Efficient Algorithms for Minimizing the Kirchhoff Index by Adding Edges

Anonymous Author(s)

ABSTRACT

The Kirchhoff index, which is the sum of the resistance distance between every pair of nodes in a network, is a key metric for gauging network performance, where lower values signify enhanced performance. In this paper, we study the problem of minimizing the Kirchhoff index by adding edges. We show that the objective function of this problem is not supermodular, refuting the claim of supermodularity by the prior work. Furthermore, we provide a greedy algorithm for solving this problem and give an analysis of its performance based on the bounds of the submodularity ratio and the curvature. Then, we introduce a gradient-based greedy algorithm as a new paradigm to solve this problem. To accelerate the computation cost, we leverage geometric properties, convex hull approximation, and approximation of the projected coordinate of each point. To further improve the algorithm, we use pre-pruning and fast update techniques, rendering it especially apt for large-scale networks. Our proposed algorithms have linear time complexity. We provide extensive experiments on ten real networks to evaluate the performance of our algorithms. The results demonstrate that our proposed algorithms outperform the state-of-the-art methods in terms of efficiency and effectiveness. Moreover, our algorithms are scalable to large graphs with over 5 million nodes and 12 million edges.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**; • **Mathematics of computing → Combinatoric problems**; • **Networks → Network algorithms**.

KEYWORDS

Kirchhoff index, graph algorithm, discrete optimization, convex hull approximation, effective resistance

1 INTRODUCTION

Regarded as a foundational metric in graph theory, effective resistance—also referred to as resistance distance—is derived in the context of an electrical network. This metric has fostered a plethora of pivotal insights and contributions spanning both theoretical and applied dimensions. Within theoretical confines, it has emerged as a linchpin for algorithmic graph theory, facilitating groundbreaking developments in computational algorithms that address key challenges such as spectral graph sparsification [66], maximum flow approximations [14], and the generation of random spanning trees [50], among others. Concurrently, its applied dimensions have seen successful deployments in domains ranging from collaborative recommendation systems [24] and graph embedding techniques [11] to image segmentation [7] and network influence mining [10, 42, 63]. This dual relevance—both theoretical and applied—has catalyzed extensive scholarly scrutiny over recent decades [19, 20, 69].

Beyond the fundamental concept of resistance distance itself, a large range of graph invariants based on this metric has been defined and explored, most notably the Kirchhoff index [28, 37]. This index, computed as the aggregate of effective resistances across all pairs of nodes within a graph, has found applications across a diverse spectrum of domains [43, 79, 81]. It serves as an instrumental tool for evaluating network connectedness [70], assessing the global utility of social recommender architectures [74], and gauging the robustness of first-order consensus algorithms in environments fraught with noise [55, 60, 80]. Furthermore, it has been used as the foundation to define efficiency and fairness in information access on social networks [47] and the oversquashing in GNN [9]. In most setups, a network with smaller values of Kirchhoff index is more beneficial and enhances the performance.

Since the Kirchhoff index encodes the performance of various systems, with smaller Kirchhoff index indicating better performance, a substantial amount of effort has been devoted to optimizing Kirchhoff index by executing different graph operations, cf. [9]. Motivated by real-world instances such as link recommendation systems in online social platforms, a substantial amount of attention has been devoted to optimizing the Kirchhoff index by adding edges [31, 59, 68, 76]. The problem studied in this paper falls under the umbrella of this line of work.

We study the following optimization problem: For a given connected undirected unweighted graph $\mathcal{G} = (V, E)$ with n nodes and $m \ll n^2$ edges, a positive integer $k \ll n$, how to add k nonexistent edges from a candidate edge set $Q = (V \times V) \backslash E$ to graph $\mathcal{G}$, so that the Kirchhoff index for the resulting graph is minimized. Although this problem has been widely studied by the prior work [9, 31, 59, 68, 76], the existing algorithms are not practical for large networks with millions of nodes, which emerge regularly in the real world, since these algorithms have a quadratic (or larger) run time. The main aim of the present work is to address this issue by devising a more efficient algorithm which can handle networks of large size.

The conventional and predominant approach to tackle the Kirchhoff index optimization problem has been the employment of the greedy approach [68, 76]. This algorithm, through its iterative selection of an optimal edge over k rounds, yields a suboptimal solution to the problem. The time complexity of their greedy algorithm stands at $O(kn^3)$, rendering it unsuitable for large-scale networks. Furthermore, their presumption that the objective function is supermodular, which results in an $(1 - 1/e)$ approximation ratio for the algorithm, is false, as we prove in this paper. In response, we present a revised approximation ratio bound for the greedy algorithm, underpinned by the bounds of submodularity ratio and the curvature of the related function. Additionally, we enhance the time complexity of the greedy algorithm from $O(kn^3)$ to $O(n^3 + kn^2)$.

Given the substantial computational demands of the greedy algorithm in solving the Kirchhoff index problem, the authors of [59] introduced several algorithms to expedite the computation. Among these, the most proficient is the state-of-the-art algorithm, ColStochJLT. This algorithm harnesses randomized techniques to augment the performance of the greedy algorithm, incorporating a sub-sampling step and random projection. Its time complexity is denoted as $\tilde{O}(n^2+km)$, where the $\tilde{O}(\cdot)$ notation obscures $\log n$ factors. Nonetheless, even $\tilde{O}(n^2+km)$ can be computationally intensive, and their empirical studies are confined to networks with fewer than 100,000 nodes. Moreover, their sub-sampling step, grounded in empirical observations, lacks rigorous theoretical justification.

To mitigate the algorithm's time complexity, we advocate for a gradient-based greedy algorithm. The primary bottleneck in the traditional greedy algorithm is the necessity to query $|Q|$ instances of marginal decrease in the objective function. This process demands $\Omega(n^2)$ time, especially considering that real-world networks tend to be sparse. Leveraging the gradient-based greedy algorithm, we can substantially curtail the number of queries. This reduction is achieved by harnessing geometric properties to determine the convex hull of n points, the coordinates of which are approximated through projection and the Laplacian solver. Furthermore, we introduce pre-pruning and rapid update techniques to accelerate this gradient-based greedy algorithm. We also propose an algorithm by determining the convex hull for only one time, and this algorithm is the most efficient one. All these gradient-based algorithms boast reduced time complexities, and we furnish error analyses for each iteration. Comprehensive experimental evaluations underscore both the efficacy and efficiency of our proposed algorithms.

Our research culminates in the following key contributions:

Advanced Analysis of the Kirchhoff Index and Greedy Algorithm: Upon a detailed examination of the objective function inherent to the Kirchhoff index optimization problem, it becomes evident that the said function is not supermodular in nature. This directly implies that the conventional greedy algorithm, traditionally deployed to address this issue, lacks the desired $(1-1/e)$ approximation guarantee. To mitigate this limitation, we delineate the boundaries of the submodularity ratio, represented as γ, in conjunction with the curvature α of the function under consideration. As a result, a renewed and corrected approximation guarantee of $(1-e^{-\gamma\alpha})/\alpha$ is established for the greedy algorithm. Noteworthy is our refinement in the time complexity of the greedy algorithm from an existing $O(kn^3)$ to a more streamlined $O(n^3+kn^2)$. In a subsequent advancement, we augment the algorithm's efficiency through the incorporation of two approximation methodologies, culminating in an enhanced greedy algorithm characterized by a $\tilde{O}(k|Q|\epsilon^{-2})$ time complexity, accompanied by a comprehensive error analysis.

A Novel Paradigm for the Kirchhoff Index Optimization Problem along with Efficient Algorithmic Solutions: We put forward a gradient-based greedy algorithm tailored for the Kirchhoff index optimization problem. An important property of this approach is the circumvention of the otherwise expensive quadratic number of queries typically encountered in each iteration. This reduction is achieved by leveraging the convex hull approximation algorithm, thereby considerably minimizing the query count. The efficiency of this method is conditional on the fast calculation of each node's coordinates, estimated via random projection in tandem with the Laplacian solver. This algorithm enjoys an almost linear time complexity and rigorous error analysis. To further improve the computational efficacy, we introduce a novel pre-pruning and update mechanism, which does not affect the algorithm's error analysis.

Comprehensive Experimental Evaluation: To complement our theoretical findings, we conduct a large set of experiments. We evaluate the performance of the algorithms on a diverse set of real-world networks. The outcomes of our experiments demonstrate that our proposed algorithms consistently and significantly outperform the state-of-the-art algorithms. Thus, our proposed solutions not only posses theoretical guarantees, but also are very effective and efficient in practice.

2 PRELIMINARIES

In this section, we introduce several notations and tools that are useful for the purpose of describing and analyzing the problems and algorithms.

2.1 Notations

We use $\boldsymbol{A}^\top$ and $\boldsymbol{a}^\top$ to represent the transpose of $\boldsymbol{A}$ and $\boldsymbol{a}$, respectively. Let $\boldsymbol{e}_i$ denote the column vector of appropriate dimension, where the i-th element is 1, and other elements are 0. We define $\mathbf{1}$ to be an appropriate-dimension column vector with all entries being ones. Let $\boldsymbol{\mathcal{J}}$ be the matrix with all elements being 1. For a matrix $\boldsymbol{A}$, $\boldsymbol{A}_{i,j}$ denotes the element of $\boldsymbol{A}$ at i-th row and j-th column. Let $\boldsymbol{a}_i$ be the i-th element of the vector $\boldsymbol{a}$. For any vector $\boldsymbol{a}$, we use $\|\boldsymbol{a}\|_2 = \sqrt{\sum_i \boldsymbol{a}_i^2}$ to denote the ℓ_2 norm of the vector $\boldsymbol{a}$, and use $\|\boldsymbol{a}\|_{\boldsymbol{X}} = \sqrt{\boldsymbol{a}^\top \boldsymbol{X} \boldsymbol{a}}$ to denote the matrix norm of the vector $\boldsymbol{a}$ for the given matrix $\boldsymbol{X}$.

For two non-negative scalars a and b, we use $a \approx_\epsilon b$ to denote that a is an ϵ-approximation of b obeying relation $(1-\epsilon)b \le a \le (1+\epsilon)b$.

2.2 Graph and Related Matrices

We consider $\mathcal{G} = (V, E)$, a connected, undirected, and unweighted graph comprised of $|V| = n$ nodes and $|E| = m$ edges. Let N_i denote the set of nodes adjacent to a node $i \in V$, in which case the degree of node i is given by $|N_i|$.

The adjacency matrix of graph $\mathcal{G}$, denoted by $\boldsymbol{A}$, is an $n \times n$ matrix such that $\boldsymbol{A}_{i,j} = 1$ if nodes i and j are adjacent, and $\boldsymbol{A}_{i,j} = 0$ otherwise. The degree matrix, represented by $\boldsymbol{D}$ of graph $\mathcal{G}$, is a diagonal matrix $\boldsymbol{D} = \mathrm{diag}(d_1, d_2, \cdots, d_n)$, where the i-th diagonal element $d_i = |N_i|$ represents the degree of node i.

The Laplacian matrix of graph $\mathcal{G}$, denoted by $\boldsymbol{L}$, is expressed as $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$. This matrix is symmetric, semi-positive definite, and singular, thereby rendering it non-invertible. The eigenvalues of $\boldsymbol{L}$ are sorted in non-decreasing order as $0 = \lambda_1(\boldsymbol{L}) \le \lambda_2(\boldsymbol{L}) \le \cdots \le \lambda_n(\boldsymbol{L})$. The pseudoinverse of $\boldsymbol{L}$ is denoted by $\boldsymbol{L}^\dagger$ and calculated as $\boldsymbol{L}^\dagger = \left(\boldsymbol{L} + \frac{1}{n}\boldsymbol{\mathcal{J}}\right)^{-1} - \frac{1}{n}\boldsymbol{\mathcal{J}}$ [28], where $\boldsymbol{\mathcal{J}}$ is an $n \times n$ matrix with all entries being 1.

2.3 Effective Resistance and Kirchhoff Index

In any given graph $\mathcal{G} = (V, E)$, the notion of effective resistance emerges when we consider replacing each edge in E with a unit resistor, thus forming an analogous electrical network [21]. For any two nodes, i and j, we use r_{ij} to denote the effective resistance between them. By its definition, r_{ij} represents the potential difference between nodes i and j when a unit current flows from i to j in the associated electrical network. Furthermore, it can be articulated [6] using the pseudoinverse of the graph's Laplacian matrix as

$$r_{ij} = \boldsymbol{L}^{\dagger}_{i,i} + \boldsymbol{L}^{\dagger}_{j,j} - \boldsymbol{L}^{\dagger}_{i,j} - \boldsymbol{L}^{\dagger}_{j,i}. \tag{1}$$

As a measure of robustness [26], the effective resistance quantifies the level of connectivity within a network. A smaller value corresponds to a more robust network [22, 28]. The effective resistance possesses numerous desirable properties, such as strictly decreasing upon edge addition [23], and considering both the quantity and length of paths between node pairs.

Due to the paramount importance of resistance distance [28], various graph resistance distance based invariants are defined and studied. Among these, the Kirchhoff index holds a prominent position [37]. The Kirchhoff index of a graph $\mathcal{G}$ is defined as

$$K(\mathcal{G}) = \frac{1}{2} \sum_{i,j \in V} r_{ij}. \tag{2}$$

This index can be computed based on the sum of the inverse nonzero Laplacian eigenvalues [37], or equivalently, the trace of the pseudoinverse of the Laplacian matrix [28], as expressed by

$$K(\mathcal{G}) = n \sum_{i=2}^{n} \frac{1}{\lambda_i(\boldsymbol{L})} = n\mathrm{Tr}\left(\boldsymbol{L}^{\dagger}\right). \tag{3}$$

2.4 Concepts Related to Set Function

We give a brief introduction to some concepts about set functions.

