2022 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

2022 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM) | 978-1-6654-5661-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/ASONAM55673.2022.10068613

# Faster Greedy Optimization of Resistance-based Graph Robustness

Maria Predari
*Department of Computer Science*
*Humboldt-Universität zu Berlin*
Berlin, Germany
predarim@hu-berlin.de

Robert Kooij
*Faculty of Electrical Engineering, Mathematics and Computer Science*
*Delft University of Technology*
Delft, the Netherlands
r.e.kooij@tudelft.nl

Henning Meyerhenke
*Department of Computer Science*
*Humboldt-Universität zu Berlin*
Berlin, Germany
meyerhenke@hu-berlin.de

***Abstract*—The total effective resistance, also called the Kirchhoff index, provides a robustness measure for a graph $G$. We consider the optimization problem of adding $k$ new edges to $G$ such that the resulting graph has minimal total effective resistance (i. e., is most robust). The total effective resistance and effective resistances between nodes can be computed using the pseudoinverse of the graph Laplacian. The pseudoinverse may be computed explicitly via pseudoinversion; yet, this takes cubic time in practice and quadratic space. We instead exploit combinatorial and algebraic connections to speed up gain computations in established generic greedy heuristics. Moreover, we leverage existing randomized techniques to boost the performance of our approaches by introducing a sub-sampling step. Our different graph- and matrix-based approaches are indeed significantly faster than the state-of-the-art greedy algorithm, while their quality remains reasonably high and is often quite close. Our experiments show that we can now process large graphs for which the application of the state-of-the-art greedy approach was infeasible before. As far as we know, we are the first to be able to process graphs with $100K+$ nodes in the order of minutes.**

***Index Terms*—graph robustness, optimization problem, effective resistance, Kirchhoff index, Laplacian pseudoinverse**

## I. Introduction

The analysis of network topologies has received considerable attention in various fields of science and engineering in the last decades [4]. Its purpose usually is to better understand the functionality, dynamics, and evolution of a network[1] and its components [4]. One important property of a network topology concerns its *robustness*, i. e., the extent to which a network is capable to withstand failures of one or more of its components. As an example, one may ask whether the network is guaranteed to remain connected if an arbitrary edge is deleted. Network robustness is a critical design issue in many areas, including telecommunication [31], power grids [17], public transport [6], supply chains [28] and water distribution [43].

Often a critical step in infrastructural maintenance is to improve the robustness of the network by adding a small number of edges. The challenge here lies in the selection of a vertex pair, among all the possible ones, such that the insertion of an edge between the vertices increases the network's robustness as much as possible. Given a graph $G = (V, E)$ and a budget of $k$ links to be added, our algorithmic formalization of this task asks to find a set $S \subset \binom{V}{2} \setminus E$ of size $k$ that optimizes the robustness of $G$. We call this problem $k$-GRIP, short for *graph robustness improvement problem*. Clearly, one must also choose a measure to capture a sensible notion of robustness; there are numerous ones proposed in the literature [4], [31].

One established measure, which was shown to be a good robustness indicator in various scenarios [9], [11], [40], is *effective graph resistance* or *total effective resistance* of a graph. Effective resistance is a pairwise metric on the vertex set of $G$, which results from viewing the graph as an electrical network. It relates to uniform spanning trees [2], random walks [20], and several centrality measures [5], [22]. For effective graph resistance, one sums the effective resistance over all vertex pairs in $G$ (for technical details see Section II). Intuitively, the effective resistance becomes small if there are many short paths between two vertices. Removing an edge then hardly disrupts the connectivity, since there are usually alternative paths. Due to this favorable property, we select effective graph resistance in this paper as robustness measure for $k$-GRIP.

This resistance-based $k$-GRIP version was already considered by Summers *et al.* [36]. They suggested a greedy algorithm for its heuristic optimization (more details in Section III). Even without an approximation guarantee, this greedy algorithm provides very good empirical results in reasonable time (at least for small networks). The algorithm performs $k$ iterations, adding each time the edge with highest marginal gain. To compute these gains, however, the corresponding effective resistance values are needed. If one acquires them by an initial (pseudo)inversion of the graph's Laplacian matrix, this takes $\mathcal{O}(n^3)$ time with standard tools in practice (where $n = |V|$). Overall, their approach leads to a running time of $\mathcal{O}(kn^3)$, which limits the applicability to large networks.

For other problems where this greedy approach works well, a recent stochastic greedy algorithm [25] has been shown to be potentially much faster – while usually producing solutions of

We gratefully acknowledge support by German Research Foundation (DFG) project ALMACOM (grant ME 3619/4-1) and by the TU Delft Safety & Security Institute project ARCIN. The second author is also affiliated with TNO, Unit ICT, The Hague, the Netherlands.

[1]We use the terms *network* and *graph* interchangeably in this paper.

IEEE/ACM ASONAM 2022, November 10-13, 2022
978-1-6654-5661-6/22/$31.00 © 2022 IEEE

nearly the same quality. It does so by *sampling* from the list of candidates to find the one with highest gain in each iteration. Our hypothesis for this paper is that this favorable time-quality tradeoff of stochastic greedy holds for our $k$-GRIP as well.

Building upon this generic stochastic greedy approach [25], we devise several heuristic strategies that leverage both graph- and matrix-related properties (Section IV). Our approaches shall accelerate the greedy algorithm by reducing the candidate set via careful selection of elements to be evaluated and/or by accelerating the gain computation. Our experiments (Section V) confirm that our approaches speed up the state-of-the-art greedy algorithm significantly. At the same time, the $k$-GRIP solution quality is more or less preserved, how well depends on the approach. For instance, for graphs with $< 57K$ nodes, we produce results that are on average $2-15\%$ away from the greedy solution, while running $3.3-68\times$ faster than the state of the art (SoA). Finally, we demonstrate that we can now process much larger graphs for which the application of the SoA greedy approach was infeasible before.

## II. Preliminaries

We assume that the input of $k$-GRIP is a connected, undirected, and simple graph $G=(V,E)$ with $n$ vertices and $m$ edges and an integer $k \in \mathbb{Z}_{>0}$ for the number of edges to be added to $G$. [2] $\mathbf{L}=\mathbf{D}-\mathbf{A}$ is the $n \times n$ Laplacian matrix of $G$, where $\mathbf{D}$ is the diagonal matrix of vertex degrees and $\mathbf{A}$ the adjacency matrix. $\mathbf{L}$ is symmetric, positive semi-definite and has zero row/column sum $s.t.$, $\mathbf{L1}=\mathbf{0}$ where $\mathbf{1}$ is the all-ones vector. The $m \times n$ incidence matrix $\mathbf{B}$ takes for $e \in E$ and $a \in V$ the values: $\mathbf{B}[e,a]=1$ if $a$ is the destination of $e$, $\mathbf{B}[e,a]=-1$ if $a$ is the origin of $e$ and $\mathbf{B}[e,a]=0$ otherwise. For undirected graphs, the direction of each edge is specified arbitrarily. Moreover, $\mathbf{L}=\mathbf{B^T B}$. It is well-known that $\mathbf{L}$ is not invertible, so its Moore-Penrose pseudoinverse ($\mathbf{L}^\dagger$) is used instead, for which holds: $\mathbf{L}\mathbf{L}^\dagger=\mathbf{L}^\dagger\mathbf{L}=\mathbf{I}-\frac{1}{n}\cdot\mathbf{1}\mathbf{1}^T$ [13]. Since $\mathbf{L}$ is symmetric, it has an orthonormal basis of eigenvectors $\mathbf{U}=[\mathbf{u_1},\ldots,\mathbf{u_n}]$. We write the spectral decomposition as: $\mathbf{L}=\sum_{i=2}^{n}\mathbf{u_i}\lambda_i\mathbf{u_i}^T$, where the eigenvectors $\mathbf{u_2},\ldots,\mathbf{u_n}$ correspond to the ordered eigenvalues $0<\lambda_2\leq,\ldots,\leq\lambda_n$ (excluding the zero eigenvalue). For a graph $G$ we use $\mathbf{L}_G$ [$\mathbf{L}_G^\dagger$] to refer to its Laplacian [Laplacian pseudoinverse]. If there is no subscript in our matrix notation, the associated graph is inferred by the context.

