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$ is independent on the source node, a natural question is that can we pre-compute all such diagonal elements and maintain them as an index? A basic method to compute all diagonal elements requires to invoke the $\mathcal{V}_l$-absorbed random walk or $\mathcal{V}_l$-absorbed push algorithms for n times, which is costly for large graphs. In this section, we first propose two efficient Monte Carlo algorithms to estimate $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{tt}$ for all $t \in \mathcal{U}$. Together with the two index matrices $\widetilde{\mathbf{P}}_r$ and $\mathbf{L}_{\mathcal{H}}^{\dagger}$ constructed in Section 4.1, we obtain a new index, including all diagonal elements of $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$, for handling single-source ER query. Based on the new index, we then propose two efficient query processing algorithms for handling single-source ER queries.

5.1 Index Construction Algorithms

To construct our index, the key is to efficiently estimate all diagonal elements of $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$. To achieve this goal, we propose two novel algorithms based on the Wilson algorithm with a root set $\mathcal{V}_l$. We show that all diagonal elements of $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$ can be approximated at the same time when building the index matrices $\widetilde{\mathbf{P}}_f$ and $\mathbf{L}_{\mathcal{H}}^{\dagger}$ using Algorithm 2.

Index building by $\mathcal{V}_l$-rooted random spanning forest sampling. Recall that Algorithm 2 can obtain two index matrices $\widetilde{\mathbf{P}}_f$ and $\mathbf{L}_{\mathcal{H}}^{\dagger}$ by sampling a set of $\mathcal{V}_l$-rooted random spanning forests. Let $\widetilde{\mathcal{F}_{\mathcal{V}_l}}$ be the set of $\mathcal{V}_l$-rooted random spanning forests sampled by Algorithm 2. Interestingly, we find that all the diagonal elements of $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$ can also be estimated by $\widetilde{\mathcal{F}_{\mathcal{V}_l}}$. Specifically, we first establish a useful lemma which shows a close connection between the number of $\mathcal{V}_l$-rooted random spanning forests to the diagonal elements of $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$.

Let $\mathcal{F}_{\mathcal{V}_l}$ be the set of $\mathcal{V}_l$-rooted random spanning forests. Denote by $\mathcal{P}_{u\rightsquigarrow\mathcal{V}_l}$ the path from u to the node set $\mathcal{V}_l$. Here, a path from a node u to a set $\mathcal{V}_l$ means that the path comes from u and terminates at any one node in $\mathcal{V}_l$. Further, we let $\mathcal{F}_{\mathcal{V}_l:u\rightsquigarrow\mathcal{V}_l}^{(e_1,e_2)}$ be the set of $\mathcal{V}_l$-rooted random spanning forests that consist of all $F \in \mathcal{F}_{\mathcal{V}_l}$ and the unique path from node u to $\mathcal{V}_l$ passes through the edge (e_1, e_2). Then, we have the following results.

Algorithm 5: RsfIndex-SS

Input: Graph $\mathcal{G}$, landmark node set $\mathcal{V}_l$, sample size ω
Output: $\widetilde{\mathbf{P}}_f, \mathbf{L}_{\mathcal{H}}^{\dagger}, (\widetilde{\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}})_{uu}$ for each $u \in \mathcal{U}$

$\widetilde{\mathbf{P}}_f, \mathbf{L}_{\mathcal{H}}^{\dagger} \leftarrow \text{RsfIndex}(\mathcal{G}, \mathcal{V}_l, \omega)$;
Let $\widetilde{\mathcal{F}_{\mathcal{V}_l}}$ be the set of $\mathcal{V}_l$-rooted random spanning forest sampled by RsfIndex;
$(\widetilde{\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}})_{uu} \leftarrow 0$ for $u \in \mathcal{U}$;
Fix a path $\mathcal{P}_{u\rightsquigarrow\mathcal{V}_l}$ from node u to $\mathcal{V}_l$ for each $u \in \mathcal{U}$ in $\mathcal{G}$;
for *each* $F \in \widetilde{\mathcal{F}_{\mathcal{V}_l}}$ **do**
  for *each node* $u \in \mathcal{U}$ **do**
    Let $\mathcal{P}_{F:u\rightsquigarrow\mathcal{V}_l}$ denote the unique path from u to $\mathcal{V}_l$ in F;
    for *each edge* $(e_1, e_2) \in \mathcal{P}_{u\rightsquigarrow\mathcal{V}_l}$ **do**
      if $(e_1, e_2) \in \mathcal{P}_{F:u\rightsquigarrow\mathcal{V}_l}$ **then**
        $(\widetilde{\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}})_{uu} \leftarrow (\widetilde{\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}})_{uu} + \frac{1}{\omega}$
      if $(e_2, e_1) \in \mathcal{P}_{F:u\rightsquigarrow\mathcal{V}_l}$ **then**
        $(\widetilde{\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}})_{uu} \leftarrow (\widetilde{\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}})_{uu} - \frac{1}{\omega}$
return $\widetilde{\mathbf{P}}_f, \mathbf{L}_{\mathcal{H}}^{\dagger}, (\widetilde{\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}})_{uu}$ for each $u \in \mathcal{U}$

LEMMA 5.1.

$$(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{uu} = \sum_{(e_1,e_2)\in P_{u\rightsquigarrow\mathcal{V}_l}} \frac{|\mathcal{F}_{\mathcal{V}_l:u\rightsquigarrow\mathcal{V}_l}^{(e_1,e_2)}| - |\mathcal{F}_{\mathcal{V}_l:u\rightsquigarrow\mathcal{V}_l}^{(e_2,e_1)}|}{|\mathcal{F}_{\mathcal{V}_l}|},$$

where the sum is taking over edges in the path $P_{u\rightsquigarrow\mathcal{V}_l}$.

PROOF. According to the all-minors matrix forest theorem [17], we have: $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{ue_1} = \frac{|\mathcal{F}_{\mathcal{V}_l|u,e_1}|}{|\mathcal{F}_{\mathcal{V}_l}|}$, where $\mathcal{F}_{\mathcal{V}_l|u,e_1}$ denotes the set of spanning forests including all F that has $|\mathcal{V}_l| + 1$ connected components, with $|\mathcal{V}_l|$ connected components having a node $v_i \in \mathcal{V}_l$ as the root, and one connected component which includes u and e_1, with e_1 being the root. For each edge $(e_1, e_2) \in \mathcal{E}_{\mathcal{U}}$, $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{ue_1} - (\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{ue_2} = \frac{|\mathcal{F}_{\mathcal{V}_l|u,e_1}| - |\mathcal{F}_{\mathcal{V}_l|u,e_2}|}{|\mathcal{F}_{\mathcal{V}_l}|}$. Let $\mathcal{F}_{u,e_1|e_2,\mathcal{V}_l}$ denote the set of spanning forests with $|\mathcal{V}_l| + 1$ connected components, with $|\mathcal{V}_l|$ connected components having a node $v_i \in \mathcal{V}_l$ as the root, and one connected component which includes e_1 and u, with e_1 being the root. Specifically, e_1 and e_2 are in different connected components. Depending on whether e_1, e_2 are in the connected component of u, we can partition the spanning forests, which results in $|\mathcal{F}_{\mathcal{V}_l|u,e_1}| - |\mathcal{F}_{\mathcal{V}_l|u,e_2}| = |\mathcal{F}_{u,e_1|e_2,\mathcal{V}_l}| - |\mathcal{F}_{u,e_2|e_1,\mathcal{V}_l}|$. For each edge $(e_1, e_2) \in \mathcal{E}_{\mathcal{U}}$, there is a bijection between $\mathcal{F}_{\mathcal{V}_l:u\rightsquigarrow\mathcal{V}_l}^{(e_1,e_2)}$ and $\mathcal{F}_{u,e_1|e_2,\mathcal{V}_l}$, since if we remove the edge (e_1, e_2) from a spanning forest in $\mathcal{F}_{\mathcal{V}_l:u\rightsquigarrow\mathcal{V}_l}^{(e_1,e_2)}$, we will exactly obtain a spanning forest in $\mathcal{F}_{u,e_1|e_2,\mathcal{V}_l}$. If we delete the edge (e_1, e_2) from a spanning forest in $\mathcal{F}_{u,e_1|e_2,\mathcal{V}_l}$, we will exactly obtain a spanning forest in $\mathcal{F}_{\mathcal{V}_l:u\rightsquigarrow\mathcal{V}_l}^{(e_1,e_2)}$. Furthermore, we have $(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{uu} = \sum_{(e_1,e_2)\in\mathcal{P}_{u\rightsquigarrow\mathcal{V}_l}} (\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{ue_1} - (\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})_{ue_2}$. Thus, the Lemma is established. □

Based on Lemma 5.1, we can fix a set of paths from each node $u \in \mathcal{U}$ to $\mathcal{V}_l$. When sampling a spanning forest $F \in \widetilde{\mathcal{F}_{\mathcal{V}_l}}$, we examine whether the paths from u to $\mathcal{V}_l$ in F pass through the

Algorithm 6: LeWalkIndex-SS

Input: Graph $\mathcal{G}$, landmark node set $\mathcal{V}_l$, sample size ω
Output: $\widetilde{\mathbf{P}}_r$, $\mathbf{L}^{\dagger}_{\mathcal{H}}$, $(\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{uu}$ for each $u \in \mathcal{U}$

1 $\widetilde{\mathbf{P}}_f, \mathbf{L}^{\dagger}_{\mathcal{H}} \leftarrow$ RsfIndex$(\mathcal{G}, \mathcal{V}_l, \omega)$;
2 $(\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{uu} \leftarrow 0$ for $u \in \mathcal{U}$;
3 **for** $i = 1 : \omega$ **do**
4 In the loop-erased walk of the Wilson algorithm in the i-th iteration of RsfIndex, if a node is visited for n_u times, then $(\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{uu} \leftarrow (\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{uu} + \frac{n_u}{\omega}$;
5 **return** $\widetilde{\mathbf{P}}_f, \mathbf{L}^{\dagger}_{\mathcal{H}}, (\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{uu}$ for $u \in \mathcal{U}$;

edges in the pre-fixed paths to obtain an unbiased estimation of $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}$ for $u \in \mathcal{U}$. Note that this can be easily implemented by a depth-first search (DFS) procedure. Specifically, we start the DFS procedure by regarding the landmark node set $\mathcal{V}_l$ as a start node, and exploit the graph in a DFS manner. During the DFS procedure, we record the visit time and the finish time for each node in $\mathcal{U}$ that we visit [19]. After the DFS procedure terminates, we can examine whether the edge (e_1, e_2) passes the path $\mathcal{P}_{F:u\rightsquigarrow\mathcal{V}_l}$ by comparing the visit time and finish time of e_1 and u. The benefit of the implementation is that we can finish the DFS procedure in $O(m+n)$ time and examine each node in $O(1)$ time [19], which is very efficient. Combining with RsfIndex to approximate $\mathbf{P_f}$ and $(\mathbf{L}/\mathcal{V}_l)^{\dagger}$, the resulting index building algorithm is shown in Algorithm 5. In Lines 5-12, the algorithm estimates the diagonal elements by examining a set of fixed paths in a sampled spanning forest. When the algorithm terminates, all $\widetilde{\mathbf{P}}_f, \mathbf{L}^{\dagger}_{\mathcal{H}}, (\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{uu}$ for $u \in \mathcal{U}$ are stored as the index. The size of the index is still $O(|\mathcal{V}_l| \times n)$. The following theorem shows the accuracy and time complexity of Algorithm 5 to build our index.