Definition 2.1 (Monotonicity). *For set Q, a set function $f : 2^Q \to \mathbb{R}$ is monotone non-decreasing (or non-increasing) if $f(H) \le f(T)$ (or $f(H) \ge f(T)$) holds for all $H \subseteq T \subseteq Q$.*

Definition 2.2 (Submodularity). *A set function $f : 2^Q \to \mathbb{R}$ is said to be submodular if*

$$f(H \cup \{u\}) - f(H) \ge f(T \cup \{u\}) - f(T)$$

holds for all $H \subseteq T \subseteq Q$ and $u \in Q \backslash T$.

Definition 2.3 (Supermodularity). *A set function $f : 2^Q \to \mathbb{R}$ is said to be supermodular if*

$$f(H \cup \{u\}) - f(H) \le f(T \cup \{u\}) - f(T)$$

holds for all $H \subseteq T \subseteq Q$ and $u \in Q \backslash T$.

A significant portion of set functions in optimization problems are not submodular (or supermodular). For such functions, one can define some quantities to observe the gap between them and submodular (or supermodular) functions.

Definition 2.4 (Submodularity ratio [18]). *For a non-negative set function $f : 2^Q \to \mathbb{R}$, its submodularity ratio is defined as the largest scalar γ satisfying*

$$\sum_{u \in T \backslash H} (f(H \cup \{u\}) - f(H)) \ge \gamma (f(T) - f(H)), \forall H \subseteq T \subseteq Q.$$

Definition 2.5 (Curvature [8]). *For a non-negative set function $f : 2^Q \to \mathbb{R}$, its curvature is defined as the smallest scalar α satisfying*

$$f(T) - f(T \backslash \{u\}) \ge (1-\alpha)(f(H) - f(H \backslash \{u\})), \forall H \subseteq T \subseteq Q, u \in H.$$

3 PROBLEM FORMULATION

The Kirchhoff index $K(\mathcal{G})$ of a graph $\mathcal{G}$ is a good measure in many application scenarios. For example, it serves as a measure of overall network connectivity [70], edge centrality within complex networks [43], and robustness of the first-order consensus algorithm in noisy networks [55, 60, 80]. Both first-order and second-order consensus problems have garnered considerable attention from the scientific community [64, 79, 81]. In these practical aspects, a smaller value of $K(\mathcal{G})$ indicates that the systems have a better performance. Based on Rayleigh's monotonicity law, adding edges to graph $\mathcal{G}$ will lead to a decrease of $K(\mathcal{G})$ [23], which motivates us to study the problem of how to minimize $K(\mathcal{G})$ by adding k new edges.

3.1 Problem Statement

Based on Rayleigh's monotonicity law, adding a set of edges to the graph will result in a decrease in the effective resistance between any pair of nodes [23]. Consequently, the Kirchhoff index of the graph will decrease when new edges are incorporated into the graph. We focus on determining how to optimally add a batch of k new nonexistent edges such that the decrease in the Kirchhoff index is maximized. We present the following Kirchhoff index minimization problem, which has been studied in previous works [9, 31, 38, 58, 59, 68, 76].

Problem 1. *Given a connected graph $\mathcal{G} = (V, E)$, a candidate edge set $Q = (V \times V) \backslash E$ and an integer $k \ll |Q|$, we aim to identify a subset $T \subset Q$ of k edges. These edges are then added to the graph to create a new graph $\mathcal{G}(T) = (V, E \cup T)$ such that the Kirchhoff index $K(T) := K(\mathcal{G}(T))$ of the graph $\mathcal{G}(T)$ is minimized. More formally, the objective is to find:*

$$T^* = \mathop{\arg\max}_{T \subseteq Q, |T|=k} K(T).$$

When $\mathcal{G}$ is a dense graph, the Kirchhoff index $K(\mathcal{G})$ is very small for practical applications. In what follows, we focus on sparse graphs, indicating $m \ll n^2$, which is satisfied in most real-world networks. Also, due to the cost constraints of the problem in real scenarios, we can only add a small number of edges, so we let $k \ll |Q|$ in Problem 1.

The Problem 1 is NP-hard [38], and is proved by the reduction form 3-colorability problem. The combinatorial nature of this problem allows for a brute-force approach, which involves exhausting all $\binom{|Q|}{k}$ possible subsets of edges. For each subset of edges T, we can compute the Kirchhoff index for its associated graph by inverting a matrix, a process which requires $\Omega(n^3)$ time. This results in a total complexity of $\Omega(\binom{|Q|}{k} n^3)$, which is very expensive even for small values of k. To avoid such high computational cost, we will explore the way to solve this problem in next sections.

3.2 Deterministic Greedy Algorithm

In [59, 68], the authors proposed a greedy algorithm by adding edges with the largest marginal decrease (best decrease of the Kirchhoff index) in each of the k iterations. To bypass the repeated operations of marginal decrease computation involved in matrix inversion, they adopted the Sherman-Morrison formula [53] to execute rank-1 updates of matrices. To be specific, for any edge $e \in Q$, the marginal decrease $\Delta(e) = K(\mathcal{G}) - K(\{e\})$ is computed by the Sherman-Morrison formula as

$$\Delta(e) = n\mathrm{Tr}\left(\boldsymbol{L}^{\dagger}\right) - n\mathrm{Tr}\left(\left(\boldsymbol{L}+\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\right)^{\dagger}\right) = n\frac{\boldsymbol{b}_e^{\top}\boldsymbol{L}^{2\dagger}\boldsymbol{b}_e}{1+\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e}. \quad (4)$$

Moreover, after adding an edge $e \in Q$ to the graph, the pseudoinverse of the Laplacian matrix of the new graph can be updated by

$$\left(\boldsymbol{L}+\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\right)^{\dagger} = \boldsymbol{L}^{\dagger} - \frac{\boldsymbol{L}^{\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}}{1+\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e}. \quad (5)$$

Based on Equations (4) and (5), in [59, 68] a simple greedy algorithm was designed as outlined in Algorithm 1 excluding Line 2 and Line 10. This simple greedy first computes the pseudoinverse of $\boldsymbol{L}$ as a preprocessing step in $O(n^3)$ (Line 1). Then, in each iteration, the marginal gain $\Delta(e)$ of nonexistent edges $e \in Q$ are computed in $O(n)$ time per edge (Line 4). The edge with the largest marginal decrease is added to the graph (Line 5), and the pseudoinverse is updated in $O(n^2)$ (Line 9). Thus, the total time complexity of this greedy algorithm is in $O(n^3 + k|Q|n)$ [59, 68].

However, their proposed greedy algorithm can be improved. Next, we provide a more efficient greedy algorithm, and analyze its performance.

3.2.1 Reducing Time Complexity. We notice that by using the Sherman-Morrison formula, the simple greedy algorithm maintains the pseudoinverse of the Laplacian matrix, which results in the $O(n)$ time cost for computing the marginal decrease $\Delta(e)$ for each $e \in Q$. However, it is still time-consuming. To accelerate the marginal decrease computation process, we maintain two matrices, $\boldsymbol{L}^{\dagger}$ and $\boldsymbol{L}^{2\dagger}$, during k iterations. Our proposed deterministic greedy algorithm is called DETER, outlined in Algorithm 1.

First, we precompute two matrices, $\boldsymbol{L}^{\dagger}$ and $\boldsymbol{L}^{2\dagger}$, in Line 1-2, in $O(n^3)$ time. Then, in each iteration, the computation of $\Delta(e)$ can be reduced to $O(1)$ time complexity per edge. After we select one optimal edge e, we add it to the graph and update $\boldsymbol{L}^{\dagger}$ and $\boldsymbol{L}^{2\dagger}$. According to the Sherman-Morrison formula, we have

$$\left(\boldsymbol{L}+\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\right)^{2\dagger} = \left(\boldsymbol{L}^{\dagger} - \frac{\boldsymbol{L}^{\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}}{1+\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e}\right)^2$$
$$=\boldsymbol{L}^{2\dagger} + \frac{\boldsymbol{L}^{\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{2\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}}{(1+\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e)^2} - \frac{\boldsymbol{L}^{2\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}}{1+\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e} - \frac{\boldsymbol{L}^{\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{2\dagger}}{1+\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e},$$

indicating that $\boldsymbol{L}^{2\dagger}$ can be updated in $O(n^2)$ time complexity in Line 10.

Based on this improvement, our proposed deterministic greedy algorithm DETER has the total time complexity of $O(n^3+kn^2)$, much smaller than $O(n^3+k|Q|n)$ [59, 68], when Q is quadratic in n which is the case based on the assumption $m \ll n^2$.

3.2.2 Non-supermodularity of the Kirchhoff Index. To analyze the performance of Algorithm 1, we need to investigate some properties of the objective function. Interestingly, contrary to the proofs in [68, 76] that the objective function is a supermodular function, we

```
Algorithm 1: DETER($\mathcal{G}, S_1, S_0, Q, k$)
  Input  : A connected graph $\mathcal{G} = (V, E)$; a candidate edge
           set $Q = (V \times V)\backslash E$; an integer $1 \le k \le |Q|$
  Output : A subset $T \subseteq Q$ with $|T| = k$
1 Compute $\boldsymbol{L}^{\dagger}$
2 Compute $\boldsymbol{L}^{2\dagger}$
3 Initialize solution $T = \emptyset$
4 for $i = 1$ to $k$ do
5     Compute $\Delta(e) = K(\mathcal{G}) - K(\{e\})$ for each $e \in Q$
6     Select $e_i$ s. t. $e_i \leftarrow \arg\max_{e \in Q}\Delta(e)$
7     Update solution $T \leftarrow T \cup \{e_i\}$
8     Update the graph $\mathcal{G} \leftarrow \mathcal{G}(V, E \cup \{e_i\})$
9     Update $\boldsymbol{L}^{\dagger} \leftarrow \boldsymbol{L}^{\dagger} - \frac{\boldsymbol{L}^{\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}}{1+\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e}$
10    Update
      $\boldsymbol{L}^{2\dagger} \leftarrow \boldsymbol{L}^{2\dagger} + \frac{\boldsymbol{L}^{\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{2\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}}{(1+\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e)^2} - \frac{\boldsymbol{L}^{2\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}}{1+\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e} - \frac{\boldsymbol{L}^{\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{2\dagger}}{1+\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e}$
11    Update the candidate edge set $Q \leftarrow Q\backslash e_i$
12 return $T$.
```

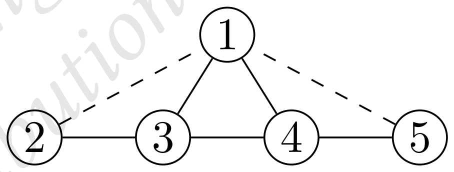

Figure 1: A counterexample with 5 nodes for Theorem 3.1.

assert that this claim is not correct. This observation is formalized in the following theorem.

THEOREM 3.1. *The objective function, namely the Kirchhoff index $K(\cdot) : 2^Q \to \mathbb{R}$, is not supermodular. In other words, there exist two edge sets A and B, with $A \subseteq B \subseteq Q$ and an edge $e \in Q\backslash B$, which satisfy*

$$K(A) - K(A \cup \{e\}) < K(B) - K(B \cup \{e\}).$$

Proof. To illustrate its non-supermodularity, we consider an example of a graph with five nodes as depicted in Figure 1. The solid lines represent the original graph, while the dashed lines correspond to the candidate edges. Let $A = \emptyset$, $B = \{(1, 2)\}$ and $e = (1, 5)$. Simple computations yield

$$K(A) = 13.33, \quad K(A \cup \{e\}) = 10.25,$$
$$K(B) = 10.25, \quad K(B \cup \{e\}) = 6.95.$$

Therefore, we obtain

$$K(A) - K(A \cup \{e\}) = 3.08 < 3.3 = K(B) - K(B \cup \{e\}).$$

This concludes the proof. □

3.2.3 Evaluating the Greedy Algorithm's Performance. Although the objective function $K(\cdot)$ of Problem 1 is not supermodular, the greedy algorithm usually has a good performance [8, 18] with a tight approximation guarantee of $(1 - e^{-\gamma\alpha})/\alpha$ [8] where γ and α are submodularity ratio and curvature of the non-negative increasing function $f(T) = K(\mathcal{G}) - K(T)$ for $T \subseteq Q$, respectively. The effectiveness of Algorithm 1 is summarized in Theorem 3.2, which can be proved similarly using the method in [8].

Theorem 3.2. *Let $\gamma \in [0,1]$ and $\alpha \in [0,1]$ be submodularity ratio and curvature of the function $f(T) = K(\mathcal{G}) - K(T)$ for $T \subseteq Q$. Then, the edge set T returned by Algorithm 1 satisfies*

$$K(\mathcal{G}) - K(T) \geq \frac{1}{\alpha}(1 - e^{-\alpha\gamma})(K(\mathcal{G}) - K(T^*)), \qquad (6)$$

where

$$T^* = \underset{T \subseteq Q, |T|=k}{\arg\min} \; K(T).$$

Since the effectiveness of the greedy algorithm in Problem 1 can be evaluated by determining its submodularity ratio γ and curvature α, we next provide bounds for γ and α, respectively.

Theorem 3.3. *The submodularity ratio γ of the set function $f(S) = K(\mathcal{G}) - K(S)$ is bounded as follows:*

$$1 > \gamma \geq \left(\frac{\lambda_2(\boldsymbol{L})}{n}\right)^2 > 0, \qquad (7)$$

and its curvature α is constrained by:

$$0 < \alpha \leq 1 - \left(\frac{\lambda_2(\boldsymbol{L})}{n}\right)^2 < 1. \qquad (8)$$

Proof. Let $Q = (V \times V)\backslash E$ be the candidate set and S, T be any two subsets of candidate edge set Q. To begin with, we first derive a lower and upper bound for the marginal benefit function $g_T(S) = f(S \cup T) - f(S)$, respectively.