Let $\Omega := \binom{V}{2} \setminus E$. For any $X \subset \Omega$, we define $G' := G \cup X = (V, E \cup X)$ as the graph obtained by adding the edges of $X$ into $G$. Then, $k$-GRIP aims at finding $X \subset \Omega$ with $|X|=k$ s.t., $|f(G)-f(G')|$ is as large as possible for a given robustness function $f(\cdot)$. Here, we use the effective graph resistance $\mathcal{R}(G)$ as robustness function, which is the sum of pairwise effective resistances $\mathbf{r_G}(\cdot,\cdot)$ between all vertex pairs:

$$\mathcal{R}(G)=\sum_{a=1}^{n}\sum_{b=a+1}^{n}\mathbf{r_G}(a,b) \tag{1}$$

[2]Our methods can easily be extended to weighted graphs. However, for simplicity of notations we only consider non weighed graphs.

Computing $\mathbf{r_G}(a,b)$ can be done via $\mathbf{L}^\dagger$:

$$\mathbf{r_G}(a,b)=\mathbf{L}^\dagger[a,a]+\mathbf{L}^\dagger[b,b]-2\mathbf{L}^\dagger[a,b] \tag{2}$$

Combining Eqs. (1) and (2), one gets:

$$\mathcal{R}(G)=n\,\mathrm{tr}(\mathbf{L}^\dagger) \tag{3}$$

For a potential new edge $\{a,b\}$, we have $G'=G\cup\{a,b\}$ and $\mathbf{L}_{G'}=\mathbf{L}_G+(\mathbf{e}_a-\mathbf{e}_b)(\mathbf{e}_a-\mathbf{e}_b)^T$, where $\mathbf{e}_a$ is a zero vector except for $\mathbf{e}[a]=1$. The gain in terms of $\{a,b\}$ is $\mathcal{R}(G)-\mathcal{R}(G')$ and relies on $\mathbf{L}_{G'}^\dagger$ (Sherman-Morrison formula [33]):

$$\mathbf{L}_{G'}^\dagger=\mathbf{L}_G^\dagger-\frac{1}{1+\mathbf{r_G}(a,b)}\mathbf{L}_G^\dagger(\mathbf{e}_a-\mathbf{e}_b)(\mathbf{e}_a-\mathbf{e}_b)^T\mathbf{L}_G^\dagger. \tag{4}$$

The gain evaluation $\mathrm{gain}(a,b)=\mathcal{R}(G)-\mathcal{R}(G')$ is then:

$$\mathrm{gain}(a,b)=n\frac{\left\|\mathbf{L}_\mathbf{G}^\dagger[:,a]-\mathbf{L}_\mathbf{G}^\dagger[:,b]\right\|^2}{1+\mathbf{r_G}(a,b)} \tag{5}$$

where $\mathbf{L}_\mathbf{G}^\dagger[:,a]$ is the $a$th column of $\mathbf{L}^\dagger$. We rewrite Eq. (5) as a function of squared $\ell_2$ norms:

$$\mathrm{gain}(a,b)=n\frac{\left\|\mathbf{L}^\dagger(\mathbf{e}_a-\mathbf{e}_b)\right\|^2}{1+\left\|\mathbf{B}\mathbf{L}^\dagger(\mathbf{e}_a-\mathbf{e}_b)\right\|^2}=n\frac{\mathbf{b_G}(a,b)}{1+\mathbf{r_G}(a,b)}, \tag{6}$$

where $\mathbf{b_G}(\cdot,\cdot)$ is known as the bi-harmonic distance of $G$ [41]. Finally, we express these distances via spectral decomposition:

$$\begin{aligned}\mathbf{r_G}(a,b)&=\left\|\mathbf{B}\mathbf{L}^\dagger(\mathbf{e}_a-\mathbf{e}_b)\right\|^2=(\mathbf{e}_a-\mathbf{e}_b)^T\mathbf{L}^\dagger(\mathbf{e}_a-\mathbf{e}_b)\\&=(\mathbf{e}_a-\mathbf{e}_b)^T\mathbf{U}\mathbf{\Lambda}^{-1}\mathbf{U}^T(\mathbf{e}_a-\mathbf{e}_b)=\sum_{i=2}^{n}\frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{\lambda_i}\end{aligned} \tag{7}$$

where $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues. Similarly:

$$\begin{aligned}\mathbf{b_G}(a,b)&=\left\|\mathbf{L}^\dagger(\mathbf{e}_a-\mathbf{e}_b)\right\|^2=(\mathbf{e}_a-\mathbf{e}_b)^T(\mathbf{L}^\dagger)^2(\mathbf{e}_a-\mathbf{e}_b)\\&=(\mathbf{e}_a-\mathbf{e}_b)^T\mathbf{U}\mathbf{\Lambda}^{-2}\mathbf{U}^T(\mathbf{e}_a-\mathbf{e}_b)=\sum_{i=2}^{n}\frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{\lambda_i^2}\end{aligned} \tag{8}$$

## III. Related Work

Robustness of networks has been an active research area for decades [29]. Several authors have proposed the use of specific network metrics to quantify the robustness of a given network, see e. g., [31], [10], [32], [7]. Rueda *et al.* [31] proposed a taxonomy of robustness metrics, consisting of three classes: *structural metrics*, *centrality metrics* and *functional metrics*. Examples of structural metrics include vertex and edge connectivity. Typical centrality metrics used in the context of robustness are betweenness and closeness centrality. Finally, functional metrics take into account a specific service that is delivered over the networks – such as throughput and elasticity. Of specific interest is the *Fiedler value*, i. e., the second smallest eigenvalue $\lambda_2$ of the graph's Laplacian [10]; it captures the overall connectivity of a graph, which is why it is also called *algebraic connectivity*. This metric is also related to synchronization of networks, including opinion dynamics [26].

Once the robustness of a network has been established, a natural next step is to determine how robustness can be improved. One approach is to rewire the edges [32]. A second approach is to add elements to the network. As an example, [39] derives heuristics for the addition of a single edge to increase the algebraic connectivity. Manghiuc *et al.* [21] consider a weighted decision variant of $k$-GRIP w. r. t. $\lambda_2$. They propose an almost-linear time algorithm that augments the graph by $k$ edges such that $\lambda_2$ exceeds a specified threshold.