LEMMA 5.2. *Algorithm 5 can output an estimation of* $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}$ *with absolute error* ϵ *for all* $u \in \mathcal{U}$ *with the time complexity* $O(\frac{\mathrm{Tr}((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}) d^{max}_{\mathcal{V}_l} \Delta^2_{\mathcal{G}} |\mathcal{V}_l| \log n}{\epsilon^2})$.

PROOF. Let X be the random variable added to $(\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{uu}$ in Algorithm 5, that is, for each edge in $\mathcal{P}_{u\rightsquigarrow\mathcal{V}_l}$, if the edge appears in the path from u to $\mathcal{V}_l$ in the sampled spanning forest F, then add $(\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{uu}$ by 1. If it appears in the opposite direction, then add $(\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{uu}$ by -1. For all such edges, since the path has at most $\Delta_{\mathcal{G}}$ edges, we have $|X| \le \Delta_{\mathcal{G}}$. Thus, according to Hoeffding bound, when $\omega \ge \frac{4\Delta^2_{\mathcal{G}} \log n}{\epsilon}$, the algorithm can achieve an ϵ absolute error for $(\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{uu}$ with high probability. Sampling the spanning forest costs time $\mathrm{Tr}((\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})$. The traverse operation in Line 5-12 costs time $|\mathcal{U}|\Delta_{\mathcal{G}}$. Thus, the total time complexity is dominated by Algorithm 2, which is $O(\frac{d^{max}_{\mathcal{V}_l} \Delta^2_{\mathcal{G}} \log n \, \mathrm{Tr}((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})}{\epsilon^2})$. The proof is completed. □

Index building by loop-erased random walk sampling. Note that Algorithm 5 requires matching the paths in the random spanning forests with a pre-fixed path, which introduces additional time overheads. Here we propose an alterative algorithm based on the loop-erased random walk (LERW) sampling. Below, we first present a novel loop-erased random walk interpretation of $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}$ for $u \in \mathcal{U}$, and based on this interpretation we can develop our algorithm.

LEMMA 5.3. *Let* $\tau^{\mathsf{LE}}_{\mathcal{V}_l}[u,u]$ *be the expected number of passes to a node* $u \in \mathcal{U}$ *in a loop-erased random walk with root set* $\mathcal{V}_l$. *Then, we have* $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu} = \frac{\tau^{\mathsf{LE}}_{\mathcal{V}_l}[u,u]}{d_u}$.

PROOF. In a loop-erased random walk with root set $\mathcal{V}_l$, the output distribution is independent on the node ordering according to the analysis in [65]. When setting the node u as the first node, the probability distribution of the $\mathcal{V}_l$-absorbed random walk starts from u and the loop-erased random walk with root set $\mathcal{V}_l$ are just the same. Note that when the first node u hits a node in $\mathcal{V}_l$, the loop-erased walk will never pass through u again. Thus, by Lemma 3.4, we have $\tau^{\mathsf{LE}}_{\mathcal{V}_l}[u,u] = \tau_{\mathcal{V}_l}[u,u] = (\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu} d_u$. As a result, the Lemma is established. □

The detailed description of this index-building algorithm is shown in Algorithm 6. According to Lemma 5.3, we can utilize the loop-erased random walks with root set $\mathcal{V}_l$ for sampling random spanning forests in Algorithm 2 to simultaneously estimate the diagonal elements $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}$ for all $u \in \mathcal{U}$. Specifically, by Lemma 5.3, we use the degree-normalized average number of visits to each node $u \in \mathcal{U}$ as an unbiased estimation of $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}$ for all $u \in \mathcal{U}$ (Line 4). Below, we analyze the accuracy and time complexity of Algorithm 6.

LEMMA 5.4. *Algorithm 6 can output an estimation of* $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}$ *for all* $u \in \mathcal{U}$ *in time* $O(\frac{(\sigma^2_{\mathcal{V}_l} + d^{max}_{\mathcal{V}_l}) \log n \, \mathrm{Tr}((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})}{\epsilon^2})$.

PROOF. Let X be the random variable which represents the number of visits to a node u in a $\mathcal{V}_l$-absorbed loop-erased walk. The distribution of the walk is identical to a v-absorbed walk. Thus, the distribution of the number of visits to a node u in a loop-erased walk is equal to the distribution of the number of passes to the node u in a $\mathcal{V}_l$-absorbed walk starts from u. According to the proof of Lemma 4.9, when the sample size satisfies $\omega \ge \frac{\sigma^2_{\mathcal{V}_l} \log n}{\epsilon^2}$, the algorithm can achieve an ϵ absolute error for $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}$. Here, the random walk steps can be computed via utilizing LERW to sample spanning forests, with little additional overhead. Combined with Lemma 4.6, the overall time complexity is $O(\frac{(\sigma^2_{\mathcal{V}_l} + d^{max}_{\mathcal{V}_l}) \log n \, \mathrm{Tr}((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})}{\epsilon^2})$, which establishes the lemma. □

Discussions. The sample size of Algorithm 5 and Algorithm 6 are $\frac{d^{max}_{\mathcal{V}_l} \Delta^2_{\mathcal{G}} |\mathcal{V}_l| \log n}{\epsilon^2}$ and $\frac{(\sigma^2_{\mathcal{V}_l} + d^{max}_{\mathcal{V}_l}) \log n}{\epsilon^2}$ respectively. As discussed before, $\Delta_{\mathcal{G}}$, $\sigma_{\mathcal{V}_l}$ are typically small numbers. We also find in experiments that our algorithms can achieve low errors with very small sample size. Note that in [31], there are also two index building algorithms based on sampling spanning trees and loop-erased walks, which takes $\mathrm{Tr}((\mathbf{I} - \mathbf{P}_v)^{-1})$ to draw a sample. The time complexity of our algorithm for drawing a sample $(\mathrm{Tr}((\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}))$ is strictly smaller than [31], since there are more nodes to hit. For example, on PowerGrid [27], $\mathrm{Tr}((\mathbf{I} - \mathbf{P}_v)^{-1}) = 4.3 \times 10^5$, while $\mathrm{Tr}((\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}) = 1.5 \times 10^5$ when $\mathcal{V}_l$ is the top-100 highest degree nodes. We will also show significantly empirical improvement of our index building algorithms compared to [31].

Algorithm 7: The single-source ER query processing algorithm

Input: Graph $\mathcal{G}$, landmark set $\mathcal{V}_l$, a source node s, $\widetilde{\mathbf{P}}_f$, $\mathcal{L}^{\dagger}_{\mathcal{H}}$, $(\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{uu}$ for all $u \in \mathcal{U}$
Output: $\tilde{r}(s,u)$ for all $u \in \mathcal{V}$

1 **if** $s \in \mathcal{V}_l$ **then**
2 **for** *each node* $u \in \mathcal{U}$ **do**
3 $\tilde{r}(s,u) \leftarrow (\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{uu} + (\mathbf{p}_s - \mathbf{e}_u)^T \mathcal{L}^{\dagger}_{\mathcal{H}}(\mathbf{p}_s - \mathbf{e}_u)$;
4 **for** *each node* $u \in \mathcal{V}_l$ **do**
5 $\tilde{r}(s,u) \leftarrow (\mathbf{e}_s - \mathbf{e}_u)^T \mathcal{L}^{\dagger}_{\mathcal{H}}(\mathbf{e}_s - \mathbf{e}_u)$;
6 **else if** $s \in \mathcal{U}$ **then**
7 Invoke the $\mathcal{V}_l$-absorbed random walk sampling or the $\mathcal{V}_l$-absorbed push algorithms to obtain an estimation $(\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{st}$ for all $t \in \mathcal{V}$;
8 **for** *each node* $u \in \mathcal{U}$ **do**
9 $\tilde{r}(s,u) \leftarrow (\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{ss} + (\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{uu} - 2(\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{su} + (\mathbf{p}_s - \mathbf{p}_u)^T \mathcal{L}^{\dagger}_{\mathcal{H}}(\mathbf{p}_s - \mathbf{p}_u)$;
10 **for** *each node* $u \in \mathcal{V}_l$ **do**
11 $\tilde{r}(s,u) \leftarrow (\widetilde{\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}})_{uu} + (\mathbf{p}_s - \mathbf{e}_u)^T \mathcal{L}^{\dagger}_{\mathcal{H}}(\mathbf{p}_s - \mathbf{e}_u)$;
12 **return** $\tilde{r}(s,u)$ for $u \in \mathcal{V}$;