On the one hand,

$$g_T(S) = f(S \cup T) - f(S) = K(S) - K(S \cup T)$$
$$= n\mathrm{Tr}\left(\boldsymbol{L}(S)^\dagger\right) - n\mathrm{Tr}\left(\boldsymbol{L}(S \cup T)^\dagger\right) = \sum_{i=1}^{n-1} \frac{n}{\lambda_i(\boldsymbol{L}(S))} - \frac{n}{\lambda_i(\boldsymbol{L}(S \cup T))}$$
$$= n\sum_{i=2}^{n} \frac{\lambda_i(\boldsymbol{L}(S \cup T)) - \lambda_i(\boldsymbol{L}(S))}{\lambda_i(\boldsymbol{L}(S))\lambda_i(\boldsymbol{L}(S \cup T))} \geq n\frac{\mathrm{Tr}\left(\boldsymbol{L}(S \cup T)\right) - \mathrm{Tr}\left(\boldsymbol{L}(S)\right)}{\lambda_n(\boldsymbol{L}(S))\lambda_n(\boldsymbol{L}(S \cup T))}$$
$$= \frac{2n|T\backslash S|}{\lambda_n(\boldsymbol{L}(S))\lambda_n(\boldsymbol{L}(S \cup T))}.$$

On the other hand,

$$g_T(S) = n\sum_{i=1}^{n-1} \frac{\lambda_i(\boldsymbol{L}(S \cup T)) - \lambda_i(\boldsymbol{L}(S))}{\lambda_i(\boldsymbol{L}(S))\lambda_i(\boldsymbol{L}(S \cup T))}$$
$$\leq n\frac{\mathrm{Tr}\left(\boldsymbol{L}(S \cup T)\right) - \mathrm{Tr}\left(\boldsymbol{L}(S)\right)}{\lambda_2(\boldsymbol{L}(S))\lambda_2(\boldsymbol{L}(S \cup T))} = \frac{2n|T\backslash S|}{\lambda_2(\boldsymbol{L}(S))\lambda_2(\boldsymbol{L}(S \cup T))}.$$

Then, we put the above two bounds together and derive the lower bound of the submodular ratio γ.

$$\frac{\sum_{e \in T\backslash S} g_{\{e\}}(S)}{g_T(S)}$$
$$\geq \sum_{e \in T\backslash S} \frac{2n}{\lambda_n(\boldsymbol{L}(S))\lambda_n(\boldsymbol{L}(S \cup \{e\}))} \times \frac{\lambda_2(\boldsymbol{L}(S))\lambda_2(\boldsymbol{L}(S \cup T))}{2n|T\backslash S|}$$
$$\geq \left(\frac{\lambda_2(\boldsymbol{L})}{\lambda_n(\boldsymbol{L}(Q))}\right)^2 = \left(\frac{\lambda_2(\boldsymbol{L})}{n}\right)^2.$$

The last equality holds since the largest eigenvalue of the Laplacian matrix of a complete graph with n nodes is n.

Similarly, we derive the upper bound of the curvature α. Let j be any candidate edge in $S\backslash T$. Then, we have

$$\frac{g_{\{j\}}(S \cup T\backslash\{j\})}{g_{\{j\}}(S\backslash\{j\})}$$
$$\geq \frac{2n}{\lambda_n(\boldsymbol{L}(S \cup T\backslash\{j\}))\lambda_n(\boldsymbol{L}(S \cup T))} \times \frac{\lambda_1(\boldsymbol{L}(S\backslash\{j\}))\lambda_1(\boldsymbol{L}(S))}{2n}$$
$$\geq \left(\frac{\lambda_2(\boldsymbol{L})}{\lambda_n(\boldsymbol{L}(Q))}\right)^2 = \left(\frac{\lambda_2(\boldsymbol{L})}{n}\right)^2,$$

which combining with the curvature definition completes the proof of Equation (8). □

Pursuing the enhancement of the theoretical efficiency of this greedy algorithm, a larger approximation ratio $(1 - e^{-\gamma\alpha})/\alpha$ is favorable. This entails that a superior γ and an inferior α could augment the magnitude of $(1 - e^{-\gamma\alpha})/\alpha$. In addition, we discern from Theorem 3.3 that the boundaries of γ and α are intrinsically related to $\lambda_2(\boldsymbol{L})$. Predominantly, denser graphs are associated with a larger $\lambda_2(\boldsymbol{L})$ [27], suggesting an enlarged lower bound for γ and a diminished upper bound for α. Thus, denser graphs potentially allow for a tighter approximation ratio bound.

3.3 Enhanced Greedy Algorithm via Two Approximation Algorithms

Algorithm 1 is primarily impeded by the time-consuming step of inverting the Laplacian matrix. While this step guarantees an accurate computation of the marginal decrease of the Kirchhoff index at every stage, it hampers the scalability of the greedy algorithm. To address this, we introduce two approximation algorithms to expedite the computation of the marginal decrease.

In every iteration, the marginal decrease of the Kirchhoff index $\Delta(e) = K(\mathcal{G}) - K(\{e\})$ can be rewritten as $\frac{\boldsymbol{b}_e^\top \boldsymbol{L}^{2\dagger} \boldsymbol{b}_e}{1+\boldsymbol{b}_e^\top \boldsymbol{L}^\dagger \boldsymbol{b}_e}$ for any $e \in Q$. We realize that the crux of the marginal decrease computation is the calculation of the effective resistance $\boldsymbol{b}_e^\top \boldsymbol{L}^\dagger \boldsymbol{b}_e$ and the biharmonic distance $\boldsymbol{b}_e^\top \boldsymbol{L}^{2\dagger} \boldsymbol{b}_e$ between all non-incident pairs of nodes in graph $\mathcal{G}$.

To circumvent matrix inversion, the authors of [66] proposed an algorithm, ERComp($\mathcal{G}$,Q,ϵ), which constructs a data structure in $\tilde{O}(m\epsilon^{-2})$ time, where $\tilde{O}(\cdot)$ suppresses poly$(\log n)$ factors, and then returns the approximate effective resistance for each node pair incident to the edge $e \in Q$ in $O(\log n\epsilon^{-2})$ time with relative error ϵ. Thus, the total time required to query every effective resistance between node pair incident to the candidate edge $e \in Q$ is $\tilde{O}(m\epsilon^{-2} + |Q|\epsilon^{-2})$. Later, the authors of [78] utilized this idea to propose another algorithm, BDComp($\mathcal{G}$,Q,ϵ), for evaluating the biharmonic distance. Similarly, the total time cost for querying biharmonic distance between node pairs in set Q is $\tilde{O}(m\epsilon^{-2} + |Q|\epsilon^{-2})$.

Utilizing these two efficient algorithms, we can obtain two sets $\{(e, \tilde{r}(e))|e \in Q\} = $ ERComp$(\mathcal{G}, Q, \epsilon/2)$ and $\{(e, \tilde{b}(e))|e \in Q\} = $ BDComp$(\mathcal{G}, Q, \epsilon/2)$, which respectively represent the approximated effective resistance and the biharmonic distance for each node pair incident to the candidate edge $e \in Q$. Consequently, $\tilde{r}(e) \approx_{\epsilon/2} \boldsymbol{b}_e^\top \boldsymbol{L}^\dagger \boldsymbol{b}_e$ and $\tilde{b}(e) \approx_{\epsilon/2} \boldsymbol{b}_e^\top \boldsymbol{L}^{2\dagger} \boldsymbol{b}_e$ hold for any candidate edge $e \in Q$. Based on these approximations, we can infer that $\tilde{\Delta}(e) = \frac{\tilde{b}(e)}{1+\tilde{r}(e)}$

```
Algorithm 2: APPROX($\mathcal{G}, S_1, S_0, Q, k, \epsilon$)
  Input  : A connected graph $\mathcal{G} = (V, E)$; a candidate edge
           set $Q = (V \times V)\backslash E$; an integer $1 \le k \le |Q|$; a real
           number $\epsilon > 0$
  Output : $T$: A subset of $T$ with $|T| = k$
1 Initialize solution $T = \emptyset$
2 for $i = 1$ to $k$ do
3     $\{(e, \tilde{r}(e)) | e \in Q\} \leftarrow$ ERCOMP($\mathcal{G}, Q, \epsilon/2$)
4     $\{(e, \tilde{b}(e)) | e \in Q\} \leftarrow$ BDCOMP($\mathcal{G}, Q, \epsilon/2$)
5     Select $e_i$ s.t. $e_i \leftarrow \arg\max_{e \in Q} \tilde{\Delta}(e) = \frac{\tilde{B}_T(e)}{1+\tilde{R}_T(e)}$
6     Update solution $T \leftarrow T \cup \{e_i\}$
7     Update the graph $\mathcal{G} \leftarrow \mathcal{G}(V, E \cup \{e_i\})$
8     Update the candidate edge set $Q \leftarrow Q \backslash e_i$
9 return $T$.
```

constitutes an ϵ-approximation of $\Delta(e)$ for any $e \in Q$, that is $\tilde{\Delta}(e) \approx_\epsilon \Delta(e)$.

Building on this analysis, we propose an approximation algorithm as described in Algorithm 2. The performance of this algorithm is discussed in Theorem 3.4.

THEOREM 3.4. *For any $k > 0$ and error ϵ, Algorithm 2 operates in $\tilde{O}(km\epsilon^{-2} + k|Q|\epsilon^{-2})$ time. It produces a solution T by iteratively selecting k edges, where for the edge chosen in each round, the marginal decrease of this edge is an ϵ-approximation of the maximal marginal decrease of the Kirchhoff index.*

Despite the efficiency of the enhanced greedy algorithm, a persistent problem arises because we need to query $|Q|$ different marginal decreases. This process can be notably time-consuming, especially for sparse graphs. Although alternative methods have been proposed to approximate effective resistance [45, 48, 56, 77], the huge time cost resulted from quadratically many queries remains a challenge and is also a main limitation of the state-of-the-art algorithms [59]. Consequently, in the subsequent sections, we will explore ways to reduce the number of queries in an attempt to expedite the optimization algorithm.

4 NEARLY LINEAR TIME ALGORITHM

In the application of the heuristic greedy algorithm, pinpointing the maximum marginal decrease, $\Delta(e)$, for each $e \in Q$ in every iteration poses a significant challenge, especially when considering the candidate edge set Q can potentially scale up to $\Omega(n^2)$. This potential size underscores the pressing need for efficient pruning strategies. In the ensuing discussions, rather than directly tackling the computation or approximation of the Kirchhoff index's marginal decrease, we introduce a suite of gradient-based algorithms. Concurrently, we detail methods designed to prune the large original candidate set down to a smaller subset P, where $P \subset Q$ and $|P| \ll |Q|$, thereby facilitating fewer query operations. By leveraging these methodologies, we are able to devise efficient algorithms that yield approximate solutions to the central problem.

4.1 Transforming Decrement Search to Gradient Maximization

Traditionally, heuristic greedy algorithms select edges based on the maximal marginal decrease in the Kirchhoff index. While the marginal decrease is commonly used, the gradient or partial derivative of a function has also been explored for its ability to measure edge modification impact, as seen in [43, 65, 78]. However, its application in optimization remains limited, with notable exceptions like [82]. Several studies have examined the gradient of the Kirchhoff index [28, 43, 76, 78], which captures the significance of an edge through its potential to alter the Kirchhoff index following modification of the edge within the graph. Inspired by these insights, we adopt the gradient of the Kirchhoff index to characterize the significance of an edge instead of the marginal decrease $\Delta(e)$, and we will provide a new approach for solving this problem.

First, we show how to express the gradient of the Kirchhoff index.

Initially, we extend the simple unweighted graph $\mathcal{G} = (V, E)$ to the weighted graph $\mathcal{H} = (V, V \times V, w)$, wherein w is the edge weight function which assigns a weight of 1 to $e \in E$ and 0 otherwise. Thus, the addition of a new edge $e \in (V \times V)\backslash E$ to the graph $\mathcal{G}$ can be interpreted as altering its edge weight $w(e)$ from 0 to 1.

Subsequently, the Kirchhoff index of graph $\mathcal{H}$ can be expressed as a function of the weight w of each edge, and the gradient or partial derivative $c(e)$ for each edge $e \in (V \times V)\backslash E$ can be determined by [78],

$$c(e) = \frac{\partial K(\mathcal{H})}{\partial w(e)} = n\boldsymbol{b}_e^\top \boldsymbol{L}^{2\dagger} \boldsymbol{b}_e.$$

Since the gradient of the Kirchhoff index of each edge also measures the importance of each edge, we can naturally propose a gradient-based greedy algorithm by iteratively finding one edge with the largest gradient and adding it to the graph. The algorithm is outlined in Algorithm 3 with its time complexity being $O(n^3 + kn^2)$, by utilizing a similar analysis as algorithm DETER. Although this gradient-based algorithm GRAD does not possess a theoretical performance guarantee, subsequent empirical evidence illustrates comparable efficacy between GRAD and DETER.

Although we employ the gradient instead of the marginal decrease of the Kirchhoff index for designing the algorithm, the inherent problem of high time complexity incurred by querying $|Q|$ gradients still exists. Rather than computing the gradient $c(e)$ for all candidate edges $e \in Q$—a process that can be computationally expensive—we will propose a pruning technique using some properties based on geometry in the following sections.

REMARK 4.1. *We notice that in the original Kirchhoff index optimization problem, we select edges from the candidate edge set Q. However, we select the furthest node pairs in the gradient-based algorithm. Since we focus on sparse graphs, the furthest node pairs are not directly connected to each other.*

4.2 Primary Nearly Linear-time Solution

First, we propose a simple but efficient way to solve this problem.