Effective graph resistance as a robustness measure dates back at least to Ellens *et al.* [9]. It has been known much longer, however, that effective resistance is proportional to commute times of random walks [11]. Refs. [40] and [29] investigate heuristics for 1-GRIP with effective graph resistance (both for edge addition *and* removal). Besides deriving theoretical bounds, Wang *et al.* [40] compare spectral strategies for edge selection with much simpler heuristics. Their experiments confirm that their spectral strategies (particularly the one based on the highest effective resistance gain) often yield the largest improvement, indicating a tradeoff between running time and the robustness gain. To further improve the solution quality in [40], Pizzuti and Socievole [29] suggest a time-consuming genetic algorithm that often seems to find the optimum when given enough time (their instances have less than $5{,}000$ vertices and $6{,}600$ edges).

The state-of-the-art heuristic for $k$-GRIP is a greedy algorithm presented by Summers *et al.* [36], called here StGreedy. In its generic form, such a greedy algorithm adds in each of the $k$ iterations the element (here: edge) with the largest marginal gain (here: best improvement of effective graph resistance). To this end, StGreedy computes the full pseudoinverse of $\mathbf{L}$ as a preprocessing. Then, the marginal gains of all vertex pairs are computed via Eq. (5) in $\mathcal{O}(n)$ time per edge. The edge with best marginal gain is added to the graph, and the pseudoinverse is updated using Eq. (4). The worst-case time complexity is $\mathcal{O}(kn^3)$, which is due to the evaluation of the gain function in $k$ rounds on $\mathcal{O}(n^2)$ node pairs. The preprocessing takes $\mathcal{O}(n^3)$ time with standard tools. For monotonic submodular problems, the generic greedy algorithm has an approximation ratio of $1-1/e$. Even for non-submodular problems such as $k$-GRIP (see [35] for a counterexample), the greedy algorithm still often leads to solutions of high quality [1], [37].

Algorithms that improve the time complexity of the greedy approach for a general setting were proposed in [14], [25]. These algorithms use randomized techniques and reduce the total number of function evaluations by a factor of $k$. They achieve provable approximation guarantees in cases where the greedy algorithm admits them, too.

## IV. Heuristics for $k$-GRIP

In this section, we propose different techniques to improve the performance of the greedy algorithm for $k$-GRIP. Our approaches are: SimplStoch, ColStoch, SimplStochJLT, ColStochJLT and SpecStoch. They all make use of an existing randomized technique and follow the general greedy framework of Algorithm 1. Functions named as Obj* relate to the objective function while those named as Candidate* relate to the set of possible candidate elements.[3] The worst-case time complexities of all approaches are shown in Table I.

TABLE I
Worst-case time complexities of all approaches involved. Columns correspond to principal steps of Algorithm 1. In general, the dominant factor comes from the total number of evaluations and their time to be evaluated (second column).

| | Compute | #Evals × SingleEval | Update |
|---|---|---|---|
| StGreedy | $\mathcal{O}(n^3)$ | $\mathcal{O}(kn^2) \times \mathcal{O}(n)$ | $\mathcal{O}(kn^2)$ |
| SimplStoch | $\mathcal{O}(n^3)$ | $\mathcal{O}(n^2) \times \mathcal{O}(n)$ | $\mathcal{O}(kn^2)$ |
| ColStoch | $\tilde{\mathcal{O}}(sm \log n)$ | $\mathcal{O}(n^2) \times \mathcal{O}(n)$ | $\tilde{\mathcal{O}}(ksm \log n)$ |
| SimplStochJLT | $\tilde{\mathcal{O}}(m \log n \eta^{-2})$ | $\mathcal{O}(n^2) \times \mathcal{O}(\log n)$ | $\tilde{\mathcal{O}}(km \log n \eta^{-2})$ |
| ColStochJLT | $\tilde{\mathcal{O}}(m \log n \log s \eta^{-2})$ | $\mathcal{O}(n^2) \times \mathcal{O}(\log s)$ | $\tilde{\mathcal{O}}(km \log n \log s \eta^{-2})$ |
| SpecStoch | $\mathcal{O}(cm)$ | $\mathcal{O}(n^2) \times \mathcal{O}(c)$ | $\mathcal{O}(kcm)$ |

For submodular functions the greedy framework can be combined with a lazy technique [23] that boosts the performance of the algorithm. This process is based on the fact that, even though marginal gains of elements might change between iterations, their order often stays the same. Important for us: "(T)he lazy greedy algorithm can be applied to cases with no strict guarantee (for submodularity) since experience shows that it most often produces the same final solution as the standard greedy algorithm" [24]. Based on the above observation and existing, positive results on lazy greedy for $k$-GRIP [36], we also employ this technique.

All our approaches improve the speed of the greedy algorithm by reducing the candidate set and/or by accelerating the objective function calculation. Nearly inevitably, the above incurs a smaller or larger trade-off between speed improvement and solution quality degradation.

### A. SimplStoch

Our first idea is to simply apply the generic randomized technique proposed in generic form by Mirzasoleiman *et al.* [25] in the context of $k$-GRIP. The main idea of [25] is to not inspect all possible elements for insertion, but only a

[3]Functions not defined explicitly in the pseudocode are described in detail in the text.

**Algorithm 1** General framework for $k$-GRIP

```
1: function GreedyFramework(G, k, δ)
2:     Input: Graph G = (V, E), k ∈ ℤ_{>0}, accuracy 0 < δ < 1
3:     Output: G_k – graph after k edge insertions
4:     G_0 ← G
5:     ComputeObj(G_0, ...)                          ▷ compute step
6:     s ← CandidateSize(m, n, k, δ)
7:     for r ← 0, ..., k − 1 do                      ▷ main loop
8:         S ← Candidates(s, G_r, ...)
9:         for each {a, b} ∈ S × S do                ▷ # of evaluations
10:            gain(a, b) ← Eval(a, b, ...)          ▷ single evaluation
11:        (a*, b*) ← argmax_{a∈S×b∈S} gain(a, b)
12:        G_{r+1} = G_r ∪ (a*, b*)
13:        Update(G_{r+1}, ...)                      ▷ update step
14:    return G_{r+1}
```

reduced sample $\mathcal{S}$. For non-negative monotone submodular functions (which does not hold for $k$-GRIP), the stochastic approach provides an approximation ratio of $(1 - 1/e - \delta)$, where $0 \leq \delta \leq 1$ is an accuracy parameter.

Regarding SIMPLSTOCH, any sample $\mathcal{S}$ is a subset of $\binom{V}{2} \setminus E$. During each iteration of the main loop we sample uniformly at random $s := \frac{n^2-m}{k} \log \frac{1}{\delta}$ vertex pairs (Line 8 in Algorithm (1)), resulting in $(n^2-m) \log \frac{1}{\delta}$ function evaluations overall. Those are performed via the Laplacian pseudoinverse, in a similar way as in STGREEDY. More precisely, $\mathbf{L}^\dagger$ is computed once before the main loop (Line 5) and is used within the loop to quickly determine single evaluations (Line 10). Every time an edge is added to the graph, $\mathbf{L}^\dagger$ is updated accordingly via Eq. (4) (Line 13). The cost of the main loop for SIMPLSTOCH is reduced compared to greedy by a factor of $k$. Yet, computing $\mathbf{L}^\dagger$ is still very time- and space-consuming.