5.2 Query processing algorithms

Armed with our index, we propose two efficient query processing algorithms based on $\mathcal{V}_l$-absorbed walk sampling and $\mathcal{V}_l$-absorbed push respectively. Algorithm 7 shows the detailed framework of our algorithm. If $s \in \mathcal{V}_l$ (Lines 1-5), there are two cases to compute $r(s,u)$ by Lemma 3.7. For $u \in \mathcal{U}$, $r(s,u) = (\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu} + (\mathbf{p}_s - \mathbf{e}_u)^T \mathcal{L}^{\dagger}_{\mathcal{H}}(\mathbf{p}_s - \mathbf{e}_u)$ (Lines 2-3). Since all the diagonal element $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}$, the probability vector $\mathbf{p}_s$ and the pseudo-inverse $\mathbf{L}^{\dagger}_{\mathcal{H}}$ can be directly obtained from the index, the computation is very fast. For $u \in \mathcal{V}_l$, two nodes are both located in the landmark node set, $r(s,u) = (\mathbf{e}_s - \mathbf{e}_u)^T \mathcal{L}^{\dagger}_{\mathcal{H}}(\mathbf{e}_s - \mathbf{e}_u)$ (Lines 4-5). Again, all the elements can be directly obtained from the index. When $s \in \mathcal{U}$, the situation is more complicated (Lines 6-11). According to Lemma 3.3, there are also two cases to compute $r(s,u)$. For $u \in \mathcal{U}$, both of the two nodes are located in the node set $\mathcal{U}$, we have $r(s,u) = (\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{ss} + (\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu} - 2(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{su} + (\mathbf{p}_s - \mathbf{e}_u)^T \mathcal{L}^{\dagger}_{\mathcal{H}}(\mathbf{p}_s - \mathbf{e}_u)$. Except the elements $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{su}$ for all $u \in \mathcal{U}$, all other elements can be directly obtained from the index. Thus, we utilize $\mathcal{V}_l$-absorbed random walk sampling or $\mathcal{V}_l$-absorbed push to estimate a column of the matrix $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}$ (Line 7). For convenience, we refer to these algorithms as RW-SS and Push-SS, respectively. For $u \in \mathcal{V}_l$, $r(s,u) = (\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu} + (\mathbf{p}_s - \mathbf{e}_u)^T \mathcal{L}^{\dagger}_{\mathcal{H}}(\mathbf{p}_s - \mathbf{e}_u)$ (Line 10-11). It can also be directly computed based on the index. Combining all these procedures together, it shows that with an $O(n \times |\mathcal{V}_l|)$ size index, we can answer a single-source ER query in time of answering a single-pair ER query. Below, we present a detailed theoretical analysis of our algorithms.

Theoretical analysis of the algorithms. Based on the above results, we can give the query time complexity for RW-SS and Push-SS to output an estimation of $r(s,u)$ with an ϵ-absolute error for all $u \in \mathcal{U}$, equipped with the index we described before.

LEMMA 5.5. *The query time complexity of* RW-SS *is* $O(\frac{\sigma^3_{\mathcal{V}_l}}{\epsilon^2})$ *for an* ϵ*-estimation of* $r(s,u)$ *for all* $u \in \mathcal{V}$.

PROOF. Based on Lemma 4.13, with the time complexity give therein, the error of index building is smaller than ϵ. According to Lemma 5.2 and Lemma 5.4, the diagonal elements are also estimated within an ϵ-absolute error. The results of Lemma 4.8 show that we can estimate a row of $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}$ simultaneously in time $O(\frac{\sigma^3_{\mathcal{V}_l}}{\epsilon^2})$. Thus, the proof is established. □

LEMMA 5.6. *Let* $h^{max}_{\mathcal{V}_l}(t)$ *be the maximum value of hitting time from all each nodes* $u \in \mathcal{U}$ *to* $\mathcal{V}_l$*, the query time complexity of* Push-SS *is* $O(\frac{\bar{d}h(s,\mathcal{V}_l)h^{max}_{\mathcal{V}_l}(t)}{\epsilon})$ *for an* ϵ*-estimation of* $r(s,u)$ *for all* $u \in \mathcal{V}$.

PROOF. Based on Lemma 4.13, with the time complexity give therein, the error of index building is smaller than ϵ. According to Lemma 5.2 and Lemma 5.4, the diagonal elements are also estimated within an ϵ-absolute error. Let $h^{max}_{\mathcal{V}_l}(t)$ denote the maximum value of hitting time from all each nodes $u \in \mathcal{U}$ to $\mathcal{V}_l$, the results of Lemma 4.9 show that we can estimate a row of $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}$ simultaneously in time $O(\frac{\bar{d}h(s,\mathcal{V}_l)h^{max}_{\mathcal{V}_l}(t)}{\epsilon})$. Thus, the proof is established. □

Based on the discussions before, the query time complexity of our algorithms is strictly better compared to the single-source query algorithms in [31]. Specifically, the query complexity RW-SS algorithm in [31] is $O(\frac{\sigma^3_v}{\epsilon^2})$ based on our analysis, and the Push-SS algorithm is $O(\frac{\bar{d}h(s,v)h^{max}_v(t)}{\epsilon})$. We improve the σ_v term into $\sigma_{\mathcal{V}_l}$ and the $h(s,v)$, $h^{max}_v(t)$ terms into $h(s,\mathcal{V}_l)$, $h^{max}_{\mathcal{V}_l}(t)$. Similar to the single-pair algorithms, RW-SS and Push-SS perform much better than such worst-case bounds in real-life graphs.

Discussions. Similar to single-pair algorithms, we can also provide bounds w.r.t. n for the single-source algorithms under the same assumptions. The results are also similar:

LEMMA 5.7. *Suppose that* $\frac{1}{1-\lambda_{\mathcal{V}_l}} = O(\log n)$, $m = O(n \log n)$ *and* $\Delta_{\mathcal{G}} = O(\log n)$*, the index building algorithms* RsfIndex-SS *(Algorithm 5) and* LeWalkIndex-SS *(Algorithm 6) can both achieve a near-linear time complexity* $\widetilde{O}(n)$.

PROOF. According to Lemma 5.2, Algorithm 5 can output an estimation of $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}$ with absolute error ϵ for all $u \in \mathcal{U}$ with the time complexity $O(\frac{\mathrm{Tr}((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}) d^{max}_{\mathcal{V}_l} \Delta^2_{\mathcal{G}} |\mathcal{V}_l| \log n}{\epsilon^2})$. Under our assumptions, $\mathrm{Tr}((\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}) = O(n \log n)$, $d^{max}_{\mathcal{V}_l} = O(\log n)$, $\Delta_{\mathcal{G}} = O(\log n)$. $|\mathcal{V}_l|$ is a small number. Thus, it can be simplified as $O(\frac{n \log^5 n}{\epsilon^2}) = \widetilde{O}(n)$. According to Lemma 5.4, the time complexity of Algorithm 6 is $O(\frac{(\sigma^2_{\mathcal{V}_l} + d^{max}_{\mathcal{V}_l}) \log n \, \mathrm{Tr}((\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1})}{\epsilon^2})$. Under our assumptions, $\sigma_{\mathcal{V}_l} = O(\log n)$, $d^{max}_{\mathcal{V}_l} = O(\log n)$, $\mathrm{Tr}((\mathbf{I} - \mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}) = O(n \log n)$. Thus, it can be simplified as $O(\frac{n \log^4 n}{\epsilon^2}) = \widetilde{O}(n)$. Combined with previous results, the single-source index building algorithms can also achieve a near-linear time complexity. □

LEMMA 5.8. *The query algorithms* RW-SS *can achieve a sublinear time complexity* $O(\mathrm{poly}(\log n))$ *suppose that* $m = O(n \log n)$ *and* $\frac{1}{1-\lambda_{\mathcal{V}_l}} = O(\log n)$. *For* Push*, it is* $\widetilde{O}(n^{\frac{1}{2}})$.

PROOF. According to Lemma 5.5, the query time complexity of RW-SS is $O(\frac{\sigma^3_{\mathcal{V}_l}}{\epsilon^2})$ for an ϵ-estimation of $r(s,u)$ for all $u \in \mathcal{V}$.

Table 2: Datasets ($\bar{d}$: average degree; Δ_G: diameter of the graph)

Dataset	n	m	$\bar{d}$	Δ_G
PowerGrid	4,941	6,594	2.67	46
Email-enron	33,696	180,811	10.73	11
Road-PA	1,087,562	1,541,514	2.83	786
Road-CA	1,957,027	2,760,388	2.82	849
Orkut	3,072,441	117,185,083	76.28	9

Under our assumptions, $\sigma_{\mathcal{V}_l} = O(\log n)$. Thus, it can be simplified as $O(\frac{\log^3 n}{\epsilon^2}) = O(\text{poly}(\log n))$. According to Lemma 5.6, the query time complexity of Push-SS is $O(\frac{\bar{d}h(s,\mathcal{V}_l)h^{max}_{\mathcal{V}_l}(t)}{\epsilon})$ for an ϵ-estimation of $r(s,u)$ for all $u \in \mathcal{V}$. Under our assumptions, $\bar{d} = O(\log n)$, $h(s,\mathcal{V}_l) = O(\log n)$. However, the maximum hitting time $h^{max}_{\mathcal{V}_l}(t)$ cannot simply bounded by $O(\log n)$. For an arbitrary u, we have:

$$\begin{aligned} & h(u,\mathcal{V}_l) \\ &= \mathbf{e}_u^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\vec{1} \\ &= \sqrt{n}\mathbf{e}_u^T(\mathbf{I}-\mathbf{P}_{\mathcal{U}\mathcal{U}})^{-1}\frac{\vec{1}}{\sqrt{n}} \\ &= \sqrt{n}\mathbf{e}_u^T\sum_{k=0}^{\infty}\mathbf{P}^k_{\mathcal{U}\mathcal{U}}\frac{\vec{1}}{\sqrt{n}} \\ &= \sqrt{n}\mathbf{e}_u^T\sum_{k=0}^{\sigma_{\mathcal{V}_l}}\mathbf{P}^k_{\mathcal{U}\mathcal{U}}\frac{\vec{1}}{\sqrt{n}} + \sqrt{n}\mathbf{e}_u^T\sum_{k=\sigma_{\mathcal{V}_l}+1}^{\infty}\mathbf{P}^k_{\mathcal{U}\mathcal{U}}\frac{\vec{1}}{\sqrt{n}}, \end{aligned}$$