4.2.1 Employing Convex Hull Approximation. For the sake of simplicity and subsequent discussions, we redefine $c(e) = \boldsymbol{b}_e^\top \boldsymbol{L}^{2\dagger} \boldsymbol{b}_e$ for any edge $e = (i, j) \in Q$, since we don't change the number of

```
Algorithm 3: GRAD($\mathcal{G}, S_1, S_0, Q, k$)
  Input  : A connected graph $\mathcal{G} = (V, E)$; a candidate edge
           set $Q = (V \times V) \backslash E$; an integer $1 \le k \le |Q|$
  Output : $T$: A subset of $T$ with $|T| = k$
1  Compute $\boldsymbol{L}^{\dagger}$
2  Compute $\boldsymbol{L}^{2\dagger}$
3  Initialize solution $T = \emptyset$
4  for $i = 1$ to $k$ do
5      Compute $\Delta(e) = K(\mathcal{G}) - K(\{e\})$ for each $e \in Q$
6      Select $e_i$ s.t. $e_i \leftarrow \arg\max_{e \in Q} c(e)$
7      Update solution $T \leftarrow T \cup \{e_i\}$
8      Update the graph $\mathcal{G} \leftarrow \mathcal{G}(V, E \cup \{e_i\})$
9      Update $\boldsymbol{L}^{\dagger} \leftarrow \boldsymbol{L}^{\dagger} - \frac{\boldsymbol{L}^{\dagger}\boldsymbol{b}_e^{\top}\boldsymbol{b}_e\boldsymbol{L}^{\dagger}}{1+\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e}$
10     Update
         $\boldsymbol{L}^{2\dagger} \leftarrow \boldsymbol{L}^{2\dagger} + \frac{\boldsymbol{L}^{\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{2\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}}{(1+\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e)^2} - \frac{\boldsymbol{L}^{2\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}}{1+\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e} - \frac{\boldsymbol{L}^{\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{2\dagger}}{1+\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e}$
11     Update the candidate edge set $Q \leftarrow Q \backslash e_i$
12 return $T$.
```

nodes in the graph. Then $c(e) = \boldsymbol{b}_e^{\top}\boldsymbol{L}^{2\dagger}\boldsymbol{b}_e = \left\|\boldsymbol{L}^{\dagger}(\boldsymbol{e}_i - \boldsymbol{e}_j)\right\|_2^2$ can be regarded as the Euclidean distance between nodes i and j. Consequently, we introduce a point set $P = \{p_1, p_2, \ldots, p_n\}$ with each point's coordinate specified as $p_i = \boldsymbol{L}^{\dagger}\boldsymbol{e}_i \in \mathbb{R}^n$. Hence, the task of maximizing the gradient of $c(e)$ is equivalent to identifying the most distant pair of points within the set P.

A naïve approach to finding the furthest point pair involves computing all pairwise distances, which is computationally costly. In examining the spatial distribution of a given set of points, we observed that certain points are, relatively speaking, located within the interior of this point set, while others lie on the exterior. It is a general observation that the distances among interior points tend to be shorter than the distances between those on the exterior. Recognizing this, our primary focus is to identify these exterior points, as doing so would facilitate faster computations in subsequent analyses. This leads us to leverage the concept of the convex hull to discern the exterior points.

Definition 4.2. *[61] Given a set of n points $P = \{p_1, p_2, \ldots, p_n\}$, its convex hull, denoted as $C(P)$, is the intersection of all convex supersets of P. Each $p \in P$ for which $p \notin C(P \backslash \{p\})$ is called an extremal point of $C(P)$. The collection of all extremal points constitutes the extremal point set $X(P) \subseteq P$.*

Remarkably, there exists a unique bijective relationship between the convex hull and the extremal point set of any given set of points. More precisely, given the convex hull of a set of points, one can unambiguously determine its corresponding extremal point set, and vice versa. Such an equivalence has profound implications, since algorithms that compute the convex hull can be immediately leveraged to identify the extremal point set, and conversely, algorithms that determine the extremal point set can be adapted to infer the convex hull. Based on this equivalence, we introduce a critical property of the convex hull, as formulated in the following lemma.

Lemma 4.3. *[61] Given a set of n points $P = \{p_1, p_2, \ldots, p_n\}$, let $C(P)$ represents the convex hull of P and $X(P)$ denotes the extremal point set of $C(P)$, we have $d(P) = d(C(P)) = d(X(P))$, where $d(\cdot)$ signifies the diameter of a set.*

In general, the extremal point set is smaller than the original point set. Leveraging the properties of the extremal point set of the convex hull, we can identify the furthest point pair by computing distances between points within the extremal point set instead of between all point pairs. This dramatically reduces computational time, particularly if the extremal points can be efficiently determined. Below, we elaborate on how to efficiently identify the extremal point set or the convex hull of a given point set.

For a set of points $P = \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^d$, the time complexity to find the convex hull of these points is $O(n^{\lfloor d/2 \rfloor})$[12, 51]. Hence, computing the convex hull in high-dimensional space is costly. Our analysis above yields n points $p_i = \boldsymbol{L}^{\dagger}\boldsymbol{e}_i \in \mathbb{R}^n$ for $i = 1, 2, \ldots, n$, signifying that finding the convex hull of these n points incurs a time complexity of $O(n^{\lfloor n/2 \rfloor})$. This is impractical due to the excessive time cost. However, an algorithm called ApproxConv can approximate the convex hull with significantly lower time complexity [4, 5, 36]. The performance of the algorithm ApproxConv is described in the following lemma:

Lemma 4.4. *[4, 5, 36] For a set of points $P = \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^d$, and a parameter $\mu \in (0, 1)$, the algorithm ApproxConv(P, μ) generates a subset $\bar{X}(P) \subseteq X(P)$ of $l = |\bar{X}(P)|$ points, where $X(P)$ is the set of extreme points of S. The algorithm has a time complexity of $O(nl(d + \mu^{-2}))$ and ensures that the Euclidean distance for any $p \in X(P)$ to the convex hull of the point set $\bar{X}(P)$ does not exceed $\mu d(P)$, where $d(P)$ represents the diameter of the point set P.*

Based on Lemma 4.4, we can conveniently obtain a point set $\bar{X}(P) \subseteq X(P)$ as an approximate extreme point set by applying the algorithm ApproxConv(P, μ) for any $\mu \in (0, 1)$. We thus have:

$$d(P) = d(X(P)) = d(C(P)) \le 2\mu d(P) + d(\bar{X}(P)),$$

implying that

$$d(P)^2 \approx_{8\mu} d(\bar{X}(P))^2. \qquad (9)$$

This approximation affords us a significant reduction in computation time, since finding the furthest point pairs from set $\bar{X}(P)$ now only needs $O(l^2 n)$ time complexity.

4.2.2 Coordinate Projection. Based on Lemma 4.4 and Equation (9), the ApproxConv algorithm can effectively generate an approximation of the extremal point set for point set P. Nonetheless, its application still requires a time complexity of $\Omega(n^2)$,since the dimension of each node is n. Given that the exclusion of points from set P isn't feasible, optimizing the ApproxConv algorithm necessitates a reduction in the dimensionality of each point. In pursuit of enhanced computational efficiency, we leverage the Johnson-Lindenstrauss (JL) lemma [1, 35]. This pivotal lemma provides a means to lower the dimensionality without significantly changing the pairwise distances between the points. Therefore, we can reduce the processing time of ApproxConv greatly.

First, we introduce the Johnson-Lindenstrauss Lemma [1].

Lemma 4.5. *Given fixed vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n \in \mathbb{R}^d$ and $\beta > 0$, let $\boldsymbol{Q}_{t \times d}$, $t \ge 24 \log n/\beta^2$, be a matrix, with each entry being either $1/\sqrt{t}$ or $-1/\sqrt{t}$ with the same probability $1/2$. Then, with a probability of* *at least* $1 - 1/n$,

$$(1-\beta)\|\boldsymbol{v}_i - \boldsymbol{v}_j\|_2^2 \le \|\boldsymbol{Q}\boldsymbol{v}_i - \boldsymbol{Q}\boldsymbol{v}_j\|_2^2 \le (1+\beta)\|\boldsymbol{v}_i - \boldsymbol{v}_j\|_2^2$$

for all pairs $i, j \le n$.

Lemma 4.5 allows the coordinates of a point p_i to be projected from an n-dimensional space to a lower-dimensional one, preserving the distances between each pair of points. Let's denote a new point set $\tilde{P} = \{\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_n\}$, where each point $\tilde{p}_i$ possesses a new coordinate $\tilde{p}_i = \boldsymbol{Q}\boldsymbol{L}^{\dagger}\boldsymbol{e}_i \in \mathbb{R}^t$ for $i = 1, 2, \ldots, n$. Then, for any i, j, we have $\left\|p_i - p_j\right\|_2^2 \approx_\beta \left\|\tilde{p}_i - \tilde{p}_j\right\|_2^2$, which leads to

$$d(P)^2 \approx_\beta d(\tilde{P})^2. \tag{10}$$

Subsequently, we can apply the ApproxConv algorithm on the point set $\tilde{P}$ with an error μ, which in turn generates a subset $\tilde{X}(\tilde{P})$ of the point set $\tilde{P}$ in $O(n\tilde{l}(\log n/\beta^2 + \mu^2))$ time complexity, where $\tilde{l} = |\tilde{X}(\tilde{P})|$. According to Equation (9), this subset meets the condition

$$d(\tilde{P})^2 \approx_{8\mu} d(\tilde{X}(\tilde{P}))^2. \tag{11}$$

By integrating Equations (10) and (11), we have

$$d(\tilde{X}(\tilde{P}))^2 \approx_{\beta+8\mu} d(P)^2, \tag{12}$$

implying that the diameter of the point set P can be approximated by projecting the coordinates of points and approximating the convex hull of the resulting points. After determining the extremal point set $\tilde{X}(\tilde{P})$, we can now find the furthest point pair from set $\tilde{X}(\tilde{P})$ in $O(\tilde{l}^2 \log n/\beta^2)$ time.

4.2.3 *Coordinate Computation.*

Given a set of points, the above analysis shows us how to approximate its diameter efficiently. However, there still exists a crucial consideration of computation of the projected coordinate. The calculation of $\boldsymbol{Q}\boldsymbol{L}^{\dagger}\boldsymbol{e}_i$ requires matrix inversion, an operation that typically demands $O(n^3)$ time for directly computing the pseudoinverse of the Laplacian matrix $\boldsymbol{L}$. To circumvent this expensive operation, we utilize the fast SDD linear system solver [15, 67], as detailed below.

Lemma 4.6. *[15, 67] Let* $\boldsymbol{T} \in \mathbb{R}^{n\times n}$ *be a semidefinite positive symmetric matrix with nonzero m entries,* $\boldsymbol{b} \in \mathbb{R}^n$ *a vector, and* $\gamma > 0$ *an error parameter. There exists a solver, denoted by* $\boldsymbol{a} = \textsc{Solve}(\boldsymbol{T}, \boldsymbol{b}, \gamma)$, *which yields a vector* $\boldsymbol{a} \in \mathbb{R}^n$ *satisfying* $\left\|\boldsymbol{a} - \boldsymbol{T}^{-1}\boldsymbol{b}\right\|_T \le \gamma\left\|\boldsymbol{T}^{-1}\boldsymbol{b}\right\|_T$. *The expected runtime of this solver is* $\tilde{O}(m\log^{0.5} n)$, *where* $\tilde{O}(\cdot)$ *suppresses* $\mathrm{poly}(\log n)$ *factors.*

Building on this solver, we can approximate the coordinates of each point $\tilde{p}_i$. Let $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{L}^{\dagger}$, and $\hat{\boldsymbol{X}}_{j,:} = \textsc{Solve}(\boldsymbol{L}, \boldsymbol{Q}_{j,:}, \gamma)$. Then, $\hat{p}_i = \hat{\boldsymbol{X}}\boldsymbol{e}_i$ becomes an approximation of the projected coordinate of point $\tilde{p}_i$ in an $O(\log n)$-dimensional space, calculated by the Laplacian solver. We now aim to show that the distances between the approximate coordinates retain an accurate approximation of the distances between the original coordinates. Lemma 4.7 verifies that this approximation can be achieved efficiently using Solve (see Lemma 4.6).

Lemma 4.7. *Assume that*

$$\|\boldsymbol{X}(\boldsymbol{e}_u - \boldsymbol{e}_v)\| \approx_\beta \|\boldsymbol{X}(\boldsymbol{e}_u - \boldsymbol{e}_v)\| \tag{13}$$

holds for every pair $u, v \in V$, *and*

$$\|\hat{\boldsymbol{X}}_{j,:} - \boldsymbol{X}_{j,:}\|_{\boldsymbol{L}} \le \gamma\|\boldsymbol{X}_{j,:}\|_{\boldsymbol{L}}, \tag{14}$$

where $\gamma \le \frac{\delta}{3n}\sqrt{\frac{6(1-\beta)}{n(n^2-1)(1+\beta)}}$, *holds for any* $j = 1, 2, \ldots, t$. *Then,*

$$\|\boldsymbol{X}(\boldsymbol{e}_u - \boldsymbol{e}_v)\|_2^2 \approx_\delta \|\hat{\boldsymbol{X}}(\boldsymbol{e}_u - \boldsymbol{e}_v)\|_2^2 \tag{15}$$

holds for every pair $u, v \in V$.

The proof is similar to the proof in [78]. I will put it in APPENDIX.

Proof. The proof proceeds by considering two bounds. First, from Equation (13), we obtain

$$\|\boldsymbol{X}(\boldsymbol{e}_u - \boldsymbol{e}_v)\|_2^2 \ge (1-\beta)\boldsymbol{b}_e^\top \boldsymbol{L}^{2\dagger}\boldsymbol{b}_e \ge (1-\beta)n^{-2},$$

which provides a lower bound for $\|\boldsymbol{X}(\boldsymbol{e}_u - \boldsymbol{e}_v)\|_2^2$.