### B. COLSTOCH

Our first improvement upon SIMPLSTOCH avoids the full pseudoinversion of $\mathbf{L}$, reducing the cost of Line 5 in Alg. 1. To this end, we make the following observation: evaluating a single vertex pair $\{a, b\}$ via Eq. (5) requires only two columns of $\mathbf{L}^\dagger$; precisely those corresponding to vertices $a$ and $b$. That is why, instead of sampling elements from $\binom{V}{2} \setminus E$, COLSTOCH restricts the sampling process to elements from $V$, the set of $\mathbf{L}^\dagger$ columns. Carefully selecting $\mathcal{S}$ is critical as it affects the quality of the solution. Even if our problem is not submodular, we choose the default sample size of $s = n\sqrt{\frac{1}{k} \cdot \log(\frac{1}{\delta})}$ elements (Line 6), leading to $n^2 \log \frac{1}{\delta}$ evaluations over all iterations, similar to SIMPLSTOCH.[4] Moreover, to limit the quality loss, we choose elements of $\mathcal{S}$ following graph-based sampling probabilities (see below). These probabilities are initially calculated during the compute step (Line 5) and are updated accordingly in the update step (Line 13). Function CANDIDATES() also receives those sampling probabilities in each iteration (Line 8). Once $\mathcal{S}$ is determined, we compute all columns of $\mathbf{L}^\dagger$ corresponding to vertices in $\mathcal{S}$.[5] We do so by solving $s$ linear systems. More precisely, we solve one linear system for each vertex $a \in \mathcal{S} : \mathbf{Lx} = \mathbf{e}_a - \frac{1}{n} \cdot \mathbf{1}$, where $\mathbf{1} = (1, \ldots, 1)^T$ and $\mathbf{x} \perp \mathbf{1}$. Then, COLSTOCH performs function evaluations only between vertex pairs in $\mathcal{S} \times \mathcal{S}$ (Line 10). Finally, to further improve the overall running time, we do not update $\mathbf{L}^\dagger_{\mathbf{G}}[:, \mathcal{S}]$ for all $a \in \mathcal{S}$ at the end of each round (Line 13 of Algorithm 1). Instead, we update individual columns of $\mathbf{L}^\dagger$ on demand; only if the corresponding vertices participate in the candidate set $\mathcal{S}$ of the following round.

To update previously computed columns, we use the outdated solver solution and apply the update formula Eq. (4) iteratively for all (in-between) rounds. To do so, we store columns together with the associated round count.

[4]The only difference here is that we sample pairs of $\mathbf{L}^\dagger$ columns, which is a subset of $\binom{V}{2}$ and not $\binom{V}{2} \setminus E$. Obviously, we reject vertex pairs that already exist in the graph as edges.

[5]This step is performed once in the main loop after Line 8. For the complexity analysis we consider it as part of the compute step and for that reason it is not depicted in the loop of the generic Algorithm 1

$\mathrm{diag}(\mathbf{L}^\dagger)$ *Strategy:* Let us now explain the sampling probabilities for selecting $\mathcal{S}$. Following previous studies [38], [40], vertex pairs with maximal effective resistance are good candidates for largely decreasing the total effective resistance of a graph. However, the effective resistance metric is not directly applicable in this context. For once because COLSTOCH requires a vertex-based metric and secondly, and more importantly, because computing the effective resistance for all vertex pairs $\{a, b\} \in \binom{V}{2} \setminus E$ would eventually require (pseudo)inverting $\mathbf{L}$. To work out these issues, we sample vertices according to their corresponding diagonal entries in $\mathbf{L}^\dagger$. Obviously, this is an effective resistance-based metric since the diagonal entry $\mathbf{L}^\dagger[a, a]$ of a vertex $a$ corresponds to the summed effective resistance between $a$ and all other vertices: $\sum_{b \in V \setminus \{a\}} \mathbf{r_G}(a, b)$. Vertices with maximum $\mathbf{L}^\dagger$ diagonal values are connected badly to all other vertices in the graph (in the electrical sense) [38]. Moreover, computing $\mathrm{diag}(\mathbf{L}^\dagger)$ can be performed in almost-linear time, using the connection of effective resistance to uniform spanning trees (USTs) of $G$. In [2] the authors proposed an algorithm that approximates $\mathrm{diag}(\mathbf{L}^\dagger)$ via UST sampling techniques. The algorithm obtains a $\pm\epsilon$-approximation with high probability in $\mathcal{O}(m \log^4 n \cdot \epsilon^{-2})$ time for small-world graphs (diameter bounded by $\mathcal{O}(\log n)$).

For $k$-GRIP, we need to sample USTs for every new graph $G_{r+1}$ (in round $r$). We do so during the update step of Algorithm 1 (Line 13). We save computations by reusing previously computed USTs corresponding to $G_r$. Those trees are not uniformly distributed in the new graph $G_{r+1} := G_r \cup \{a, b\}$ and need to be reweighted accordingly. Moreover, we still need to sample a number of USTs corresponding to trees of $G_{r+1}$ that contain the additional edge $\{a, b\}$. To do so, we use a variant of Wilson's algorithm [42]. The final sample set is the union of the reweighted USTs (originally from $G_r$) and the newly sampled USTs in $G_{r+1}$.

### C. *STOCHJLT

In this section we propose an improvement that exploits the following observation: to evaluate the gain function for an arbitrary vertex pair $\{a, b\}$, we only require to compute the squared $\ell_2$-norm of two distance vectors: $\mathbf{b_G}(a, b) = \left\|\mathbf{L}^\dagger(\mathbf{e}_a - \mathbf{e}_b)\right\|^2$ and $\mathbf{r_G}(a, b) = \left\|\mathbf{B}^T\mathbf{L}^\dagger(\mathbf{e}_a - \mathbf{e}_b)\right\|^2$ (Eq. (6)). Viewing $\mathbf{b_G}(a, b)$ and $\mathbf{r_G}(a, b)$ as pair-wise distances between vectors in $\{\mathbf{L}^\dagger\}_{a \in V}$ and $\{\mathbf{B}^T\mathbf{L}^\dagger\}_{a \in V}$ (respectively) allows us to apply the Johnson-Lindenstrauss lemma [16]. In this case, pairwise distances among vectors are nearly preserved if we project the vectors onto a low-dimensional subspace, spanned by $\mathcal{O}(\log n)$ random vectors. The JLT lemma states the following:

*Lemma IV-C.1 (Johnson-Lindenstrauss):* Given fixed vectors $\mathbf{u_1} \ldots, \mathbf{u_n} \in \mathbb{R}^d$ and $\eta > 0$, let $\mathbf{Q} \in \mathbb{R}^{q \times d}$ be a random $\pm 1\sqrt{q}$ matrix (i. e., independent Bernoulli entries) with $q \leq 24 \log n / \eta^2$. Then with probability at least $1 - 1/n$

$$(1 - \eta) \left\|\mathbf{u_i} - \mathbf{u_j}\right\|^2 \leq \left\|\mathbf{Qu_i} - \mathbf{Qu_j}\right\|^2 \leq (1 + \eta) \left\|\mathbf{u_i} - \mathbf{v_j}\right\|^2 \tag{9}$$