For the first term, for any $k \in [0,\sigma_{\mathcal{V}_l}]$, we have $\mathbf{e}_u^T\mathbf{P}^k_{\mathcal{U}\mathcal{U}}\frac{1}{\sqrt{n}} \leq 1$, since $\mathbf{P}_{\mathcal{U}\mathcal{U}}$ is a submatrix of a probability transition matrix. Thus, the first term is $O(\log n)$. For the second term, since $\sigma_{\mathcal{V}_l} = O(\frac{\log(1/(1-\lambda_{\mathcal{V}_l}))}{1/\lambda_{\mathcal{V}_l}})$, we have:

$$\begin{aligned} \mathbf{e}_u^T\sum_{k=\sigma_{\mathcal{V}_l}+1}^{\infty}\mathbf{P}^k_{\mathcal{U}\mathcal{U}}\frac{\vec{1}}{\sqrt{n}} &= \|\mathbf{e}_u^T\sum_{k=\sigma+1}^{\infty}\mathbf{P}^k_{\mathcal{U}\mathcal{U}}(\frac{\vec{1}}{\sqrt{n}})\|_2 \\ &\leq \|\mathbf{e}_u\|_2\frac{\lambda^{\sigma+1}_{\mathcal{V}_l}}{1-\lambda_{\mathcal{V}_l}}\|(\frac{1}{\sqrt{n}})\|_2 \\ &= \frac{\lambda^{\sigma+1}_{\mathcal{V}_l}}{1-\lambda_{\mathcal{V}_l}} = O(1), \end{aligned}$$

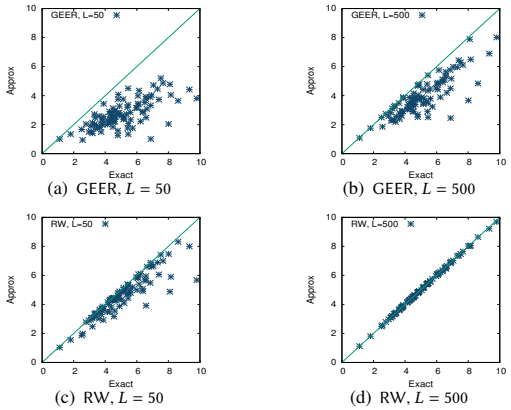
Thus, we can derive that $h(u,\mathcal{V}_l) = O(n^{\frac{1}{2}}\log n)$. Combined with Lemma 5.6, the query time complexity of Push-SS is $O(\frac{n^{\frac{1}{2}}\log^3 n}{\epsilon}) = \widetilde{O}(n^{\frac{1}{2}})$. As a result, the single-source query algorithms can also achieve a sub-linear time complexity. □

These results show that when ignoring the time of outputting the results (which takes $O(n)$ time), it only requires a sub-linear time complexity to answer a single-source query, which is very efficient.

6 EXPERIMENTS

6.1 Experimental Settings

Datasets. We employ five real-world datasets encompassing various types of graphs, mainly including social networks and road networks. The detailed statistics of the datasets is illustrated in Table 2. Among the five datasets, Email-enron and Orkut are social networks. Road-PA and Road-CA are road networks of America. For PowerGrid, it is an infrastructure network, which we find its statistics similar to the road networks. Road networks and social networks have very different structures. It is common to observe a comparatively smaller average degree combined with a larger diameter on road networks. The algorithm behavior is also different on the two types of networks. For single-pair query, we randomly generate 100 node pairs as the query set and report the average results. For single-source query, we randomly generate 50 source nodes as the query set and report the average performance. We can obtain ground truth for both single-pair and single-source effective resistance by applying deterministic eigen-decomposition to the small datasets. On large graphs, we obtain the ground-truth results for single-pair query by applying the state-of-the-art deterministic algorithm Push-v [31] with $r_{max} = 10^{-9}$ following [31, 67]. We apply LEwalk in [31] with $\omega = 10^6$ to obtain the ground-truth results for single-source query following [31] on large datasets.

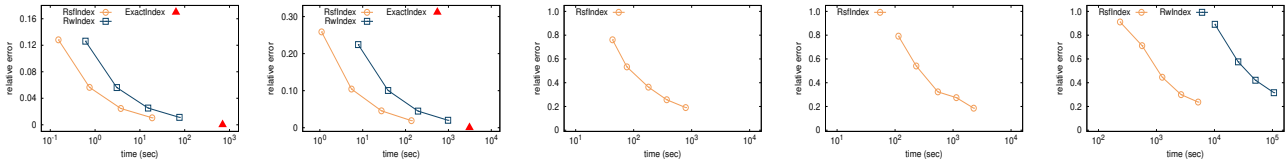


(a) GEER, $L = 50$ (b) GEER, $L = 500$
(c) RW, $L = 50$ (d) RW, $L = 500$

Figure 3: The estimation quality of different single-pair query processing algorithms

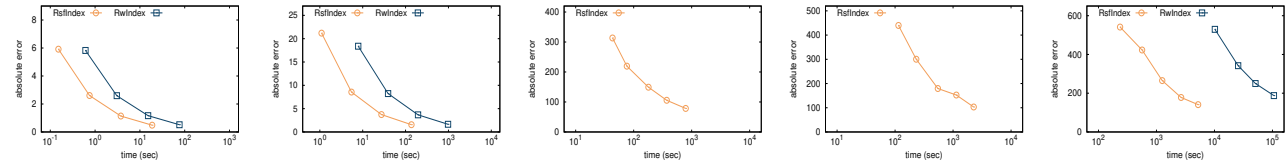
Different algorithms. For the index building algorithm, we implement two index building algorithms RwIndex (Algorithm 1) and RsfIndex (Algorithm 2). To support single-source query, we also implement two extended algorithms based on RsfIndex named RsfIndex-SS (Algorithm 5) and LeWalkIndex-SS (Algorithm 6). We compare the two single-source index building algorithms with the state-of-the-art index building algorithm UST and LEwalk proposed in [31] which are special cases of our index building algorithms when $|\mathcal{V}_l| = 1$. For single-pair query processing algorithms, we implement three algorithms RW, Push and Bipush which is described in Algorithm 4. We compare the three query processing algorithms with four state-of-the-art algorithms, GEER [67] and the single landmark node special cases RW-v, Push-v and Bipush-v proposed in [31]. For single-source query, we implement two query processing algorithms RW-SS and Push-SS which are described in Algorithm 7. We compare them with the single landmark node special cases RW-SS-v and Push-SS-v proposed in [31].

Different algorithms. For the index building algorithm, we implement two algorithms RwIndex (Algorithm 1) and RsfIndex (Algorithm 2). We have also implemented an exact method ExactIndex



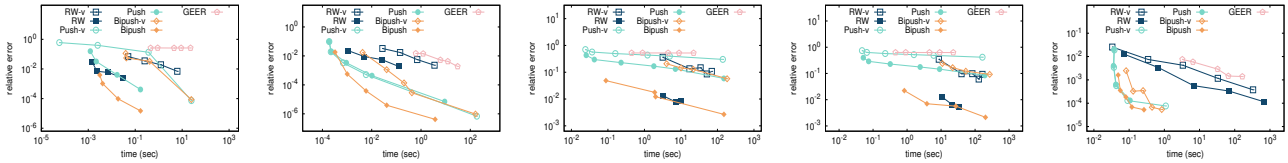
(a) PowerGrid (b) Email-enron (c) Road-PA (d) Road-CA (e) Orkut

Figure 4: Performance of different index building algorithms for single-pair ER query



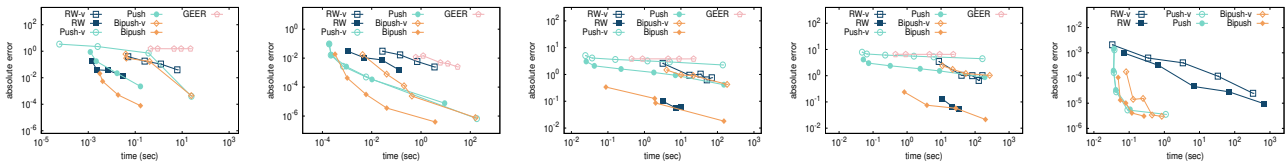
(a) PowerGrid (b) Email-enron (c) Road-PA (d) Road-CA (e) Orkut

Figure 5: Performance of different index building algorithms for single-pair ER query (absolute error)



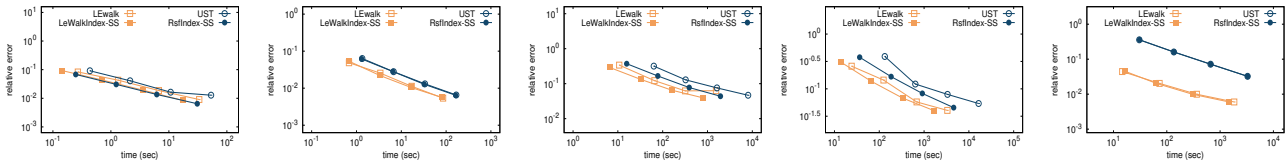
(a) PowerGrid (b) Email-enron (c) Road-PA (d) Road-CA (e) Orkut

Figure 6: Performance of different query processing algorithms for single-pair ER query



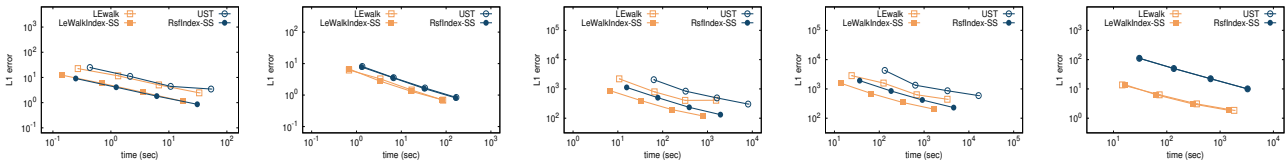
(a) PowerGrid (b) Email-enron (c) Road-PA (d) Road-CA (e) Orkut

Figure 7: Performance of different query processing algorithms for single-pair ER query (absolute error)