Second, let S be a simple path between any two nodes u and v. Applying the triangle inequality, we obtain

$$\begin{aligned}
&\left|\|\boldsymbol{X}(\boldsymbol{e}_u - \boldsymbol{e}_v)\| - \left\|\hat{\boldsymbol{X}}(\boldsymbol{e}_u - \boldsymbol{e}_v)\right\|\right| \le \left\|(\boldsymbol{X} - \hat{\boldsymbol{X}})(\boldsymbol{e}_u - \boldsymbol{e}_v)\right\| \\
\le &\sum_{(a,b)\in S} \left\|(\boldsymbol{X} - \hat{\boldsymbol{X}})(\boldsymbol{e}_a - \boldsymbol{e}_b)\right\| \\
\le &\sqrt{n \sum_{(a,b)\in S} \left\|(\boldsymbol{X} - \hat{\boldsymbol{X}})(\boldsymbol{e}_a - \boldsymbol{e}_b)\right\|^2} \\
\le &\sqrt{n \sum_{(a,b)\in E} \left\|(\boldsymbol{X} - \hat{\boldsymbol{X}})(\boldsymbol{e}_a - \boldsymbol{e}_b)\right\|^2} \\
= &\sqrt{n}\left\|(\boldsymbol{X} - \hat{\boldsymbol{X}})\boldsymbol{B}^\top\right\|_F \le \sqrt{n}\gamma\left\|\boldsymbol{X}\boldsymbol{B}^\top\right\|_F \\
\le &\gamma\sqrt{n(1+\beta)\mathrm{Tr}\left(\boldsymbol{L}^{\dagger}\right)} \le \gamma\sqrt{n(n^2-1)(1+\beta)/6}.
\end{aligned}$$

The last inequality follows by the fact that $\mathrm{Tr}\left(\boldsymbol{L}^{\dagger}\right) = K(\mathcal{G})/n$, where $K(\mathcal{G})$ achieves the maximum value $n(n^2-1)/6$ when $\mathcal{G}$ is an n-node path graph [49].

Combining above two bounds together, we obtain that

$$\frac{\left|\|\boldsymbol{X}(\boldsymbol{e}_u - \boldsymbol{e}_v)\| - \left\|\hat{\boldsymbol{X}}(\boldsymbol{e}_u - \boldsymbol{e}_v)\right\|\right|}{\|\boldsymbol{X}(\boldsymbol{e}_u - \boldsymbol{e}_v)\|} \le \frac{\delta}{3}.$$

And we can further obtain that

$$\begin{aligned}
&\left|\|\boldsymbol{X}(\boldsymbol{e}_u - \boldsymbol{e}_v)\|_2^2 - \|\hat{\boldsymbol{X}}(\boldsymbol{e}_u - \boldsymbol{e}_v)\|_2^2\right| \\
= &\left|\|\boldsymbol{X}(\boldsymbol{e}_u - \boldsymbol{e}_v)\|_2 - \|\hat{\boldsymbol{X}}(\boldsymbol{e}_u - \boldsymbol{e}_v)\|_2\right| \times \\
&\left|\|\boldsymbol{X}(\boldsymbol{e}_u - \boldsymbol{e}_v)\|_2 + \|\hat{\boldsymbol{X}}(\boldsymbol{e}_u - \boldsymbol{e}_v)\|_2\right| \\
\le &\left(\frac{2\delta}{3} + \frac{\delta^2}{9}\right)\|\boldsymbol{X}(\boldsymbol{e}_u - \boldsymbol{e}_v)\|_2^2 \le \delta\,\|\boldsymbol{X}(\boldsymbol{e}_u - \boldsymbol{e}_v)\|_2^2,
\end{aligned}$$

thereby verifying Equation (15). □

Lemma 4.7 allows us to approximate the projected coordinates of each point in $\tilde{O}(m/\beta^2)$ and then constructs a set of points $\hat{P} = \{\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_n\}$ with approximate projected coordinates being $\hat{p}_i = \boldsymbol{X}\boldsymbol{e}_i \in \mathbb{R}^t$. Consequently, we obtain

$$d(\tilde{P})^2 \approx_\delta d(\hat{P})^2. \tag{16}$$

By applying the algorithm ApproxConv with the approximated projected point set $\hat{P}$ computed in $\tilde{O}(m/\beta^2)$, the algorithm returns a subset $\hat{X}(\hat{P})$ of point set $\hat{P}$ in $O(n\hat{l}(\log n/\beta^2 + \mu^2))$, where $\hat{l} = |\hat{X}(\hat{P})|$, satisfying

$$d(\hat{X}(\hat{P}))^2 \approx_{8\mu+\beta+\delta} d(P)^2. \tag{17}$$

Since the furthest point pair of set $\hat{X}(\hat{P})$ can be computed in $O(\hat{l}^2 \log n/\beta^2)$, we now can efficiently determine the furthest point pairs.

Algorithm 4: FastGrad$(\mathcal{G}, \epsilon, k)$

```
Input  : A connected graph $\mathcal{G} = (V, E)$; a real number
         $\epsilon > 0$; an integer $k$
Output : $T$: A subset of $(V \times V) \backslash E$ satisfying $|T| = k$
1  Initialize solution $T = \emptyset$
2  Set $\mu = \epsilon/24$, $\beta = \epsilon/3$, $\delta = \epsilon/3$
3  Set $\gamma = \frac{\delta}{3n}\sqrt{\frac{6(1-\beta)}{n(n^2-1)(1+\beta)}}$
4  $t = \lceil \log(n)/\beta^2 \rceil$
5  $\boldsymbol{L} \leftarrow$ the Laplacian matrix of graph $\mathcal{G}$
6  for $i = 1$ to $k$ do
7      Generate random Gaussian matrices $\boldsymbol{Q}_{t \times n}$
8      for $j = 1$ to $t$ do
9          $\boldsymbol{X}_{j,:} = $ Solve $(\boldsymbol{L}, \boldsymbol{Q}_{j,:}, \gamma)$
10     $\hat{P} = \{\hat{p}_1, \hat{p}_2, \dots, \hat{p}_n\}$ with the coordinate of each point
       being $\boldsymbol{X}_{:,1}, \boldsymbol{X}_{:,2}, \dots, \boldsymbol{X}_{:,n}$
11     $H = $ ApproxConv$(\hat{P}, \mu)$
12     $(x, y) \leftarrow \arg\max_{u,v \in H} \|\boldsymbol{X}_{:,u} - \boldsymbol{X}_{:,v}\|_2^2$
13     Update solution $T \leftarrow T \cup \{(x, y)\}$
14     Update the graph $\mathcal{G} \leftarrow \mathcal{G}(V, E \cup \{(x, y)\})$
15 return $T$.
```

4.2.4 Approximation Algorithm. Equipped with the convex hull approximation and coordinate projection, we now delineate an efficient algorithm for Problem 1, as detailed in Algorithm 3.

The structure of Algorithm 3 consists of k iterative rounds (Lines 5-13), each focused on the selection of an individual edge. The steps within each round are systematically organized as follows:

(1) Random matrix generation: A random matrix $\boldsymbol{Q}_{t \times n}$ is constructed in $O(n \log n\epsilon^{-2})$ (Line 6).
(2) Coordinate approximation: Utilizing the JL lemma and the Laplacian solver, the approximated projected coordinates of points $\hat{p}_1, \hat{p}_2, \dots, \hat{p}_n$ are computed in $\tilde{O}(m\epsilon^{-2})$ time (Lines 7-9).
(3) Extremal point determination: The extremal point set of the point set $\hat{P}$ is computed in $O(n \log n\epsilon^{-2}\hat{l})$ time (Line 10).
(4) Maximal distance finding: The maximal distance between the points within the extremal point set is identified in $O(\hat{l}^2 \log n\epsilon^{-2})$ time (Line 11).
(5) Solution and graph update: Finally, the solution and graph are updated based on the obtained results (Lines 12-13).

Hence, the total time complexity of Algorithm 4 is $\tilde{O}(kn\hat{l}/\epsilon^2)$. The following theorem rigorously characterizes the performance of Algorithm 4.

Theorem 4.8. *Given any positive integer k and an error parameter $\epsilon \in (0, 1)$, the run-time of Algorithm 3 is $\tilde{O}(kn\hat{l}/\epsilon^2)$. Through the iterative selection of k edges, the algorithm produces a solution T. Furthermore, the gradient of the edge selected in each round constitutes an ϵ-approximation of the maximal gradient of the objective function.*

4.3 Faster Algorithm by Pre-pruning and Update

While the Algorithm 4 shows significant efficiency, we identify certain redundancies. The most time-consuming steps in each iteration are step (2), computing the approximated projected coordinates, and step (3), invoking the ApproxConv routine. In what follows, we introduce improvements to accelerate the algorithm by addressing these two aspects.

4.3.1 Fast Update of the Coordinate. Repeatedly calling the Laplacian solver $t = \lceil \log n/\epsilon^2 \rceil$ times to compute the approximated projected coordinates in each iteration is a significant computational overhead. We recognize that after selecting an edge in each iteration, we can efficiently update the coordinate of each node by employing the Sherman-Morrison formula [53] rather than recomputing them.

For each node $i \in V$, its projected coordinate is $\boldsymbol{Q}\boldsymbol{L}^{\dagger}\boldsymbol{e}_i$. After adding edge $e = (x, y)$ to the graph, the coordinate updates to $\boldsymbol{Q}(\boldsymbol{L}+\boldsymbol{b}_e\boldsymbol{b}_e^{\top})^{\dagger}\boldsymbol{e}_i = \boldsymbol{Q}\boldsymbol{L}^{\dagger}\boldsymbol{e}_i - \frac{\boldsymbol{Q}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{e}_i}{\boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}\boldsymbol{b}_e}$ according to Sherman-Morrison formula. Instead of re-approximating the projected coordinates, we only need to compute $\boldsymbol{x} = \boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}$ once, then we can update the coordinate of each point to be $\boldsymbol{Q}\boldsymbol{L}^{\dagger}\boldsymbol{e}_i - \frac{\boldsymbol{Q}\boldsymbol{x}\boldsymbol{x}^{\top}\boldsymbol{e}_i}{\boldsymbol{b}_e^{\top}\boldsymbol{x}}$ for each $i \in V$ by matrix manipulation. Direct computation of vector $\boldsymbol{x} = \boldsymbol{b}_e^{\top}\boldsymbol{L}^{\dagger}$ relies on matrix inversion and is time-consuming. So we use the Laplacian solver a single time in $\tilde{O}(m)$ to get an approximation of the vector $\boldsymbol{x}$ by $\tilde{\boldsymbol{x}} = $ Solve$(\boldsymbol{L}, \boldsymbol{b}_e, \delta_0)$, where δ_0 can be sufficiently small for accuracy. Consequently, the coordinates of the nodes can be then updated efficiently in $O(n \log n/\epsilon^{-2})$ time, rather than using Laplacian solver for $\lceil \log n\epsilon^{-2} \rceil$ times.

Remark 4.9. *Although the use of the Laplacian solver may lead to some inaccuracy, it has very accurate performance in practice since δ_0 can be very small. Also, we only use the Laplacian solver once in each iteration. So we believe that this update process does not lead to more errors.*

4.3.2 Preprocessing Techniques for Extremal Point Selection. The time complexity of approximating the extremal point set of the convex hull is correlated with the number of nodes in the network. As the extremal point set consists of a small-sized subset of the total n points, our aim is to prune as many non-extremal points as possible before calling the ApproxConv routine, thereby saving time. It seems infeasible to directly remove some points from the point set, however, by combining the information of the graph structure, we will prune as many internal points as possible.

An essential property of extremal points is that the point furthest from any given point must itself be an extremal point. The distance considered here refers to the Euclidean distance in space, correlating with the biharmonic distance on graphs. Determining whether a node is or is not the furthest node against any other node on the graph is time-consuming. To expedite this, we employ node centrality with two prerequisites: (1) it encapsulates the graph's furthest distance information, (2) it can be computed efficiently.

The eccentricity centrality, related to the shortest path distance, serves our purpose. If any node's eccentricity is smaller than any one of its neighbor's eccentricity, we think this node is a central node and we prune this node. Reference [44] furnishes a linear-time algorithm EccComp($\mathcal{G}$) for calculating the eccentricities of all nodes. Since this step prunes those inner nodes, it does not influence the approximate accuracy of our algorithm. Moreover, after adding a few edges to the graph, the central nodes shouldn't be affected substantially and remain central, so we can simply preprocess this step once to prune those central nodes before using algorithm ApproxConv.

Some prior research suggests that node pairs with maximal effective resistance decrease the Kirchhoff index of a graph significantly [71, 72]. Other studies sample nodes based on their diagonal entries in $\boldsymbol{L}^\dagger$, an effective resistance-based metric that captures the total effective resistances between the node and all other nodes [59]. However, the focus on the largest distance in the convex hull does not match this approach well. Moreover, the computational expense of calculating $\boldsymbol{L}^\dagger$ diagonal values outweighs that of eccentricities [3, 44]. Hence, to avoid excessive time on preprocessing, we select eccentricities as the key metric to prune central nodes.

4.3.3 Faster Algorithm. Armed with techniques of preprocessing of the convex hull approximation and coordinate update, we are now in position to propose a more efficient algorithm for Problem 1, which is depicted in Algorithm 5.

The structure of Algorithm 5 consists of a preprocessing step (Lines 1-13) and k iterative rounds (Lines 14-23). It first computes the eccentricities of all nodes (Line 5) in $O(m)$ time and then selects nodes that are not the central nodes (Lines 6-9) in $O(m)$ time. Then it computes the approximate projected coordinates of each node in $\tilde{O}(m\epsilon^{-2})$ time. Next, the steps within each round are systematically organized as follows:

(1) Extremal point determination: The extremal point set of the point set $\hat{P}$ is determined in $O(n \log n\epsilon^{-2}\hat{l})$ time (Line 15-16).
(2) Maximal distance finding: The maximal distance between the points within the extremal point set is identified in $O(\hat{l}^2 \log n\epsilon^{-2})$ time (Line 17).
(3) Coordinates, solution and graph update: Finally, the coordinates are updated (Line 18-20) in $\tilde{O}(m+n\epsilon^{-2})$ and solution and graph are updated based on the obtained results (Lines 18-23).