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

Using Lemma IV-C.1, we can simply project matrices $\mathbf{L}^\dagger$ and $\mathbf{B}^T\mathbf{L}^\dagger$ onto $q$ vectors, i. e., the $q$ rows of some random matrices $\mathbf{P} \in \mathbb{R}^{q\times n}$ and $\mathbf{Q} \in \mathbb{R}^{q\times m}$, respectively. To actually reduce the overall computation time, we need to avoid the involved pseudoinversion. For that, we resort to efficient linear system solvers. Thus, combining the random projections technique with fast linear solvers, one can approximate distances between vertex pairs within a factor of $(1 \pm \eta)$ in $\mathcal{O}(I(n,m)\log n/\eta^2)$ time, where $I(n,m)$ is the running time of the Laplacian solver. Hence to approximate $\mathbf{b_G}(a,b)$ and $\mathbf{r_G}(a,b)$, we compute the projected distances $\left\|\mathbf{PL}^\dagger(\mathbf{e}_a-\mathbf{e}_b)\right\|^2$ and $\left\|\mathbf{QBL}^\dagger(\mathbf{e}_a-\mathbf{e}_b)\right\|^2$, respectively. We avoid the solution of two sets of Laplacian systems by expressing the effective resistances directly via the projection of bi-harmonic distances onto the lower dimension space. More precisely, we only solve $\mathbf{LY} = \mathbf{P}^T - \frac{1}{n}\mathbf{1}\mathbf{1}^T\mathbf{P}^T$ and use[6] $\mathbf{Y} = \mathbf{L}^\dagger\mathbf{P}^T$ to express effective resistances:

$$\begin{aligned}\left\|\mathbf{QB}^T\mathbf{YP}(\mathbf{e}_a-\mathbf{e}_b)\right\|^2 &= (\mathbf{e}_a-\mathbf{e}_b)^T\mathbf{P}^T\mathbf{Y}^T\mathbf{BQ}^T\mathbf{QB}^T\mathbf{YP}(\mathbf{e}_a-\mathbf{e}_b)\\ &= (\mathbf{e}_a-\mathbf{e}_b)^T\mathbf{L}^\dagger\mathbf{B}\mathbf{B}^T\mathbf{L}^\dagger(\mathbf{e}_a-\mathbf{e}_b) = \left\|\mathbf{B}^T\mathbf{L}^\dagger(\mathbf{e}_a-\mathbf{e}_b)\right\|^2\end{aligned} \tag{10}$$

since $\mathbf{Q}$ and $\mathbf{P}$ are orthogonal matrices.

We can integrate the JLT approximation both in the context of ColStoch and SimplStoch (having ColStochJLT and SimplStochJLT respectively). Let us consider the case of ColStochJLT: Again, the compute step is performed after selecting set $\mathcal{S}$ (just after Line 8). Indeed, we compute the vectors in $\{\mathbf{L}^\dagger\}_{a\in\mathcal{S}}$ and $\{\mathbf{B}^T\mathbf{L}^\dagger\}_{a\in\mathcal{S}}$ for $G_0$, where $|\mathcal{S}| = n\sqrt{\frac{1}{k}\cdot\log(\frac{1}{\delta})}$. Since, later, we only perform evaluations for pairs in $\mathcal{S}\times\mathcal{S}$, it suffices to consider projections onto $\log s$ rows (via $\mathbf{P} \in \mathbb{R}^{\log s\times n}$ and $\mathbf{Q} \in \mathbb{R}^{log s\times m}$). During the main loop of Algorithm 1 we perform the same number of overall function evaluations as in ColStoch, that is $\mathcal{O}(n^2)$. However, now a single function evaluation for an arbitrary vertex pair takes $\mathcal{O}(\log s)$ via the formula $\mathrm{gain}(a,b) \approx \frac{\left\|\mathbf{PL}^\dagger(\mathbf{e}_a-\mathbf{e}_b))^2\right\|}{1+\left\|\mathbf{QB}^T\mathbf{YP}(\mathbf{e}_a-\mathbf{e}_b)^2\right\|}$ (up to a relative error of $\eta$).[7] For the update step, we need to sample new projections $\mathbf{P}$ and $\mathbf{Q}$ and recompute the two matrices $\mathbf{PL}^\dagger$ and $\mathbf{QB}^T\mathbf{YP}$.

### D. SpecStoch

Finally, we propose an approach that exploits the spectral expression of the gain function. More precisely, we combine the spectral expressions of effective resistance and bi-harmonic distance (Eqs. (8) and (7)) to write Eq. (5) as:

$$\mathrm{gain}(a,b) = \frac{\sum_{i=2}^{n}\frac{1}{(\lambda_i)^2}\cdot(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{1+\sum_{i=2}^{n}\frac{1}{\lambda_i}\cdot(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2} \tag{11}$$

Eq. (11) benefits from the fact that both effective resistance and bi-harmonic distance only depend on the spectrum of the same matrix $\mathbf{L}$. Still, the full spectral decomposition of $\mathbf{L}$ incurs $\mathcal{O}(n^3)$ time and is equally prohibitive as computing $\mathbf{L}^\dagger$ for larger $G$. To reduce the complexity, we propose an approximation of Eq. (11) using standard low-rank techniques and new bounds for both distances. To do so, we exploit the fact that the bulk of eigenvalues tends to concentrate away from the smallest eigenvalues [8]. Moreover, we compute only a small number of eigenpairs on the lower side of the spectrum. We expect that the smaller eigenpairs have a larger influence on the sums of Eq. (11). For small $i$, the entries of eigenvector $\mathbf{u_i}$ fluctuate slowly, so we should carefully select $\{a,b\}$ to avoid near-zero contributions. Moreover, for small $i$, contributions are accentuated by a large weight, $\frac{1}{\lambda_i^2}$. On the other hand, for large $i$, the eigenvectors $\mathbf{u_i}$ fluctuate rapidly, since they correspond to high frequency modes of the spectrum. Their importance is undermined by $\frac{1}{\lambda_i^2}$ (small for large $i$). The above observations suggest that for a new edge addition $\{a,b\}$, the focus should be on eigenpairs corresponding to small $i$.

We now show how to derive bounds for $\mathbf{b}(a,b)$. First we break Eq. (8) into partial sums where $c \le n$ is a cut-off value.

$$\begin{aligned}\mathbf{b_G}(a,b) &= \sum_{i=2}^{c}\frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{\lambda_i^2} + \sum_{i=c+1}^{n}\frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{{\lambda_i}^2}\\ &\le \sum_{i=2}^{c}\frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{{\lambda_i}^2} + \frac{1}{{\lambda_c}^2}\sum_{i=c+1}^{n}(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2\\ &\le \sum_{i=2}^{c}\frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{{\lambda_i}^2} + \frac{1}{{\lambda_c}^2}(2-\sum_{i=2}^{c}(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2)\\ &= \frac{2}{{\lambda_c}^2} + \sum_{i=2}^{c}(\frac{1}{{\lambda_i}^2}-\frac{1}{{\lambda_c}^2})(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2\end{aligned} \tag{12}$$