(a) PowerGrid (b) Email-enron (c) Road-PA (d) Road-CA (e) Orkut

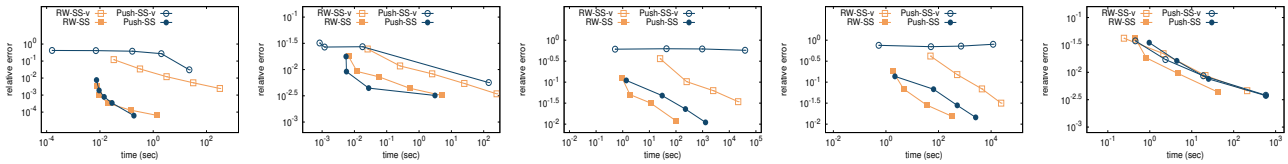
Figure 8: Performance of different index building algorithms for single-source ER query



(a) PowerGrid (b) Email-enron (c) Road-PA (d) Road-CA (e) Orkut

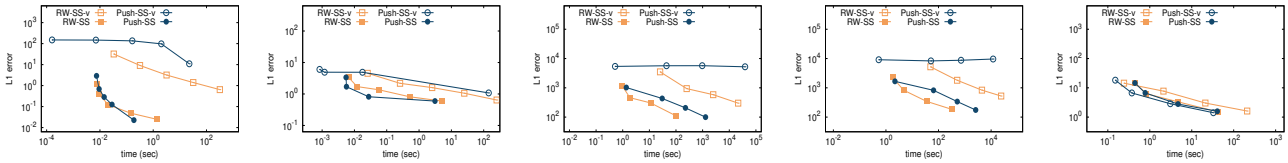
Figure 9: Performance of different index building algorithms for single-source ER query ($L1$-error)

to build the index by exactly solving $|\mathcal{U}|$ linear systems. To support single-source query, we also implement two extended algorithms based on RsfIndex named RsfIndex-SS (Algorithm 5) and LeWalkIndex-SS (Algorithm 6). We compare the two single-source



(a) PowerGrid (b) Email-enron (c) Road-PA (d) Road-CA (e) Orkut

Figure 10: Performance of different query processing algorithms for single-source ER query

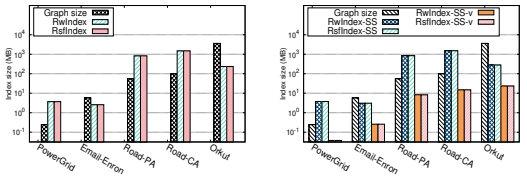


(a) PowerGrid (b) Email-enron (c) Road-PA (d) Road-CA (e) Orkut

Figure 11: Performance of different query processing algorithms for single-source ER query ($L1$-error)

index building algorithms with the state-of-the-art index building algorithm UST and LEwalk proposed in [31] which are special cases of our index building algorithms when $|\mathcal{V}_l| = 1$. For single-pair query processing algorithms, we implement three algorithms RW, Push and Bipush which is described in Algorithm 4. We compare the three query processing algorithms with four state-of-the-art algorithms, GEER [67] and RW-v, Push-v and Bipush-v proposed in [31], which are the special cases of our algorithms with only one landmark node. For single-source query, we implement two query processing algorithms RW-SS and Push-SS which are described in Algorithm 7. We compare them with the state-of-the-art algorithms RW-SS-v and Push-SS-v proposed in [31], which are the special cases of our algorithms with one landmark node.

Parameters. For all the proposed algorithms including the index building algorithms and query processing algorithms for both single-pair query and single-source query, we have shown the sample size needed and the total time complexity for the algorithms to achieve an ϵ-estimation of ER. However, the bounds are just worst-case bounds for comparing the algorithms theoretically. There will be too much unnecessary work if we use these bounds to determine sample sizes. Thus, following [31, 45, 67], we compare the algorithms by carefully setting the sample size ω and the error threshold r_{max} to make the algorithms run in a reasonable time. Specifically, for the single-pair index building algorithms, we set $\omega = 10^4$ by default for both RwIndex and RsfIndex. For the single-pair query processing algorithms, we set $\omega = 10^4$ for RW, $r_{max} = 10^{-4}$ for Push, and $\omega = 10^3$, $r_{max} = 10^{-3}$ for Bipush by default. We also vary the sample size ω and the error threshold r_{max} to compare the empirical errors. For the compared algorithms, there is one parameter ϵ which controls the accuracy for GEER, we set ϵ as 0.01 by default following [67]. For the algorithms proposed in [31], we set $\omega = 10^4$ for RW-v, $r_{max} = 10^{-4}$ for Push-v, and $\omega = 10^{-3}$, $r_{max} = 10^{-3}$ for Bipush-v following [31]. For the single-source index building algorithms, we set $\omega = 10^4$ by default for RsfIndex-SS and LeWalkIndex-SS. We also set $\omega = 10^4$ for UST and LEwalk following [31]. For single-source query processing algorithms, we set $\omega = 10^4$ for RW-SS and $r_{max} = 10^{-4}$ for Push-SS by default. We also set $\omega = 10^4$



(a) single-pair index (b) single-source index

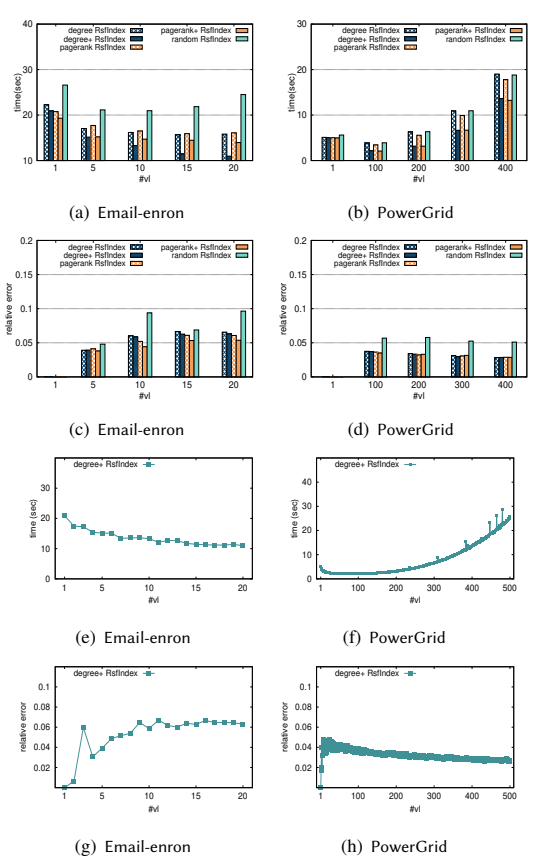
Figure 12: Index space consumption of different algorithms

for RW-SS-v and $r_{max} = 10^{-4}$ for Push-SS-v following [31]. For the landmark node set $\mathcal{V}_l$, we set $|\mathcal{V}_l| = 10$ in social networks and $|\mathcal{V}_l| = 100$ in road networks and use degree+ to select the landmark node set in all our algorithms. We also vary $|\mathcal{V}_l|$ and different landmark selection methods to compare the performance in Section 6.5. For the query processing algorithms, we use the index built by RsfIndex for single-pair query and the index built by LeWalkIndex-SS for single-source query by default. We also evaluate the performance of the query processing algorithms with the index built by RwIndex and RsfIndex-SS in Section 6.5.

Experimental environment. All the proposed algorithms are implemented in C++ and run on a Linux 20.04 server with Intel 2.0 GHz CPU and 128GB memory. For the compared methods, we use their original C++ implementations in [9, 31, 67]. All the implementations are complied using GCC9.3.0 with -O3 optimization.

6.2 Limitations of the existing studies

Recall that a major limitation of the existing random walk-based approaches is that it requires a large random walk length L to achieve a small absolute error. In this experiment, we demonstrate this phenomenon for two random walk-based single-pair query algorithms GEER (the state-of-the-art algorithm) and RW (the proposed algorithm). In both algorithms, we set L as the maximum random walk length and set the sample size as 1000. We conduct experiments on PowerGrid. The results of other datasets are consistent. Specifically, we randomly select 100 query node pairs and plot the results of approximate value v.s. the exact value. As shown in Fig. 3(a), when setting $L = 50$ in GEER, the estimation quality of the results is poor,



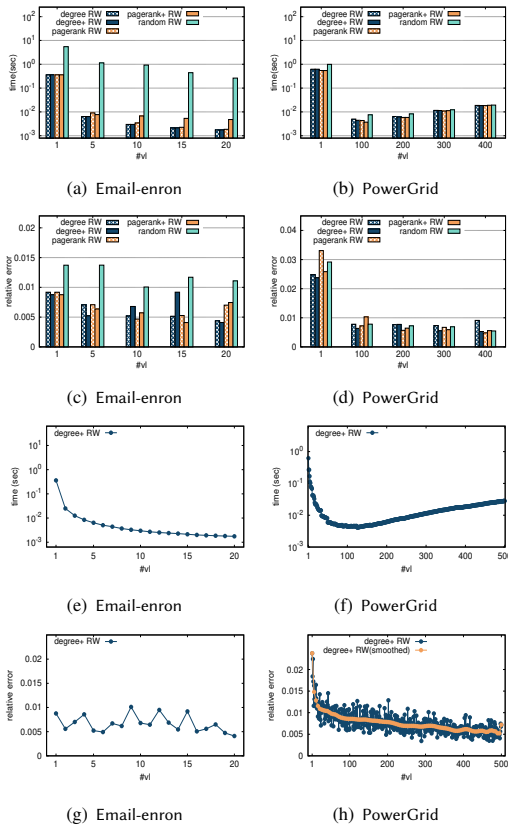
(a) Email-enron (b) PowerGrid

(c) Email-enron (d) PowerGrid

(e) Email-enron (f) PowerGrid

(g) Email-enron (h) PowerGrid

Figure 13: Effect of the choice of $\mathcal{V}_l$ for single-pair index building algorithms