Hence, the total time complexity of Algorithm 5 is meticulously computed to be $\tilde{O}(m\epsilon^{-2} + kn\hat{l}/\epsilon^2 + km)$, which is much lower than the time complexity of Algorithm 4. The following theorem rigorously characterizes the performance of Algorithm 5.

Theorem 4.10. *Given any positive integer k and an error parameter $\epsilon \in (0, 1)$, Algorithm 5 executes in $\tilde{O}(m\epsilon^{-2} + kn\hat{l}/\epsilon^2 + km)$ time. Through the iterative selection of k edges, the algorithm produces a solution T. Furthermore, the gradient of the edge selected in each round constitutes an ϵ-approximation of the maximal gradient of the objective function.*

Since our pruning and updating process do not influence the accuracy of our algorithm while it can greatly reduce some time complexity, Algorithm FastGrad+ is more efficient than Algorithm FastGrad and they achieve similar performance.

Algorithm 5: FastGrad+($\mathcal{G}, \epsilon, k$)

```
Input  : A connected graph $\mathcal{G} = (V, E)$; a real number $\epsilon > 0$; an integer $k$
Output : $T$: A subset of $(V \times V)\backslash E$ satisfying $|T| = k$
1  Initialize solution $T = \emptyset$
2  Set $\mu = \epsilon/24$, $\beta = \epsilon/3$, $\delta = \epsilon/3$
3  Set $\gamma = \frac{\delta}{3n}\sqrt{\frac{6(1-\beta)}{n(n^2-1)(1+\beta)}}$
4  $t = \lceil \log(n)/\beta^2 \rceil$
5  $\{(i, p_i)\} \leftarrow$ EccComp($\mathcal{G}$)
6  $Y = \emptyset$
7  for $i \in V$ do
8      if $p_i \geq p_j, j \in N_i$ then
9          $Y \leftarrow Y \cup \{i\}$
10 $\boldsymbol{L} \leftarrow$ the Laplacian matrix of graph $\mathcal{G}$
11 Generate random Gaussian matrices $\boldsymbol{Q}_{t\times n}$
12 for $j = 1$ to $t$ do
13     $\boldsymbol{X}_{i,:} =$ Solve $(\boldsymbol{L}, \boldsymbol{Q}_{i,:}, \gamma)$
14 for $i = 1$ to $k$ do
15     $\hat{P} = \{\hat{p}_i | i \in Y\}$ with the coordinate of each point being $\boldsymbol{X}_{:,i}$
16     $H =$ ApproxConv($\hat{P}, \mu$)
17     $(x, y) \leftarrow \arg\max_{u,v\in H} \|\boldsymbol{X}_{:,u} - \boldsymbol{X}_{:,v}\|_2^2$
18     $\boldsymbol{y} =$ Solve $(\boldsymbol{L}, \boldsymbol{e}_x - \boldsymbol{e}_y, \gamma)$
19     for $i \in Y$ do
20         $\boldsymbol{X}_{i,:} \leftarrow \boldsymbol{X}_{i,:} - \frac{\boldsymbol{Q}\boldsymbol{y}\boldsymbol{y}^\top \boldsymbol{e}_i}{\boldsymbol{y}^\top(\boldsymbol{e}_x - \boldsymbol{e}_y)}$
21     Update solution $T \leftarrow T \cup \{(x, y)\}$
22     Update the graph $\mathcal{G} \leftarrow \mathcal{G}(V, E \cup \{(x, y)\})$
23     Update the Laplacian matrix $\boldsymbol{L}$
24 return $T$.
```

4.4 Convex Hull Approximation for Only One Time

Although we have reduced the time complexity of our algorithm by fast update of the coordinate and pruning the inner nodes, we still need to use the algorithm ApproxConv in each iteration, which takes much time. Since we add a small number of edges to the graph, the coordinate of each node does not change significantly during the edge adding process, which means that the extremal point set also does not change substantially after edge addition. So, we can compute the convex hull for only one time to get the extremal point set, then find edges from this reduced set for k iterations. This is the most efficient algorithm proposed in the present paper, and the performance is illustrated in the following theorem.

Theorem 4.11. *Given any positive integer k and an error parameter $\epsilon \in (0, 1)$, Algorithm 6 executes in $\tilde{O}(m\epsilon^{-2} + n\hat{l}\epsilon^{-2} + km\epsilon^{-2})$ time. Through the iterative selection of k edges, the algorithm produces a solution T.*

```
Algorithm 6: OneConv($\mathcal{G}, \epsilon, k$)
Input   : A connected graph $\mathcal{G} = (V, E)$; a real number $\epsilon > 0$; an integer $k$
Output  : $T$: A subset of $(V \times V) \backslash E$ satisfying $|T| = k$
1  Initialize solution $T = \emptyset$
2  Set $\mu = \epsilon/24, \beta = \epsilon/3, \delta = \epsilon/3$
3  Set $\gamma = \frac{\delta}{3n}\sqrt{\frac{6(1-\beta)}{n(n^2-1)(1+\beta)}}$
4  $t = \lceil \log(n)/\beta^2 \rceil$
5  $\boldsymbol{L} \leftarrow$ the Laplacian matrix of graph $\mathcal{G}$
6  Generate random Gaussian matrices $\boldsymbol{Q}_{t\times n}$
7  for $j = 1$ to $t$ do
8      $\boldsymbol{X}_{i,:} = $ Solve $(\boldsymbol{L}, \boldsymbol{Q}_{i,:}, \gamma)$
9  $\hat{P} = \{\hat{p}_i | i \in Y\}$ with the coordinate of each point being $\boldsymbol{X}_{:,i}$
10 $H = $ ApproxConv$(\hat{P}, \mu)$
11 for $i = 1$ to $k$ do
12     $(x, y) \leftarrow \arg\max_{u,v\in H} \left\|\boldsymbol{X}_{:,u} - \boldsymbol{X}_{:,v}\right\|_2^2$
13     $\boldsymbol{y} = $ Solve $(\boldsymbol{L}, \boldsymbol{e}_x - \boldsymbol{e}_y, \gamma)$
14     for $i \in H$ do
15         $\boldsymbol{X}_{i,:} \leftarrow \boldsymbol{X}_{i,:} - \frac{\boldsymbol{Q}\boldsymbol{y}\boldsymbol{y}^\top \boldsymbol{e}_i}{\boldsymbol{y}^\top(\boldsymbol{e}_x - \boldsymbol{e}_y)}$
16     Update solution $T \leftarrow T \cup \{(x, y)\}$
17     Update the graph $\mathcal{G} \leftarrow \mathcal{G}(V, E \cup \{(x, y)\})$
18     Update the Laplacian matrix $\boldsymbol{L}$
19 return $T$.
```

5 EXPERIMENTS

5.1 Experiment Setup

Data Description. We utilize ten datasets of undirected networks from the Network Repository [62] and SNAP [39]. Our experiments focus on the largest connected component (LCC) of these networks. Table 1 provides a detailed statistical overview of the LCC for each network. Notably, the bigger LCC encompasses over 5 million nodes and 12 million edges.

Table 1: Networks and the numbers of nodes and edges of their LCC.

Networks	Nodes	Edges	Networks	Nodes	Edges
EmailUniv	1133	5451	EmailEU	32430	54397
BitcoinAlpha	3783	24186	Douban	154908	327162
Gnutella08	6301	20777	Delicious	536108	1365961
Government	7057	89429	YoutubeSnap	1134890	2987624
Anybeat	12645	67053	DBLP	5624219	12282055

Experimental Setup. All experiments were executed on a Linux server equipped with a 4.2 GHz Intel i9-9900K CPU and 128G memory. We restricted our experiments to a single-threaded environment. The experiments were implemented using the *Julia* programming language. The codebase for our experiments is publicly accessible at https://anonymous.4open.science/r/kirchhoffmin-3579.

State-of-the-Art Algorithm Implementation. The work of [59] introduced several algorithms tailored for the Kirchhoff index optimization problem. Among these, the ColStochJLT algorithm

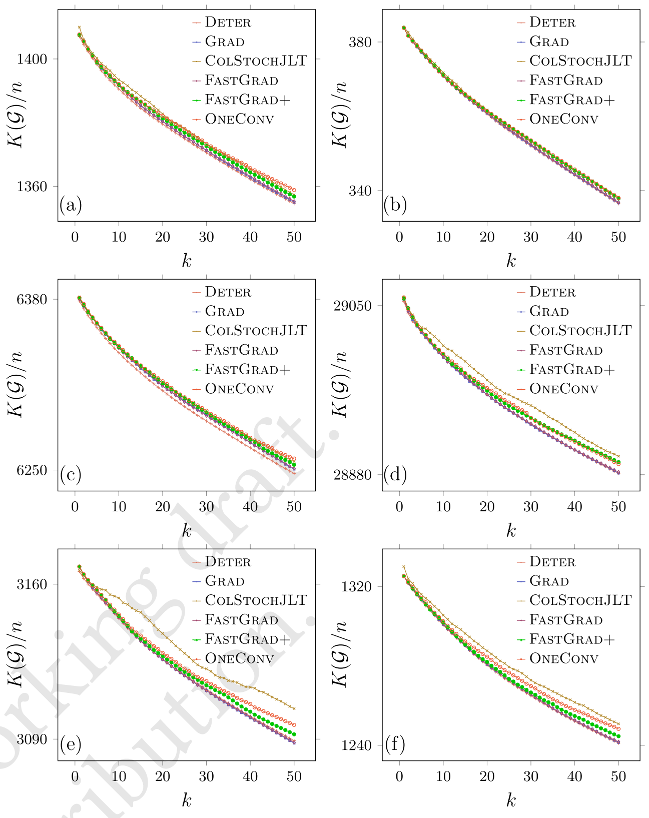

Figure 2: Kirchhoff index divided by n returned by all algorithms for $k = 1, 2, \ldots, 50$ on six small networks, BitcoinAlpha (a), EmailUniv (b), Anybeat (c), EmailEU (d), Gnutella08 (e), and Government (f).

stands out as the state-of-the-art, boasting superior runtime and performance metrics in comparison to other baseline algorithms. We adopted the settings and configurations as delineated in their article [59] for our experiments. Subsequent sections will present the performance results of the ColStochJLT algorithm.

Edge Addition Strategy Comparison. We compare the performance of various algorithms: Deter, Grad, FastGrad, FastGrad+, OneConv, and ColStochJLT. It is worth noting that we have excluded the Approx algorithm from this comparison. Although it is an enhancement of Deter, it does not address the computational challenge at hand.

Results. To evaluate the performance of different algorithms, we need to compute the Kirchhoff index for different networks. Computing the Kirchhoff index directly can not be used on large graphs. So, we use the Hutchinson's Monte-Carlo method [33] and the Laplacian solver [15, 67] to get an approximation of the Kirchhoff index as [63] did.

5.2 Algorithms Comparison

We investigate the impact of the number of added edges, denoted as k, on the performance of our proposed algorithms. We vary k in the range $1, 2, \ldots, 50$, setting parameters $\mu = 0.01$, $\beta = 0.1$,

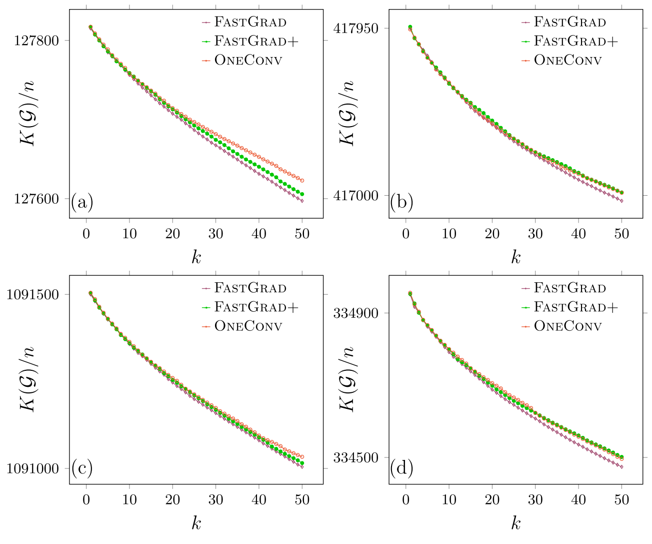

Figure 3: Kirchhoff index divided by n returned by algorithms FastGrad, FastGrad+, and OneConv, for $k = 1, 2, \ldots, 50$ on four large networks, Douban (a), Delicious (b), YoutubeSnap (c), and DBLP (d).

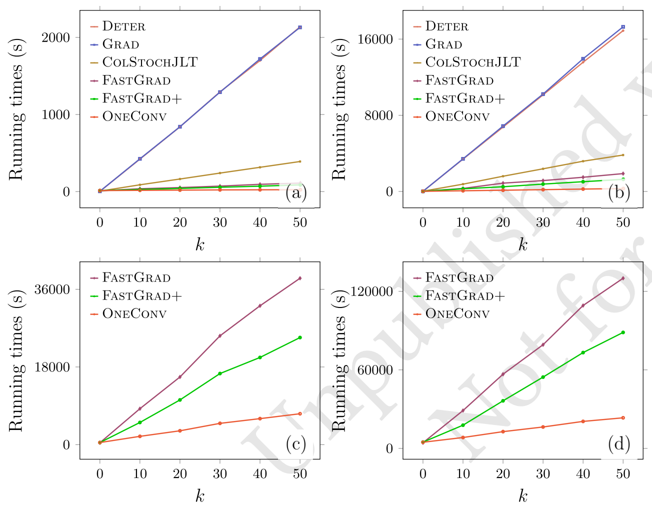

Figure 4: Running time of different algorithms for $k = 10, 20, \ldots, 50$ on four networks, Anybeat (a), EmailEU (b), YoutubeSnap (c), and DBLP (d).

and $\gamma = 0.1$ without directly specifying ϵ. For smaller networks (those with fewer than 100,000 nodes), we execute all algorithms and present their performance in Figure 2. Notably, the gradient-based algorithm Grad exhibits performance comparable to the standard greedy algorithm Deter. This observation suggests that identifying a single edge with the highest gradient is an effective strategy for the Kirchhoff index optimization problem. Furthermore, our newly introduced algorithms, namely FastGrad, FastGrad+, and OneConv, demonstrate comparable performance. For larger networks, those with more than 100,000 nodes, ColStochJLT is infeasible due to its prohibitive computational time, as noted in [59]. In contrast, our algorithms remain operational. As depicted in Figure 3, all three of these algorithms exhibit analogous performance metrics.