The first inequality holds for large enough eigenvalues ($\ge 1$), since $\lambda_c \le \lambda_{c+i}$ and $\frac{1}{(\lambda_c)^2} \ge \frac{1}{(\lambda_{c+i})^2}$ for any $i$ (recall that we index the eigenvalues ordered non-decreasingly). Moreover, the third line comes from the following observation:

$$\begin{aligned}\sum_{i=2}^{n}(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2 &= \sum_{i=1}^{n}(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2\\ &= \sum_{i=1}^{n}\mathbf{u_i}[a]^2 + \sum_{i=1}^{n}\mathbf{u_i}[b]^2 - 2\sum_{i=1}^{n}\mathbf{u_i}[a]\mathbf{u_i}[b]\\ &= \left\|\mathbf{u_a^T}\right\|^2 + \left\|\mathbf{u_b^T}\right\|^2 - 2\mathbf{U}[a,:]\mathbf{U^T}[:,b] = 2\end{aligned} \tag{13}$$

for $a \ne b$ since $\mathbf{U}$ is double-orthogonal. Moreover:

$$\begin{aligned}\mathbf{b_G}(a,b) &= \sum_{i=2}^{c}\frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{\lambda_i^2} + \sum_{i=c+1}^{n}\frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{{\lambda_i}^2}\\ &\ge \sum_{i=2}^{c}\frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{{\lambda_i}^2} + \frac{1}{{\lambda_n}^2}\sum_{i=c+1}^{n}(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2\\ &\ge \sum_{i=2}^{c}\frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{{\lambda_i}^2} + \frac{1}{{\lambda_n}^2}(2-\sum_{i=2}^{c}(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2)\\ &= \frac{2}{{\lambda_n}^2} + \sum_{i=2}^{c}(\frac{1}{{\lambda_i}^2}-\frac{1}{{\lambda_n}^2})(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2\end{aligned} \tag{14}$$

where the inequality in the third line holds, since $\lambda_n \ge \lambda_{c+i}$ for any $i$. Following the above, we can easily derive similar

[6] Due to $\mathbf{L}^\dagger\cdot\frac{1}{n}\mathbf{1}\mathbf{1}^T = \mathbf{O}$ (the zero matrix).

[7] In our experiments we set $\eta = 0.55$.

bounds for $\mathbf{r_G}(a,b)$. Plugging those bounds together, we can approximate Eq. (11) using the following inequality:

$$\begin{aligned}\frac{\frac{2}{{\lambda_c}^2}+\sum_{i=2}^{c}(\frac{1}{{\lambda_i}^2}-\frac{1}{{\lambda_c}^2})(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{1+\frac{2}{\lambda_n}+\sum_{i=2}^{c}(\frac{1}{\lambda_i}-\frac{1}{\lambda_n})(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2} &\leq \text{gain}(a,b)\\ \leq \frac{\frac{2}{{\lambda_n}^2}+\sum_{i=2}^{c}(\frac{1}{{\lambda_i}^2}-\frac{1}{{\lambda_n}^2})(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{1+\frac{2}{\lambda_c}+\sum_{i=2}^{c}(\frac{1}{\lambda_i}-\frac{1}{\lambda_c})(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}&\end{aligned} \tag{15}$$

Adapting the general framework of Algorithm 1 for SPECSTOCH is rather straightforward: In Line 5 we compute the first $c$ eigenpairs along with the largest eigenvalue of $\mathbf{L}$ (corresponding to $G_0$). We do so using standard iterative methods, such as the Lanczos algorithm [27], which often takes $\mathcal{O}(cm)$ time for sparse matrices, depending on the desired accuracy and eigenvalue distribution. During the main loop, the algorithm performs $\mathcal{O}(n^2)$ function evaluations (dictated by the stochastic approach). Single function evaluations in Line 10 require only $\mathcal{O}(c)$ time using the bounds in Eq. (15). Finally, we update the eigenpairs of $G_{r+1}$ in Line 13. To speed up the update step, we bootstrap the solution of the eigensolver with the solution of the previous round.

## V. EXPERIMENTAL RESULTS

We conduct experiments to demonstrate the performance of our contributions compared to STGREEDY. All algorithms are implemented in C++, using the NetworKit [34] graph APIs. Our test machine is a shared-memory server with a 2x 18-Core Intel Xeon 6154 CPU and a total of 1.5 TB RAM. To ensure reproducibility, experiments are managed by SimexPal [3]. Moreover, we use both synthetic and real-world input instances. The synthetic ones follow the Erdős-Rényi (ER), Barabási-Albert (BA) and Watts-Strogatz (WS) models. The real-world graphs are taken from SNAP [18] and NR [30], including application-relevant power grid and road networks. In this context, we consider medium graphs those whose vertex count is $< 57$K. The largest graph has around 129K nodes. To evaluate the quality of the solutions, we measure gain improvements: $\mathcal{R}(G)-\mathcal{R}(G_k)$. Our code and the experimental pipeline are available at https://github.com/predari/graph-robustness-k.

*Configuration experiments:* First, it is essential to verify that STGREEDY produces good quality, as claimed by the authors of [36], even without any approximation guarantees. To this end, we compare STGREEDY with an exhaustive search on three synthetically generated graphs (BA, WS and ER of size $n = 1000$) and different values of $k = 2, 5, 20, 50, 100$ [8] and different values of $k = 2, 5, 20, 50, 100$. Our experiment confirms that STGREEDY achieves the same solution as the exhaustive search in all cases. Moreover, we evaluate the performance of SIMPLSTOCH for different accuracy values on the medium graphs of Table II. Following the experiments in [25], we set the accuracy parameter $\delta$ to .9 and .99[9] (which are reasonable values according to the experiments of Ref. [25] and our own preliminary experiments). In Table III, we see that there is a clear trade-off between quality and running time, controlled by the accuracy parameter. Still, even for a large $\delta$, the solution of SIMPLSTOCH is not far off compared to STGREEDY, being only 8% off in the worst data point ($k = 2$). We also note that the solution quality is improved as $k$ becomes larger. To benefit from that trade-off, in the following experiments we set $\delta$ at .09 for medium graphs and .099 for larger ones.

[8]The parameters used for generating those graphs are described in Fig. 1.
[9]Providing that $\delta$ can be larger than $1 - \frac{1}{e}$

TABLE II
SUMMARY OF GRAPH INSTANCES, PROVIDING (IN ORDER) NETWORK NAME, VERTEX COUNT, AND EDGE COUNT.

| Graph | $\|V\|$ | $\|E\|$ |
|---|---|---|
| inf-power | 4K | 6K |
| facebook-ego-combined | 4K | 8.8K |
| web-spam | 4K | 37K |
| Wiki-Vote | 7K | 100K |
| p2p-Gnutella09 | 8K | 2.6K |
| p2p-Gnutella04 | 10K | 39K |
| web-indochina | 11K | 47K |
| ca-HepPh | 11K | 117K |
| web-webbase-2001 | 16K | 25K |
| arxiv-astro-ph | 17K | 196K |
| as-caida20071105 | 26K | 53K |
| cit-HepTh | 27K | 352K |
| ia-email-EU | 32K | 54.4K |
| loc-brightkite | 57K | 213K |
| soc-Slashdot0902 | 82K | 504K |
| ia-wiki-Talk | 92K | 360K |
| flickr | 106K | 2.31M |
| livemocha | 104K | 2.19M |
| road-usroads | 129K | 165K |

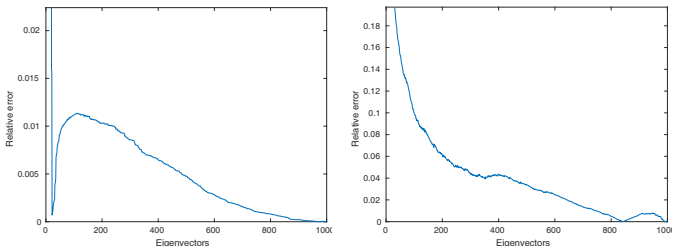
TABLE III
QUALITY AND SPEEDUP OF SIMPLSTOCH (RELATIVE TO STGREEDY) FOR DIFFERENT APPROXIMATION BOUND.