(a) Email-enron (b) PowerGrid

(c) Email-enron (d) PowerGrid

(e) Email-enron (f) PowerGrid

(g) Email-enron (h) PowerGrid

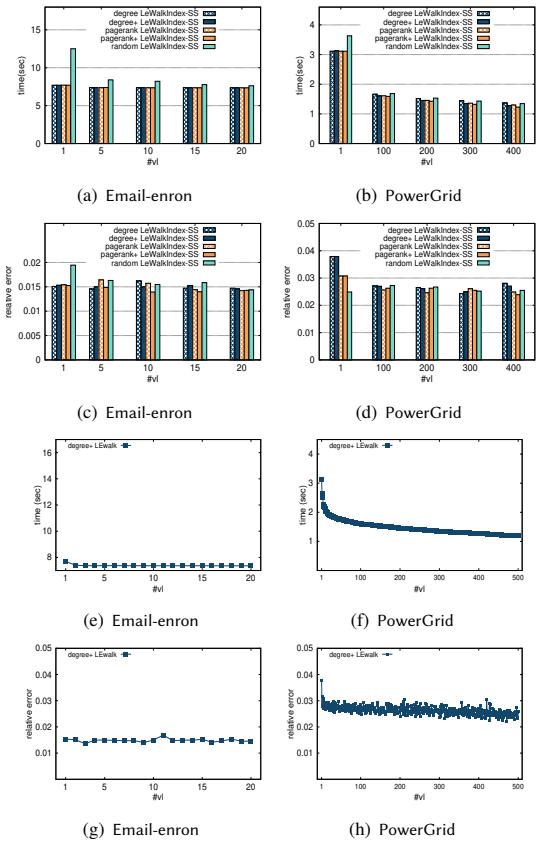
Figure 14: Effect of the choice of $\mathcal{V}_l$ for single-pair query processing algorithms

especially for the query node pair with large exact $r(s,t)$ value. For example, for queries with $r(s,t)$ as large as 10, the estimation results cannot exceed 6 in Fig. 3(a). Notice that an accurate approximation of a large $r(s,t)$ value is often highly demanded in applications such as clustering [51] and robust routing [53]. When $L = 500$, the results become better (Fig. 3(b)). However, in both of the cases, our algorithm RW can produce significantly better results, as shown in Fig. 3(c) and Fig. 3(d). This implies that our query algorithms demand a much smaller L to achieve a small absolute error, which leads to high efficiency.

6.3 Results of Single-pair ER Computation

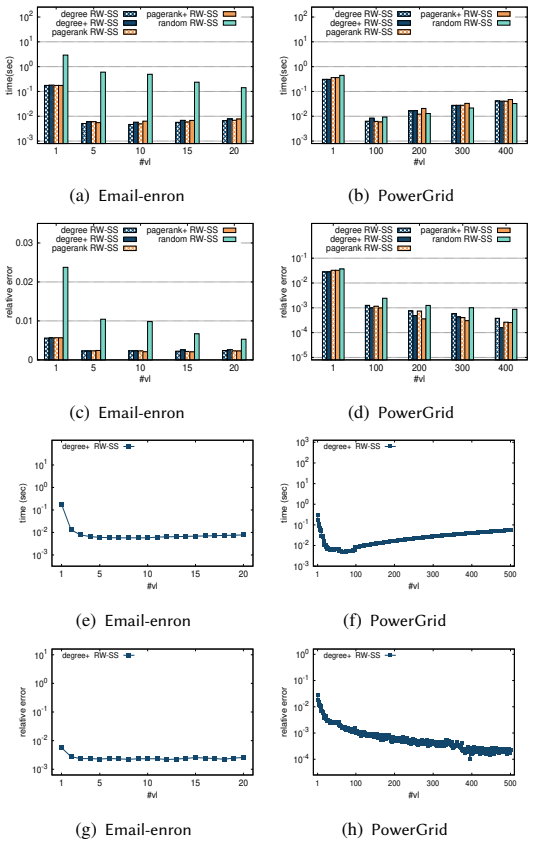
We first study the performance of the algorithms for single-pair effective resistance computation.

Evaluation of index building algorithms. We evaluate the index building time and the index size for different single-pair index building algorithms. We use the relative error of the norm of the related matrices to evaluate the index building quality. Suppose that $\widetilde{\mathbf{P}}$ and $\mathbf{L}^{\dagger}_{\mathcal{H}}$ is the estimated matrices output by RwIndex and RsfIndex, the relative error is defined as $\frac{\|\widetilde{\mathbf{P}}-\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}\|_F}{\|\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}\mathbf{L}_{\mathcal{U}\mathcal{V}_l}\|_F} + \frac{\|\mathbf{L}^{\dagger}_{\mathcal{H}}-(\mathbf{L}/\mathcal{V}_l)^{\dagger}\|_F}{\|\mathbf{L}^{\dagger}_{\mathcal{H}}\|_F}$, where $\|\cdot\|_F$ is the Frobenius norm of a matrix. We also evaluate the error by the absolute error of the Frobenius norm. We vary the parameters of RwIndex and RsfIndex and plot the running time and empirical error trade-offs. For the exact method ExactIndex, as the error is 0, we use a triangle to depict the running time. The results are shown in Fig. 4. As can be seen, both RwIndex and RsfIndex can approximately build the index with a small relative error, and much faster than the exact method ExactIndex. For example, on PowerGrid, to achieve a relative error 0.01, RsfIndex takes 18 seconds, RwIndex takes 72 seconds while the exact method ExactIndex needs 706 seconds. Notice that on large networks Orkut, Road-PA and Road-CA, ExactIndex cannot output a result within 24 hours. Thus, we omit the results of ExactIndex in Fig. 4(c)-(e). On two large road networks Road-PA and Road-CA, RwIndex cannot output a result within 24 hours. Thus, we omit the results of RwIndex. Specifically, RsfIndex is much faster than RwIndex. For example, on Orkut, it takes 2.7×10^3 secs for RsfIndex to build the index with a relative error 0.3, while it takes 1.06×10^5 secs for RwIndex, which is more than 2 orders of magnitude slower. The reason is that RsfIndex uses loop-erased walks which are more efficient than sampling $|\mathcal{U}|$ $\mathcal{V}_l$-absorbed random walks. The index size of different index building algorithms is illustrated in Fig. 12(a). As can be seen, the index size of RwIndex and RsfIndex is smaller than the graph size on social networks, and the index size of RwIndex and RsfIndex is larger than the graph size on road graphs. Recall that the space complexity of the index is $O(|\mathcal{V}_l| \times n)$. However, even on the large road network Road-CA, it consumes only 1493 MB space to store the index when $|\mathcal{V}_l| = 100$. Thus, the index size is acceptable and can be easily loaded in memory.



(a) Email-enron (b) PowerGrid (c) Email-enron (d) PowerGrid (e) Email-enron (f) PowerGrid (g) Email-enron (h) PowerGrid

Figure 15: Effect of the choice of $\mathcal{V}_l$ for single-source index building algorithms



(a) Email-enron (b) PowerGrid (c) Email-enron (d) PowerGrid (e) Email-enron (f) PowerGrid (g) Email-enron (h) PowerGrid

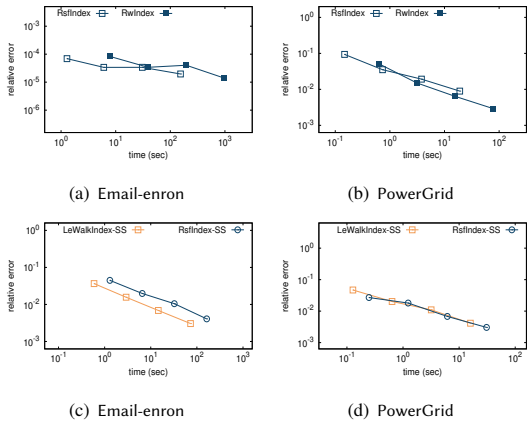
Figure 16: Effect of the choice of $\mathcal{V}_l$ for single-source query processing algorithms

Evaluation of query processing algorithms. We compare the proposed single-pair query processing algorithms RW, Push, Bipush with four SOTA methods, the RW-v, Push-v, Bipush-v algorithm proposed in [31], which is the special cases of our algorithms, as well as GEER proposed in [67]. For all the proposed algorithms, we use the index that we build with the algorithm RsfIndex in a similar degree of accuracy. We vary the parameters of the algorithms and plot the empirical error and query time trade-offs. Specifically, let $\hat{r}(s,t)$ be the estimated ER value, we evaluate the errors by the relative error defined as $\frac{|\hat{r}(s,t)-r(s,t)|}{r(s,t)}$. We also evaluate the errors by the absolute error defined as $|\hat{r}(s,t) - r(s,t)|$. The results are illustrated in Fig. 6 and Fig. 7. We can find that on all five datasets, the query processing algorithms RW, Push and Bipush all improve significantly over their single landmark node special cases RW-v, Push-v and Bipush-v. They can also achieve lower errors compared to the SOTA method GEER. This demonstrates the effectiveness of the multiple landmark nodes approaches. Among these algorithms, Bipush obtains the best performance, especially on road networks. For example, on Road-PA, Bipush can achieve a relative error 0.05 in 0.09 secs, while the fastest SOTA method Bipush-v takes 184 secs. Thus, our algorithm is up to 4-orders of magnitude faster than the SOTA methods. Push performs well on social networks, while RW performs better on road networks. This is perhaps because, on road networks, the hitting time from a node u to the landmark node set $\mathcal{V}_l$ is relatively larger, which makes the push operation inefficient. In such cases, random walks exploit the graph more quickly. However, the Bipush algorithm takes advantage of both algorithms, which performs well on all the datasets.

6.4 Results of single-source ER Computation

Next, we study the performance of the algorithms for single-source effective resistance computation.