Figure 4 delineates the runtime of various algorithms across four networks for k values ranging from 10 to 50 in increments of 10. Among the algorithms, our proposed methods consistently outperform in terms of computational efficiency, with OneConv emerging as the most time-efficient. Specifically, on the largest network, which comprises over 5 million nodes and 12 million edges, FastGrad requires approximately 36 hours, FastGrad+ takes around 24 hours, while OneConv completes in a mere 3 hours. It's worth noting that both FastGrad+ and OneConv demand more preprocessing time compared to FastGrad, yet they compensate with reduced overall runtime.

Our empirical findings align well with our theoretical analyses. Given that FastGrad possesses the highest time complexity, it's unsurprising that it demands the longest runtime. Conversely, OneConv, with its minimal time complexity, ensures the quickest execution. However, while we offer error guarantees for FastGrad, OneConv does not provide such guarantees. This trade-off elucidates why FastGrad delivers superior performance, whereas OneConv boasts the shortest runtime.

5.3 Impact of Various Parameters

Among the parameters in our proposed algorithms, ϵ stands out as pivotal, playing a decisive role in both the efficiency and effectiveness of the algorithms. Specifically, the coordinate projection process and the convex hull determination process are predominantly influenced by this parameter. In the following sections, we delve into the effects of varying ϵ on the experimental outcomes.

Impact of Varying μ. Our initial set of experiments focuses on the parameter μ, which governs the convex hull determination process. Keeping $\beta = 0.1$, $\gamma = 0.1$ and $k = 50$ constant, we vary μ in the range $0.01, 0.02, \ldots, 0.05$. Using the marginal decrease Δ produced by the Deter algorithm as a reference, we compute the marginal decrease Δ' for the algorithms FastGrad, FastGrad+, and OneConv across the aforementioned μ values. The ratio Δ'/Δ serves as a metric to gauge the quality of these algorithms. The quality assessments for different algorithms are visualized in Figure 5. The following trend emerges: as μ increases, the performance of all three algorithms deteriorates. This decline is particularly pronounced for OneConv, which determines the convex hull only once. Consequently, a suboptimal solution from OneConv can significantly degrade the overall performance. Additionally, we chart the runtimes of these algorithms for varying μ values in Figure 6. Interestingly, as μ grows, the runtime for all algorithms decreases. However, drawing a direct correlation between the runtime of the subroutine ApproxConv and the parameter μ is non-trivial, given that the number of extremal points also influences ApproxConv's runtime.

Impact of Varying β. Subsequently, we turn our attention to the parameter β, which dictates the precision of the coordinate projection process. With $\mu = 0.01$, $\gamma = 0.1$ and $k = 50$ held constant, we adjust β over the range $0.1, 0.2, \ldots, 0.5$. The performance metrics for the different algorithms, as influenced by these β values, are

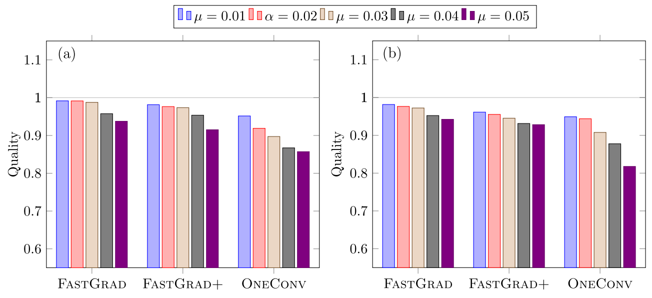

Figure 5: Qualities of algorithms FastGrad, FastGrad+, and OneConv, with $\mu = 0.01, 0.02, \ldots, 0.05$ on two networks, Anybeat (a), and EmailEU (b).

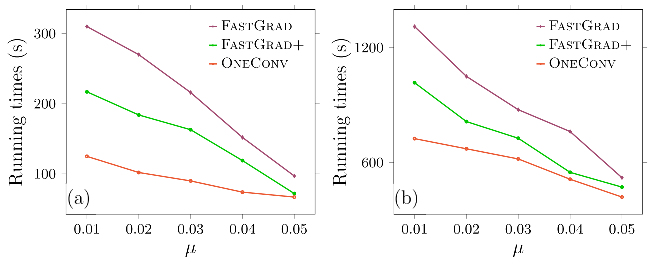

Figure 6: Running time of different algorithms for $\mu = 0.01, 0.02, \ldots, 0.05$ on two networks, Anybeat (a), and EmailEU (b).

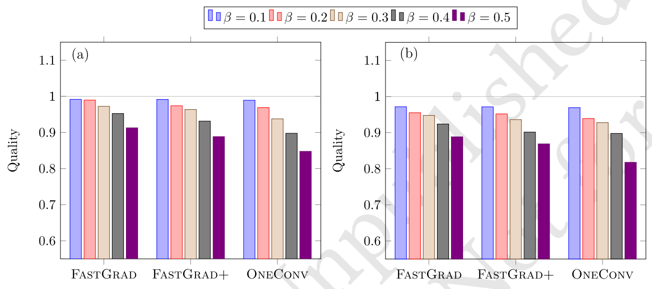

Figure 7: Qualities of algorithms FastGrad, FastGrad+, and OneConv, with $\beta = 0.1, 0.2, \ldots, 0.5$ on two networks, Anybeat (a), and EmailEU (b).

illustrated in Figure 7. A clear pattern emerges: as β increases, the efficacy of all three algorithms diminishes. Furthermore, the runtimes of these algorithms for the respective β values are captured in Figure 8. Interestingly, as β ascends, there's a corresponding reduction in the runtime for all algorithms. This inverse quadratic relationship between runtime and β aligns with the time complexity analysis presented earlier.

Exploring the Impact of γ. The parameter γ is instrumental in determining the precision of the Laplacian solver. However, upon examination, we observed that γ exerts minimal influence on both the performance and runtime of the algorithms under consideration. Consequently, we have opted not to present detailed experimental results pertaining to this parameter.

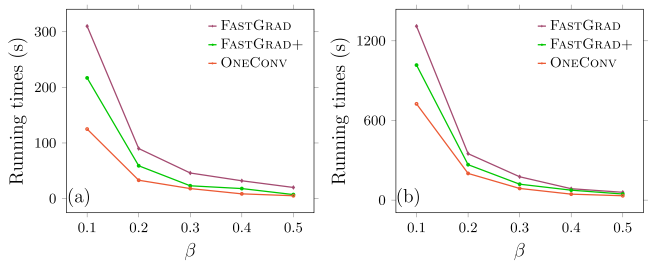

Figure 8: Running time of different algorithms for $\beta = 0.1, 0.2, \ldots, 0.5$ on two networks, Anybeat (a), and EmailEU (b).

6 RELATED WORK

Efficient computation of effective resistances is essential in a large range of applications. This has initiated the introduction of many algorithms for estimating the resistance distances between node pairs. A method anchored in random projection was introduced in [52, 66], targeting the resistance distances between endpoints of each edge, which was subsequently optimized in [30] through the integration of Wilson's algorithm [73]. The work presented in [57] uses a localized algorithm to deduce pairwise effective resistance, leveraging random walk samplings and the formulation of random spanning trees. Moreover, Monte Carlo techniques were employed in [46, 77] to augment the efficacy of preceding algorithms, as delineated in [57]. While the present landscape offers a suite of methods for computing resistance distances between individual nodes, the task of computing the Kirchhoff index remains challenging, necessitating calculations involving resistance distances for quadratically many node pairs.

A significant body of research has delved into the problem of adding new edges to graphs to minimize the Kirchhoff index [28, 37]. The authors in [38] tackled the NP-hard problem of selecting a maximum of k edges for this purpose. Interestingly, their proof establishes the NP-hardness of the Kirchhoff index optimization problem via a reduction from the well-known 3-colorability problem. Several prior studies, including [9, 31, 59, 68, 76], have investigated the optimization of the Kirchhoff index by the addition of a predetermined number of edges, from an algorithmic perspective. Initial greedy algorithms proposed in this domain exhibited a time complexity of $O(kn^3)$ [68, 76]. However, advancements by [59] have since whittled this down to a more efficient $\tilde{O}(n^2 + km)$. In a separate line of work, [28] delved into the problem of determining edge weights in an existing weighted network to curtail the total effective resistance. Drawing from this, the authors of [72] formulated both upper and lower bounds for total effective resistance, considering both edge addition and deletion operations. They further proposed four distinct strategies optimized for adding or deleting a singular edge, thus modulating the total effective resistance. The problem of optimally introducing a single edge to graphs, defined by a set number of nodes and diameter, to minimize total effective resistance was carefully investigated in [23]. However, a notable drawback of these methodologies remains their limited scalability, especially to large-scale graphs with strong expansion properties.

Besides the Kirchhoff index, several novel modifications to it have emerged over recent years. Notably, these include the multiplicative degree-Kirchhoff index [13] and the additive degree-Kirchhoff index [29]. For a given graph $\mathcal{G} = (V, E)$, the indices are articulated as $K^*(\mathcal{G}) = \frac{1}{2}\sum_{i,j\in\mathcal{V}}(d_i d_j)r_{ij}$ for the multiplicative variant, and $K^+(\mathcal{G}) = \frac{1}{2}\sum_{i,j\in\mathcal{V}}(d_i + d_j)r_{ij}$ for the additive counterpart. A notable revelation from the literature is the equivalency of the multiplicative degree-Kirchhoff index $K^*(\mathcal{G})$ of a graph $\mathcal{G}$ to four times the edge count in graph $\mathcal{G}$ multiplied by the graph's Kemeny constant [13]. The Kemeny constant itself boasts of diverse applications across various domains [32, 75]. For instance, it's harnessed as a yardstick for gauging user navigation efficiency within the vast expanse of the World Wide Web [40]. Further applications extend to measuring surveillance efficiency in robotic networks [54] and characterizing the noise robustness intrinsic to specific formation control protocols [34].

In practical graph editing, the operation of edge addition has been ubiquitously employed, catering to a myriad of application purposes. Noteworthy instances include enhancing the centrality of a particular node [16, 17, 63], amplifying the quantity of spanning trees [41], and optimizing overall opinion [83]. Additionally, it has been harnessed to mitigate polarization and disagreement within opinion dynamics [84]. Recent forays into graph theory have witnessed the edge addition operation's utility in reducing the network diameter [2, 25]. Parallel endeavors have explored graph augmentation through edge additions, targeting diverse goals such as bolstering algebraic connectivity [27]. However, it's imperative to note that these existing methodologies and approaches aren't directly translatable to the distinct challenge of minimizing the Kirchhoff index through edge addition.

7 CONCLUSION

In this paper, We studied the Kirchhoff index minimizing problem by adding k new edges. We proved that the objective function is not supermodular, and then we proposed greedy algorithms to solve this problem with error guarantees based on the bounds of the submodularity ratio and the curvature. Then, we introduced a gradient-based greedy algorithm as a new paradigm to solve this problem. We leveraged geometric properties, convex hull approximation, and approximation of the projected coordinate of each point, to provide an efficient algorithm with an error guarantee. We used pre-pruning and fast update techniques to further reduce the time complexity. We also proposed an algorithm by using the convex hull approximation for only once. Our proposed algorithms have linear time complexity. Extensive experimental results on ten real-life networks demonstrated that our proposed algorithms outperform the state-of-the-art methods in terms of efficiency and effectiveness. Moreover, our fast algorithms are scalable to large graphs with over 5 million nodes and 12 million edges.

There are several potential avenues for future work. It would be interesting to leverage other novel approximation and sparsification techniques to speed up our proposed algorithms further while preserving the accuracy guarantee. Furthermore, we believe that the proof techniques developed in the present work can be exploited to provide more efficient algorithms for other relevant network-based optimization problems. In particular, our techniques could prove useful for tackling the problems of optimizing Kemeny's constant and resistance distance.

REFERENCES

[1] Dimitris Achlioptas. 2001. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM, 274–281.

[2] Florian Adriaens and Aristides Gionis. 2022. Diameter minimization by shortcutting with degree constraints. In *2022 IEEE International Conference on Data Mining (ICDM)*. IEEE, 843–848.

[3] 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*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[4] Pranjal Awasthi, Bahman Kalantari, and Yikai Zhang. 2018. Robust vertex enumeration for convex hulls in high dimensions. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1387–1396.

[5] Pranjal Awasthi, Bahman Kalantari, and Yikai Zhang. 2020. Robust vertex enumeration for convex hulls in high dimensions. *Annals of Operations Research* 295, 1 (2020), 37–74.

[6] RB Bapat. 1999. Resistance distance in graphs. *Math. Student* 68, 1-4 (1999), 87–98.

[7] Régis Behmo, Nikos Paragios, and Véronique Prinet. 2008. Graph commute times for image representation. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–8.

[8] Andrew An Bian, Joachim M Buhmann, Andreas Krause, and Sebastian Tschiatschek. 2017. Guarantees for greedy maximization of non-submodular functions with applications. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 498–507.

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

[10] Ulrik Brandes and Daniel Fleischer. 2005. Centrality Measures Based on Current Flow. In *Proceedings of 22nd Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 533–544.

[11] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Trans. Knowl. Data. Eng.* 30, 9 (2018), 1616–1637.

[12] Bernard Chazelle. 1993. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry* 10, 4 (1993), 377–409.