| SIMPLSTOCH | Relative Quality | | | | |
|---|---|---|---|---|---|
| | $k=2$ | $k=5$ | $k=20$ | $k=50$ | $k=100$ |
| $\delta = .9$ | 0.9662 | 0.9610 | 0.9696 | 0.9810 | 0.9898 |
| $\delta = .99$ | 0.9239 | 0.9241 | 0.9442 | 0.9559 | 0.9694 |

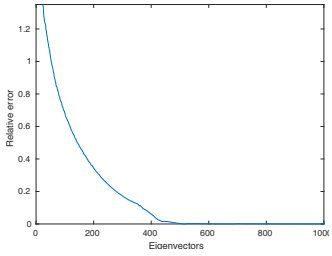
| SIMPLSTOCH | Relative Speedup | | | | |
|---|---|---|---|---|---|
| | $k=2$ | $k=5$ | $k=20$ | $k=50$ | $k=100$ |
| $\delta = .9$ | 2.6 | 2.6 | 2.7 | 2.7 | 3.1 |
| $\delta = .99$ | 4.0 | 3.9 | 4.2 | 4.1 | 4.6 |

Additionally, we perform configuration experiments to determine the quality of the gain approximation via Eq. (15) for SPECTSTOCH. To do so, we randomly select a vertex pair and compute Eq. (15) for a different number of eigenvectors. We measure the relative error of the approximation compared to a full spectrum computation. In Fig. 1 we depict the results for synthetic graphs and eigenvector number from 1 to $n = 1000$. Even for a few tens of eigenvectors, the relative errors for WS and ER are already quite small. The relative error for BA is larger and would require a couple of hundreds eigenvectors to achieve a similar approximation.

Finally, we experiment with different solvers for the solution of Laplacian linear systems. We decide to use the sparse LU solver from the Eigen [12] library for medium graphs and the LAMG solver [19] from NetworKit for larger ones. We do so, because LAMG exhibits a better empirical running time for larger complex networks than other Laplacian solvers. For the solution of the eigensystem (required by SPECSTOCH), we use the Slepc [15] library.




(a) WS: 40 avg. degree, rewir. prob. 0.01 (b) ER: probability $p = 0.01$



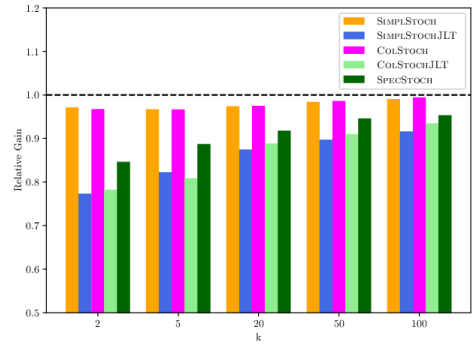

(c) BA: $m_0 = m = 4$

Fig. 1. Relative error of gain via Eq. (15) for different number of eigenvectors.

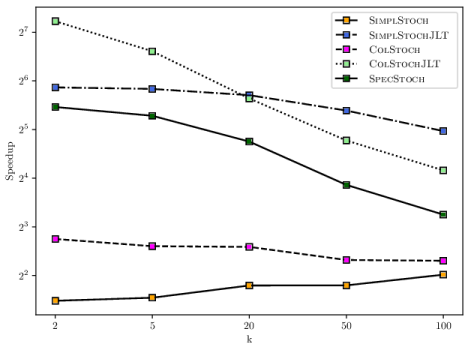
*Comparison to SoA:* We first compare our approaches on the medium graphs of Table II, configured according to the previous section. Closely behind StGreedy, SimplStoch and ColStoch produce the best solutions and they are on average 2% away from the reference (Fig. 2(a)). Moreover, SpecStoch, SimplStochJLT and ColStochJLT are away by 9%, 14% and 15%, respectively. On the other hand, regarding running time, the JLT-based approaches are the fastest, being on average 48× (SimplStochJLT) and 68× (ColStochJLT) faster than StGreedy (Fig. 2(b)). The scaling of ColStochJLT is worse than that of SimplStochJLT for large $k$. This is due to the update step of Algorithm 1, where ColStochJLT needs to update both the effective resistance metric and the necessary operations for JLT. Although the slowest, SimplStoch has a good scaling behavior as it performs only few computations in the update step and thus is independent of $k$. Overall, SpecStoch is the best approach for medium graphs as it produces good quality results and is on average 26× faster than StGreedy. A disadvantage of SpecStoch is that the running time becomes worse as $k$ grows due to the $k$ eigensystem updates.

Finally, in Fig. 3 we depict results for the large graphs of Table II. For this experiment we report absolute values since we do not have a clear reference; StGreedy always times out.[10] The best approaches for large graphs are ColStochJLT and ColStoch. Both of them produce the highest quality results and ColStochJLT is the fastest approach, requiring on average 2 [20] minutes for $k = 2$ [$k = 20$]. SpecStoch is on average as fast as ColStoch but its performance depends a lot on spectral properties (clustered eigenvalues or not) of each input, as shown by the degree of skewness in Fig. 3.

[10]The time limit is set to 43,200 seconds (12 hours).




(a) Quality




(b) Speedup

Fig. 2. Aggregated results (via geometric mean) of $k$-GRIP on medium graphs ($n < 57K$) for different $k$. Results are relative to StGreedy.

## VI. Conclusions

To conclude, our randomized techniques for speeding up the state-of-the-art greedy algorithm for $k$-GRIP do pay off. For medium-sized graphs, ColStoch provides a decent 6× acceleration with a quality close to greedy's. Here, a subset of vertices $i$ is selected for which $\mathbf{L}^{\dagger}[i,i]$ and, thus, their summed effective resistances are large. When favoring speed over quality, SpecStoch, which exploits spectral properties of the graph, seems to offer the better trade-off (on average 28× faster than greedy). For larger graphs and whenever high quality is crucial, the best option is ColStoch. When running time is important and a decrease in quality is allowed, ColStoch can still be significantly accelerated by JLT, i. e., ColStochJLT.

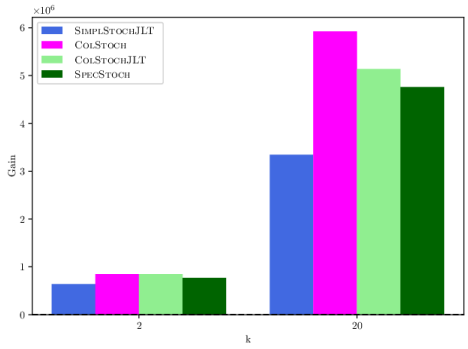
Our future plans include the extension of the problem to edge deletions. This problem is related to the protection of infrastructure and also important in corresponding applications.

*Acknowledgments:* We are grateful for coding support in early development stages by HU Berlin student Matthias Görg.

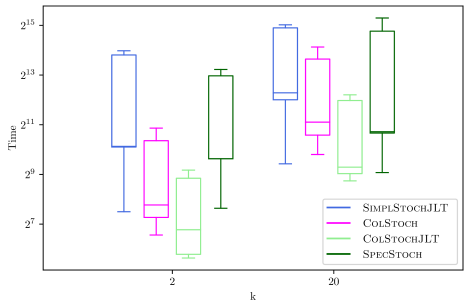
## References

[1] E. Angriman, R. Becker, G. D'Angelo, H. Gilbert, A. van der Grinten, and H. Meyerhenke. Group-harmonic and group-closeness maximization - approximation and engineering. In *Proc. of the Symp. on Algorithm Engineering and Experiments, ALENEX*, pages 154–168. SIAM, 2021.