Evaluation of index building algorithms. First, we evaluate the performance of the index building time algorithms. Notice that the performance of RsfIndex-SS and LeWalkIndex-SS for estimating the matrix $\widetilde{\mathbf{P}}$ and the pseudo-inverse $\mathbf{L}^{\dagger}_{\mathcal{H}}$ is totally the same as RsfIndex. Thus, we focus on evaluating the performance of estimating the diagonal of $\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}}$. Suppose that $\widetilde{(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}}$ is the estimation value of $(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}$, we evaluate the errors by the maximum relative error, which is defined as $\max_{u\in\mathcal{U}}\{\frac{|\widetilde{(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}}-(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}|}{(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}}\}$. We also evaluate the errors by the $L1$-error, which is defined as $\sum_{u\in\mathcal{U}}|\widetilde{(\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}} - (\mathbf{L}^{-1}_{\mathcal{U}\mathcal{U}})_{uu}|$. For the compared methods UST and LEwalk, we also evaluate the errors by the maximum relative error and $L1$-error of the diagonal vector. We vary the parameters of all the algorithms and plot the error and time trade-offs. The results are illustrated in Fig. 8 and



(a) Email-enron (b) PowerGrid

(c) Email-enron (d) PowerGrid

Figure 17: Effect of the accuracy of index for different query processing algorithms

Fig. 9. As can be seen, both RsfIndex-SS and LeWalkIndex-SS can achieve similar errors with the same running time compared to UST and LEwalk on social networks. They are slightly faster than UST and LEwalk on road networks. RsfIndex-SS is the best algorithm. For example, on Road-PA, it takes 65 secs for LEwalk to achieve a relative error 0.12, while it takes 32 secs for LeWalkIndex-SS. Thus, our algorithm only takes 50% time. The index size of different index building algorithms is also illustrated in Fig 12(b). As can be seen, the index size of RsfIndex-SS and LeWalkIndex-SS is similar to the index size of RwIndex and RsfIndex. Recall that the space complexity is still $O(n \times |\mathcal{V}_l|)$. On the largest road network Road-PA, it takes 1523 MB memory. Thus, the index size is still acceptable on large graphs.

Evaluation of query processing algorithms. We also study the performance of the query processing algorithms. We compare RW-SS and Push-SS with the SOTA methods RW-SS-v and Push-SS-v. We vary the parameters and plot the empirical error v.s. running time trade-offs. We evaluate the errors by the maximum relative error, we also evaluate the errors by the $L1$-error. The results are illustrated in Fig. 10 and Fig. 11. We can find that RW-SS and Push-SS can both answer a single-source query much faster than RW-SS-v and Push-SS-v. For example, on Road-PA, it takes 30 secs for RW-SS to achieve a relative error 0.05, while it takes 2×10^4 secs. Thus, our algorithm is up to 4 orders of magnitude faster than the SOTA methods. This further confirms the effectiveness of our multiple landmark nodes approaches.

6.5 Effect of Parameters

In this subsection, we study the performance of our algorithms with different parameters, including (i) different settings of the landmark node set $\mathcal{V}_l$; (ii) different index building algorithms to support query processing algorithms.

Effect of $\mathcal{V}_l$. First, we study the effect of the choice of the landmark node set $\mathcal{V}_l$. We vary both the selection strategies and $|\mathcal{V}_l|$ (i.e. the size of the landmark node set $\mathcal{V}_l$). Specifically, we vary $|\mathcal{V}_l|$ from 1 to 20 for Email-enron, and vary $|\mathcal{V}_l|$ from 1 to 500 for PowerGrid. Five heuristic landmark node set selection strategies including degree, pagerank, Random, degree+ and pagerank+ are evaluated. As discussed in Section 4, degree chooses the nodes with the highest degree, pagerank picks the node with the largest PageRank centrality value when $\alpha = 0.15$, and Random selects nodes randomly. Moreover, degree+ first picks the node with the highest degree. Then, it removes the node and its neighbors, picks the next node with the highest degree among the remaining nodes. This process repeats until $|\mathcal{V}_l|$ nodes have been selected. pagerank+ is similar to degree+ but the node is selected by its PageRank centrality values. The results of Email-enron and PowerGrid are illustrated in Fig. 13-Fig. 16. We select 5 different $|\mathcal{V}_l|$ results to compare different landmark selection rules. We also show the results of all $|\mathcal{V}_l|$ values with the landmark selection rule degree+. Similar results can also be observed on the other datasets. For single-pair index building algorithms, we choose the algorithm RsfIndex as an example. The results of other index building algorithms are consistent. As can be seen in Fig. 13, the running time of RsfIndex decreases when $|\mathcal{V}_l|$ grows larger, while the empirical error of all the algorithms is similar. Among all the landmark node selection methods, all the proposed heuristics are significantly better than Random. For the proposed heuristics, the performance is at the same level, while degree+ is slightly better. For single-pair query processing algorithms, we choose RW as an example, the results of other query processing algorithms are consistent. The results can be found in Fig. 14, as expected, when we use degree+ to select the landmark nodes, it has the smallest running time, followed by pagerank+, pagerank and degree. The empirical errors of all the algorithms are similar. Moreover, Fig. 14 (e)-(h) show that when $|\mathcal{V}_l|$ increases, the running time and the empirical error of RW decrease on both datasets. For PowerGrid, the running time first decreases when $|\mathcal{V}_l|$ grows from 1 to 100, then increases when $|\mathcal{V}_l|$ grows larger than 200. This is because our query processing requires computing a quantity that is related to the index matrices. When $\mathcal{V}_l$ is large, the overhead of this computation is also large. Notice that the curves in Fig. 14(g)-(h) exhibit some oscillations. This is because RW is a Monte Carlo algorithm that can produce errors in each execution due to the randomness. Thus, when $|\mathcal{V}_l|$ increases, the error can also increase. However, a long-term decrease trend can be observed on both datasets. As a result, we set $|\mathcal{V}_l| = 10$ for social networks and $|\mathcal{V}_l| = 100$ for road networks by default. For single-source index building algorithms, we select the results of LeWalkIndex-SS to illustrate. As shown in Fig. 15, we can find similar results that degree+ achieves the best running time while the errors are similar. For single-source query processing algorithms, we can find that in Fig. 16, if we choose $\mathcal{V}_l$ by degree and pagerank, the performance of the algorithms will be much better than Random. degree+ and pagerank+ further improve the performance of the algorithms. These results suggest that degree+ is a very good landmark node set selection heuristics in practice, which is also used in our previous experiments.

Effect of different index building algorithms. In former experiments, we study the query performance by using a pre-defined index structure built by RsfIndex and LeWalkIndex-SS. Here, we also study the effect of the index quality with different index building algorithms. We vary the sample size of the index building algorithms from 10^2 to 10^5, and plot the relative error of the query processing algorithm v.s. the index building time of different algorithms. We show

the results for Bipush on Email-enron and PowerGrid, the results of other algorithms and datasets are consistent. The results are depicted in Fig. 17. As can be seen, the empirical error of the query processing algorithm Bipush decreases when the index building time increases. For single-pair query, the relative error of the query processing algorithm based on RwIndex is similar to the error based on RsfIndex when the sample size is the same, but the index building time is much longer than RsfIndex. For single-source query, the relative error of the query processing algorithm based on LeWalkIndex-SS is smaller than the error based on RsfIndex-SS, and the running time is also smaller. Thus, we choose the index built by RsfIndex and LeWalkIndex-SS as the default index structure for single-pair and single-source query processing algorithms, respectively.

7 RELATED WORK

Effectiveness resistance computation. There exist several practical algorithms to compute single-pair (or single-source) effective resistance on large graphs [31, 45, 67]. For example, Peng et al. [45] proposed several sublinear local algorithms based on random walk sampling to compute single-pair effective resistance on the graphs with bounded mixing time. Recently, Yang et al. [67] further improved their algorithm by optimizing the lengths of random walks. However, both of these algorithms can be costly when the effective resistance between the two query nodes is large. Liao et al. [31] proposed an alternative random walk sampling approach that utilizes an easily reachable landmark node. The fundamental idea behind their algorithm is that by using the easily reachable landmark node, the random walk sampling process can be terminated more efficiently, leading to significant improvements in efficiency. Nonetheless, their algorithm only employs a single landmark node, making it challenging to extend the method to use multiple landmark nodes. In our work, we address this challenge and also develop several efficient index algorithms for effective resistance computation.

In addition, there also exist several algorithms to compute the effective resistance of each edge (also called spanning edge centrality) in the graph [24, 40, 72]. Note that such a problem is often much easier than the problem of computing the effective resistance of any pair of query nodes (the two query nodes may not have an edge), and existing techniques for the spanning edge centrality computation cannot be used for single-pair (or single-source) effective resistance computation. It is worth mentioning that theoretically, all-pairs of effective resistance can be approximated in $O(\frac{m \log n}{\epsilon^2} + n^2)$ time [54]. However, the algorithm proposed in [54] relies on several complicated techniques which is only with theoretical interests, and its practical performance is often very poor on large graphs [31, 45, 67].

Personalized PageRank computation. Our problem is also closely related to the personalized PageRank (PPR) computation problem. Random walk sampling algorithms [11, 30, 64] and push based algorithms [8, 13, 20, 36, 37, 60] are proposed for computing PPR. The state-of-the-art algorithms compute PPR by combining the two techniques [30, 32, 33, 35, 62–64, 66]. Index-based methods are also proposed by computing the information of the hub nodes [25, 52, 61, 70]. These techniques have also been generalized to compute heat Kernel PageRank [26, 68]. However, all these techniques are mainly designed for PPR computation, and it is unclear how to generalize these techniques for effective resistance computation. In this paper, we propose several novel random walk sampling and push based algorithms for computing effective resistance, which are fundamentally different from existing PPR computation approaches.

Other landmark-based approaches. There are also some other landmark-based approaches in the field of graph data management. In the problem of shortest path computation [5, 22, 38, 44, 48, 58, 71], a set of landmark nodes is selected to accelerate shortest path distance query. Specifically, by using distances to landmark vertices and the triangle inequality, they were able to compute more accurate lower bounds leading to a significant speed-up of A* search. As the shortest path problem is closely related NN (nearest neighbor), kNN (k-nearest neighbor) search, a wide range of studies [1, 2, 15, 41, 42, 59] also select a landmark set and compute the distances from landmarks to all nodes as an index, building lower bounds to prune unnecessary computation in NN, kNN search. These studies are focused on the shortest path distance metric. In this paper, we study the problem of effective resistance computation, which is another distance metric defined based on random walks. Thus, our multiple landmark-based approaches are totally different from these studies.