[13] Haiyan Chen and Fuji Zhang. 2007. Resistance distance and the normalized Laplacian spectrum. *Discrete Appl. Math.* 155, 5 (2007), 654–661.

[14] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. 2011. Electrical Flows, Laplacian Systems, and Faster Approximation of Maximum Flow in Undirected Graphs. In *Proc. 43rd Annu. ACM Symp. Theory Comput.* San Jose, CA, USA, 273–282.

[15] Michael B Cohen, Rasmus Kyng, Gary L Miller, Jakub W Pachocki, Richard Peng, Anup B Rao, and Shen Chen Xu. 2014. Solving SDD linear systems in nearly $m \log^{1/2} n$ time. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*. ACM, 343–352.

[16] Pierluigi Crescenzi, Gianlorenzo D'angelo, Lorenzo Severini, and Yllka Velaj. 2016. Greedily improving our own closeness centrality in a network. *ACM Transactions on Knowledge Discovery from Data* 11, 1 (2016), 9.

[17] Gianlorenzo D'Angelo, Martin Olsen, and Lorenzo Severini. 2019. Coverage centrality maximization in undirected networks. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, Vol. 33. 501–508.

[18] Abhimanyu Das and David Kempe. 2011. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the 28th International Conference on Machine Learning*. Omnipress, 1057–1064.

[19] Florian Dörfler and Francesco Bullo. 2013. Kron Reduction of Graphs With Applications to Electrical Networks. *IEEE Transatcions on Circuits and Systems—I. Regular Papers* 60, 1 (2013), 150–163.

[20] Florian Dörfler, John W Simpson-Porco, and Francesco Bullo. 2018. Electrical Networks and Algebraic Graph Theory: Models, Properties, and Applications. *Proc. IEEE* 106, 5 (2018), 977–1005.

[21] Peter G Doyle and J Laurie Snell. 1984. *Random Walks and Electric Networks*. Mathematical Association of America.

[22] Wendy Ellens and Robert E Kooij. 2013. Graph measures and network robustness. *arXiv preprint arXiv:1311.5064* (2013).

[23] W Ellens, FM Spieksma, P Van Mieghem, A Jamakovic, and RE Kooij. 2011. Effective graph resistance. *Linear Algebra Appl.* 435, 10 (2011), 2491–2506.

[24] Francois Fouss, Alain Pirotte, J-M Renders, and Marco Saerens. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering* 19, 3 (2007), 355–369.

[25] Fabrizio Frati, Serge Gaspers, Joachim Gudmundsson, and Luke Mathieson. 2015. Augmenting graphs to minimize the diameter. *Algorithmica* 72 (2015), 995–1010.

[26] Scott Freitas, Diyi Yang, Srijan Kumar, Hanghang Tong, and Duen Horng Chau. 2022. Graph vulnerability and robustness: A survey. *IEEE Transactions on Knowledge and Data Engineering* 35, 6 (2022), 5915–5934.

[27] Arpita Ghosh and Stephen Boyd. 2006. Growing well-connected graphs. In *Proceedings of the 45th IEEE Conference on Decision and Control*. IEEE, 6605–6611.

[28] Arpita Ghosh, Stephen Boyd, and Amin Saberi. 2008. Minimizing effective resistance of a graph. *SIAM Rev.* 50, 1 (Feb. 2008), 37–66.

[29] Ivan Gutman, Linhua Feng, and Guihai Yu. 2012. Degree resistance distance of unicyclic graphs. *Trans. Comb.* 1, 2 (June 2012), 27–40.

[30] Takanori Hayashi, Takuya Akiba, and Yuichi Yoshida. 2016. Efficient algorithms for spanning tree centrality. In *IJCAI*. 3733–3739.

[31] Guixian Huang, Weihua He, and Yuanyao Tan. 2019. Theoretical and computational methods to minimize Kirchhoff index of graphs with a given edge k-partiteness. *Appl. Math. Comput.* 341 (2019), 348–357.

[32] Jeffrey J Hunter. 2014. The role of Kemeny’s constant in properties of Markov chains. *Communications in Statistics — Theory and Methods* 43, 7 (2014), 1309–1321.

[33] MF Hutchinson. 1989. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation* 18, 3 (1989), 1059–1076.

[34] Ali Jadbabaie and Alex Olshevsky. 2019. Scaling Laws for Consensus Protocols Subject to Noise. *IEEE Trans. Autom. Control* 64, 4 (April 2019), 1389–1402.

[35] William B Johnson and Joram Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space. *Contemp. Math.* 26 (1984), 189–206.

[36] Bahman Kalantari. 2015. A characterization theorem and an algorithm for a convex hull problem. *Annals of Operations Research* 226 (2015), 301–349.

[37] Douglas J Klein and Milan Randić. 1993. Resistance distance. *Journal of Mathematical Chemistry* 12, 1 (1993), 81–95.

[38] Robert E Kooij and Massimo A Achterberg. 2023. Minimizing the effective graph resistance by adding links is NP-hard. *arXiv preprint arXiv:2302.12628* (2023).

[39] Jure Leskovec and Rok Sosič. 2016. SNAP: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology* 8, 1 (2016), 1.

[40] Mark Levene and George Loizou. 2002. Kemeny’s Constant and the Random Surfer. *The American Mathematical Monthly* 109, 8 (2002), 741–745.

[41] Huan Li, Stacy Patterson, Yuhao Yi, and Zhongzhi Zhang. 2019. Maximizing the number of spanning trees in a connected graph. *IEEE Transactions on Information Theory* 66, 2 (2019), 1248–1260.

[42] Huan Li, Richard Peng, Liren Shan, Yuhao Yi, and Zhongzhi Zhang. 2019. Current flow group closeness centrality for complex networks. In *Proceedings of the 2019 World Wide Web Conference*. ACM, 961–971.

[43] Huan Li and Zhongzhi Zhang. 2018. Kirchhoff index as a measure of edge centrality in weighted networks: Nearly linear time algorithms. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2377–2396.

[44] Wentao Li, Miao Qiao, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. 2022. On scalable computation of graph eccentricities. In *Proceedings of the 2022 International Conference on Management of Data*. 904–916.

[45] Meihao Liao, Rong-Hua Li, Qiangqiang Dai, Hongyang Chen, Hongchao Qin, and Guoren Wang. 2023. Efficient resistance distance computation: the power of landmark-based approaches. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.

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

[47] Changan Liu, Xiaotian Zhou, Ahad N Zehmakan, and Zhongzhi Zhang. 2023. A Fast Algorithm for Moderating Critical Nodes via Edge Removal. *IEEE Transactions on Knowledge and Data Engineering* (2023).

[48] Zhiqiang Liu and Wenjian Yu. 2023. Computing effective resistances on large graphs based on approximate inverse of Cholesky factor. In *2023 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 1–6.

[49] William S Lovejoy and Christoph H Loch. 2003. Minimal and maximal characteristic path lengths in connected sociomatrices. *Social Networks* 25, 4 (2003), 333–347.

[50] Aleksander Madry, Damian Straszak, and Jakub Tarnawski. 2015. Fast Generation of Random Spanning Trees and the Effective Resistance Metric. In *Proc. 26th Annu. ACM-SIAM Symp. Discrete Algorithms*. San Diego, CA, USA, 2019–2036.

[51] de Berg Mark, Cheong Otfried, van Kreveld Marc, and Overmars Mark. 2008. *Computational geometry algorithms and applications*. Spinger.

[52] Charalampos Mavroforakis, Richard Garcia-Lebron, Ioannis Koutis, and Evimaria Terzi. 2015. Spanning edge centrality: Large-scale computation and applications. In *Proceedings of the 24th International Conference on World Wide Web*. 732–742.

[53] Carl D Meyer, Jr. 1973. Generalized inversion of modified matrices. *SIAM J. Appl. Math.* 24, 3 (1973), 315–323.

[54] Rushabh Patel, Pushkarini Agharkar, and Francesco Bullo. 2015. Robotic surveillance and Markov chains with minimal weighted Kemeny constant. *IEEE Transactions on Automatic Control* 60, 12 (2015), 3156–3167.

[55] Stacy Patterson and Bassam Bamieh. 2014. Consensus and coherence in fractal networks. *IEEE Transactions on Control of Network Systems* 1, 4 (2014), 338–348.

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

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

[58] Clara Pizzuti and Annalisa Socievole. 2022. Incremental computation of effective graph resistance for improving robustness of complex networks: A comparative study. In *International Conference on Complex Networks and Their Applications*. Springer, 419–431.

[59] Maria Predari, Robert Kooij, and Henning Meyerhenke. 2022. Faster greedy optimization of resistance-based graph robustness. In *2022 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE, 1–8.

[60] Yi Qi, Zhongzhi Zhang, Yuhao Yi, and Huan Li. 2019. Consensus in Self-Similar Hierarchical Graphs and Sierpiński Graphs: Convergence Speed, Delay Robustness, and Coherence. *IEEE Trans. Cybern.* 49, 2 (2019), 592–603.

[61] R Tyrrell Rockafellar. 1970. *Convex Analysis*. Vol. 18. Princeton university press.

[62] Ryan Rossi and Nesreen Ahmed. 2015. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI, 4292–4293.

[63] Liren Shan, Yuhao Yi, and Zhongzhi Zhang. 2018. Improving information centrality of a node in complex networks by adding edges. *27th International Joint Conference on Artificial Intelligence*, 3535–3541.

[64] Xinli Shi, Jinde Cao, and Wei Huang. 2018. Distributed parametric consensus optimization with an application to model predictive consensus problem. *IEEE Trans. Cybern.* 48, 7 (July 2018), 2024–2035.

[65] Milad Siami, Sadegh Bolouki, Bassam Bamieh, and Nader Motee. 2017. Centrality measures in linear consensus networks with structured network uncertainties. *IEEE Transactions on Control of Network Systems* 5, 3 (2017), 924–934.

[66] Daniel A. Spielman and Nikhil Srivastava. 2008. Graph Sparsification by Effective Resistances. In *Proc. 40th Annu. ACM Symp. Theory Comput.* Victoria, British Columbia, Canada, 563–568.

[67] Daniel A Spielman and Shang-Hua Teng. 2014. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM J. Matrix Anal. Appl.* 35, 3 (2014), 835–885.

[68] Tyler Summers, Iman Shames, John Lygeros, and Florian Dörfler. 2015. Topology design for optimal network coherence. In *2015 European Control Conference (ECC)*. IEEE, 575–580.

[69] Krishnaiyan Thulasiraman, Mamta Yadav, and Kshirasagar Naik. 2019. Network Science Meets Circuit Theory: Resistance Distance, Kirchhoff Index, and Foster’s Theorems With Generalizations and Unification. *IEEE Trans. Circuits Syst. I: Regular Papers* 66, 3 (2019), 1090–1103.

[70] Ali Tizghadam and Alberto Leon-Garcia. 2010. Autonomic traffic engineering for network robustness. *IEEE J. Sel. Areas Commun.* 28, 1 (2010), 39–50.

[71] Piet Van Mieghem, Karel Devriendt, and H Cetinay. 2017. Pseudoinverse of the Laplacian and best spreader node in a network. *Physical Review E* 96, 3 (2017), 032311.

[72] Xiangrong Wang, Evangelos Pournaras, Robert E Kooij, and Piet Van Mieghem. 2014. Improving robustness of complex networks via the effective graph resistance. *The European Physical Journal B* 87 (2014), 1–12.

[73] 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.

[74] Felix Ming Fai Wong, Zhenming Liu, and Mung Chiang. 2016. On the efficiency of social recommender networks. *IEEE/ACM Transactions on Networking* 24, 4 (2016), 2512–2524.

[75] Wanyue Xu, Yibin Sheng, Zuobai Zhang, Haibin Kan, and Zhongzhi Zhang. 2020. Power-law graphs have minimal scaling of Kemeny constant for random walks. In *Proc. World Wide Web Conf.* Taipei Taiwan, China, 46–56.

[76] Changpeng Yang, Jianfeng Mao, Xiongwen Qian, and Peng Wei. 2018. Designing robust air transportation networks via minimizing total effective resistance. *IEEE Transactions on Intelligent Transportation Systems* 20, 6 (2018), 2353–2366.

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

[78] Yuhao Yi, Liren Shan, Huan Li, and Zhongzhi Zhang. 2018. Biharmonic Distance Related Centrality for Edges in Weighted Networks.. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. 3620–3626.

[79] Yuhao Yi, Bingjia Yang, Zuobai Zhang, Zhongzhi Zhang, and Stacy Patterson. 2022. Biharmonic Distance-Based Performance Metric for Second-Order Noisy Consensus Networks. *IEEE Trans. Inf. Theory* 68, 2 (Feb. 2022), 1220–1236.

[80] Yuhao Yi, Zhongzhi Zhang, and Stacy Patterson. 2020. Scale-Free Loopy Structure Is Resistant to Noise in Consensus Dynamics in Complex Networks. *IEEE Trans. Cybern.* 50, 1 (Jan. 2020), 190–200.

[81] Zuobai Zhang, Wanyue Xu, Yuhao Yi, and Zhongzhi Zhang. 2022. Fast Approximation of Coherence for Second-Order Noisy Consensus Networks. *IEEE Trans. Cybern.* 52, 1 (Jan. 2022), 677–686.

[82] Zuobai Zhang, Zhongzhi Zhang, and Guanrong Chen. 2021. Minimizing spectral radius of non-backtracking matrix by edge removal. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2657–2667.

[83] Xiaotian Zhou and Zhongzhi Zhang. 2021. Maximizing influence of leaders in social networks. In *Proceedings of the 27th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2400–2408.

[84] Liwang Zhu, Qi Bao, and Zhongzhi Zhang. 2021. Minimizing Polarization and Disagreement in Social Networks via Link Recommendation. In *Proceedings of the 35th Conference on Neural Information Processing Systems*. 2072–2084.