[2] E. Angriman, M. Predari, A. van der Grinten, and H. Meyerhenke. Approximation of the diagonal of a laplacian's pseudoinverse for complex network analysis. In *ESA 2020, Italy*, volume 173, pages 6:1–6:24, 2020.




(a) Quality




(b) Running time (log scale)

Fig. 3. Aggregated results (via geometric mean) of $k$-GRIP on large graphs ($n \geq 57K$) for different $k$.


[3] E. Angriman, A. van der Grinten, M. von Looz, H. Meyerhenke, M. Nöllenburg, M. Predari, and C. Tzovas. Guidelines for experimental algorithmics: A case study in network analysis. *Algorithms*, 12(7):127, 2019.

[4] A.-L. Barabási and M. Pósfai. *Network science*. Cambridge University Press, Cambridge, 2016.

[5] U. Brandes and D. Fleischer. Centrality measures based on current flow. In *STACS*, pages 533–544. Springer Berlin Heidelberg, 2005.

[6] O. Cats, G.-J. Koppenol, and M. Warnier. Robustness assessment of link capacity reduction for complex networks: Application for public transport systems. *Reliability Engineering & System Safety*, 167:544–553, 2017.

[7] H. Cetinay, C. Mas-Machuca, J. L. Marzo, R. Kooij, and P. Van Mieghem. *Comparing Destructive Strategies for Attacking Networks*, pages 117–140. Springer International Publishing, 2020.

[8] F. Chung and L. Lu. *Complex graphs and networks*. 2004.

[9] W. Ellens, F. Spieksma, P. Van Mieghem, A. Jamakovic, and R. Kooij. Effective graph resistance. *Linear Algebra and its Applications*, 435(10):2491–2506, 2011.

[10] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23:298 – 305, 1973.

[11] A. Ghosh, S. Boyd, and A. Saberi. Minimizing effective resistance of a graph. Technical report, SIAM Review, 2005.

[12] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[13] I. Gutman and W. Xiao. Generalized inverse of the laplacian matrix and some applications. *Bulletin: Classe Des Sciences Mathematiques Et Natturalles*, 129:15–23, 2004.

[14] A. Hassidim and Y. Singer. Robust guarantees of stochastic greedy algorithms. In *Proc. of the 34th Intl. Conference on Machine Learning*, volume 70, pages 1424–1432. PMLR, 2017.

[15] V. Hernandez, J. E. Roman, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3):351–362, 2005.

[16] W. B. Johnson. Extensions of lipschitz mappings into hilbert space. *Contemporary mathematics*, 26:189–206, 1984.

[17] Y. Koç, M. Warnier, P. Van Mieghem, R. E. Kooij, and F. M. Brazier. A topological investigation of phase transitions of cascading failures in power grids. *Physica A: Statistical Mechanics and its Applications*, 415:273–284, 2014.

[18] J. Leskovec. Stanford Network Analysis Package (SNAP).

[19] O. Livne and A. Brandt. Lean algebraic multigrid (lamg): Fast graph laplacian linear solver. *SIAM Journal on Scientific Computing*, 34, 2011.

[20] L. M. Lovász. Random walks on graphs: A survey. pages 1–46, 1996.

[21] B. Manghiuc, P. Peng, and H. Sun. Augmenting the algebraic connectivity of graphs. In *28th European Symp. on Algorithms, ESA*, volume 173 of *LIPIcs*, pages 70:1–70:22. Schloss Dagstuhl, 2020.

[22] C. Mavroforakis, R. Garcia-Lebron, I. Koutis, and E. Terzi. Spanning edge centrality: Large-scale computation and applications. In *Proc. of the 24th Intl. Conference on World Wide Web*, page 732–742. Intl. World Wide Web Conferences Steering Committee, 2015.

[23] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. 1978.

[24] M. Minoux. Networks synthesis and optimum network design problems: Models, solution methods and applications. *Networks*, 19(3):313–360, 1989.

[25] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrák, and A. Krause. Lazier than lazy greedy. In *AAAI*, 2015.

[26] R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and cooperation in networked multi-agent systems. *Proc. of the IEEE*, 95(1):215–233, 2007.

[27] C. Paige. Accuracy and effectiveness of the lanczos algorithm for the symmetric eigenproblem. *Linear Algebra and its Applications*, 34:235–258, 1980.

[28] S. Perera, M. G. H. Bell, and M. C. J. Bliemer. Modelling supply chains as complex networks for investigating resilience: an improved methodological framework. 2015.

[29] C. Pizzuti and A. Socievole. A genetic algorithm for enhancing the robustness of complex networks through link protection. In *International Conference on Complex Networks and their Applications*, pages 807–819. Springer, 2018.

[30] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.

[31] D. F. Rueda, E. Calle, and J. L. Marzo. Robustness comparison of 15 real telecommunication networks: Structural and centrality measurements. *Journal of Network and Systems Management*, 25(2):269–289, Apr 2017.

[32] C. M. Schneider, A. A. Moreira, J. S. Andrade, S. Havlin, and H. J. Herrmann. Mitigation of malicious attacks on networks. *Proceedings of the National Academy of Sciences*, 108(10):3838–3841, 2011.

[33] J. Sherman and W. J. Morrison. Adjustment of an Inverse Matrix Corresponding to a Change in One Element of a Given Matrix. *The Annals of Mathematical Statistics*, 21(1):124 – 127, 1950.

[34] C. L. Staudt, A. Sazonovs, and H. Meyerhenke. Networkit: A tool suite for large-scale complex network analysis. *Network Science*, 4(4):508–530, 2016.

[35] T. Summers, I. Shames, J. Lygeros, and F. Dörfler. Correction to "topology design for optimal network coherence".

[36] T. Summers, I. Shames, J. Lygeros, and F. Dörfler. Topology design for optimal network coherence. In *2015 European Control Conference (ECC)*, pages 575–580. IEEE, 2015.

[37] T. H. Summers and M. Kamgarpour. Performance guarantees for greedy maximization of non-submodular controllability metrics. In *17th European Control Conf., ECC*, pages 2796–2801. IEEE, 2019.

[38] P. Van Mieghem, K. Devriendt, and H. Cetinay. Pseudoinverse of the laplacian and best spreader node in a network. *Phys. Rev. E*, 96:032311, 2017.

[39] H. Wang and P. Van Mieghem. Algebraic connectivity optimization via link addition. In *Proc. of the 3rd Intl. Conf. on Bio-Inspired Models of Network, Information and Computing Sytems*, pages 22:1–22:8. ICST, 2008.

[40] X. Wang, E. Pournaras, R. E. Kooij, and P. V. Mieghem. Improving robustness of complex networks via the effective graph resistance. *The European Physical Journal B*, 87:1–12, 2014.

[41] Y. Wei, R. Li, and W. Yang. Biharmonic distance of graphs. 2021.

[42] D. B. Wilson. Generating random spanning trees more quickly than the cover time. In *Proc. of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 296–303. Association for Computing Machinery, 1996.

[43] A. Yazdani and P. Jeffrey. Complex network analysis of water distribution systems. *Chaos (Woodbury, N.Y.)*, 21:016111, 03 2011.