8 CONCLUSION

In this paper, we develop a novel index-based approach to efficiently answer both the single-pair and single-source effective resistance (ER) queries. We first propose three new ER formulas based on a newly-developed multiple landmarks technique and a concept of Schur complement. These new formulas enable us to pre-compute a compact index which contains several small-sized matrices related to the landmark nodes. We propose several efficient and provable Monte Carlo algorithms to construct the index based on the newly-established probability interpretations of the Schur complement. With this powerful index, we also develop several provable query processing algorithms to efficiently answer both the single-pair and single-source ER queries. Extensive experiments on 5 real-life graphs show that with an acceptable additional space, our algorithms can achieve up to 4 orders of magnitude faster than the state-of-the-art algorithms while maintaining the same accuracy.

REFERENCES

[1] Tenindra Abeywickrama and Muhammad Aamir Cheema. 2017. Efficient Landmark-Based Candidate Generation for kNN Queries on Road Networks. In *Database Systems for Advanced Applications: 22nd International Conference, DASFAA 2017, Suzhou, China, March 27-30, 2017, Proceedings, Part II 22*. Springer, 425–440.

[2] Tenindra Abeywickrama, Muhammad Aamir Cheema, and David Taniar. 2016. k-Nearest Neighbors on Road Networks: A Journey in Experimentation and In-Memory Implementation. *Proceedings of the VLDB Endowment* 9, 6 (2016).

[3] Florian Adriaens, Honglian Wang, and Aristides Gionis. 2023. Minimizing Hitting Time between Disparate Groups with Shortcut Edges. *CoRR* abs/2306.03571 (2023).

[4] Rafig Agaev and Pavel Chebotarev. 2006. Spanning Forests of a Digraph and Their Applications. *CoRR* abs/math/0602061 (2006). arXiv:math/0602061

[5] 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.

[6] David J Aldous. 1990. The random walk construction of uniform spanning trees and uniform labelled trees. *SIAM Journal on Discrete Mathematics* 3, 4 (1990), 450–465.

[7] Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. 2007. Local Computation of PageRank Contributions. In *WAW*. 150–165.

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

[9] Eugenio Angriman, Maria Predari, Alexander van der Grinten, and Henning Meyerhenke. 2020. Approximation of the Diagonal of a Laplacian's Pseudoinverse for Complex Network Analysis. In *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference) (LIPIcs, Vol. 173)*. 6:1–6:24.

[10] Luca Avena and Alexandre Gaudillière. 2018. Two applications of random spanning forests. *Journal of Theoretical Probability* 31, 4 (2018), 1975–2004.

[11] Konstantin Avrachenkov, Nelly Litvak, Danil Nemirovsky, and Natalia Osipova. 2007. Monte Carlo Methods in PageRank Computation: When One Iteration is Sufficient. *SIAM J. Numer. Anal.* 45, 2 (2007), 890–904.

[12] Ravindra B Bapat. 2010. *Graphs and matrices*. Vol. 27. Springer.

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

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

[15] Sergey Brin. 1995. Near neighbor search in large metric spaces. In *VLDB*, Vol. 95. 574–584.

[16] Dongrun Cai, Xue Chen, and Pan Peng. 2023. Effective Resistances in Non-Expander Graphs. *arXiv preprint arXiv:2307.01218* (2023).

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

[18] Fan R. K. Chung and Lincoln Lu. 2006. Survey: Concentration Inequalities and Martingale Inequalities: A Survey. *Internet Math.* 3, 1 (2006), 79–127.

[19] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.

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

[21] Rajat Vadiraj Dwaraknath, Ishani Karmarkar, and Aaron Sidford. 2023. Towards Optimal Effective Resistance Estimation. *arXiv preprint arXiv:2306.14820* (2023).

[22] Andrew V Goldberg and Chris Harrelson. 2005. Computing the shortest path: A search meets graph theory. In *SODA*, Vol. 5. 156–165.

[23] Gramoz Goranci, Monika Henzinger, and Pan Peng. 2018. Dynamic Effective Resistances and Approximate Schur Complement on Separable Graphs. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland (LIPIcs, Vol. 112)*. 40:1–40:15.

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

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

[26] Kyle Kloster and David F. Gleich. 2014. Heat kernel based community detection. In *KDD*. ACM, 1386–1395.

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

[28] Huan Li, Richard Peng, Liren Shan, Yuhao Yi, and Zhongzhi Zhang. 2019. Current Flow Group Closeness Centrality for Complex Networks?. In *WWW*. ACM, 961–971.

[29] 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 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2728–2745.

[30] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, Hongyang Chen, Hongchao Qin, and Guoren Wang. 2023. Efficient Personalized PageRank Computation: The Power of Variance-Reduced Monte Carlo Approaches. *Proc. ACM Manag. Data* 1, 2 (2023), 160:1–160:26.

[31] 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. *Proc. ACM Manag. Data* 1, 1 (2023), 68:1–68:27.

[32] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, and Guoren Wang. 2022. Efficient Personalized PageRank Computation: A Spanning Forests Sampling Based Approach. In *SIGMOD*. ACM, 2048–2061.

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

[34] Yang Liu, Chuan Zhou, Shirui Pan, Jia Wu, Zhao Li, Hongyang Chen, and Peng Zhang. 2023. CurvDrop: A Ricci Curvature Based Approach to Prevent Graph Neural Networks from Over-Smoothing and Over-Squashing. In *WWW*. ACM, 221–230.

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

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

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

[38] Shlomi Maliah, Rami Puzis, and Guy Shani. 2017. Shortest path tree sampling for landmark selection in large networks. *Journal of Complex Networks* 5, 5 (2017), 795–815.

[39] Philippe Marchal. 2000. Loop-erased random walks, spanning trees and Hamiltonian cycles. *Electronic Communications in Probability* 5 (2000), 39–50.

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

[41] Luisa Micó, José Oncina, and Rafael C Carrasco. 1996. A fast branch & bound nearest neighbour classifier in metric spaces. *Pattern Recognition Letters* 17, 7 (1996), 731–739.

[42] María Luisa Micó, José Oncina, and Enrique Vidal. 1994. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognition Letters* 15, 1 (1994), 9–17.

[43] Abedelaziz Mohaisen, Aaram Yun, and Yongdae Kim. 2010. Measuring the mixing time of social graphs. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 383–389.

[44] 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 International Conference on Management of Data*. 709–724.

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

[46] Jim Pitman and Wenpin Tang. 2018. Tree formulas, mean first passage times and Kemeny's constant of a Markov chain. (2018).

[47] Jim Pitman and Wenpin Tang. 2018. Tree formulas, mean first passage times and Kemeny's constant of a Markov chain. *Bernoulli* 24, 3 (2018), 1942 – 1972.

[48] Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. 2009. Fast shortest path distance estimation in large networks. In *CIKM*. 867–876.

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

[50] Aaron Schild. 2018. An almost-linear time algorithm for uniform random spanning tree generation. In *STOC*. 214–227.

[51] Jieming Shi, Nikos Mamoulis, Dingming Wu, and David W. Cheung. 2014. Density-based place clustering in geo-social networks. In *SIGMOD*. ACM, 99–110.

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

[53] Ali Kemal Sinop, Lisa Fawcett, Sreenivas Gollapudi, and Kostas Kollias. 2021. Robust Routing Using Electrical Flows. In *SIGSPATIAL '21: 29th International Conference on Advances in Geographic Information Systems, Virtual Event / Beijing, China, November 2-5, 2021*. ACM, 282–292.

[54] Daniel A. Spielman and Nikhil Srivastava. 2008. Graph sparsification by effective resistances. In *STOC*. ACM, 563–568.

[55] Kumar Sricharan and Kamalika Das. 2014. Localizing anomalous changes in time-evolving graphs. In *SIGMOD*. ACM, 1347–1358.

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

[57] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. 2022. Understanding over-squashing and bottlenecks on graphs via curvature. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*.

[58] Konstantin Tretyakov, Abel Armas-Cervantes, Luciano García-Bañuelos, Jaak Vilo, and Marlon Dumas. 2011. Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. 1785–1794.

[59] Nimish Ukey, Zhengyi Yang, Binghao Li, Guangjian Zhang, Yiheng Hu, and Wenjie Zhang. 2023. Survey on exact knn queries over high-dimensional data space. *Sensors* 23, 2 (2023), 629.

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

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

[62] Sibo Wang and Yufei Tao. 2018. Efficient Algorithms for Finding Approximate Heavy Hitters in Personalized PageRanks. In *SIGMOD*. 1113–1127.

[63] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *KDD*. 505–514.

[64] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. 2018. TopPPR: Top-k Personalized PageRank Queries with Precision Guarantees on Large Graphs. In *SIGMOD*. 441–456.

[65] David Bruce Wilson. 1996. Generating random spanning trees more quickly than the cover time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 296–303.

[66] Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. 2021. Unifying the Global and Local Approaches: An Efficient Power Iteration with Forward Push. In *SIGMOD*. 1996–2008.

[67] Renchi Yang and Jing Tang. 2023. Efficient Estimation of Pairwise Effective Resistance. *Proc. ACM Manag. Data* 1, 1 (2023), 16:1–16:27.

[68] Renchi Yang, Xiaokui Xiao, Zhewei Wei, Sourav S. Bhowmick, Jun Zhao, and Rong-Hua Li. 2019. Efficient Estimation of Heat Kernel PageRank for Local Clustering. In *SIGMOD*. ACM, 1339–1356.

[69] Hongzhi Yin, Bin Cui, Jing Li, Junjie Yao, and Chen Chen. 2012. Challenging the Long Tail Recommendation. *VLDB* 5, 9 (2012), 896–907.

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

[71] Junhua Zhang, Wentao Li, Long Yuan, Lu Qin, Ying Zhang, and Lijun Chang. 2022. Shortest-path queries on complex networks: experiments, analyses, and improvement. *VLDB* 15, 11 (2022), 2640–2652.

[72] Shiqi Zhang, Renchi Yang, Jing Tang, Xiaokui Xiao, and Bo Tang. 2023. Efficient Approximation Algorithms for Spanning Centrality. In *KDD*. ACM, 3386–3395.