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, 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 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 $S \subseteq \binom{V}{2} \setminus E$ 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 $O(n^3)$ time with standard tools in practice (where $n = |V|$). Overall, their approach leads to a running time of $O(kn^3)$, 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
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 – 68x 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 \( G \). \(^2\) \( L = D - A \) is the \( n \times n \) Laplacian matrix of \( G \), where \( D \) is the diagonal matrix of vertex degrees and \( A \) the adjacency matrix. \( L \) is symmetric, positive semi-definite and has zero row/column sum \( s.t., L1 = 0 \) where \( 1 \) is the all-ones vector. The \( m \times n \) incidence matrix \( B \) takes for \( e \in E \) and \( a \in V \) the values: \( B[e, a] = 1 \) if \( a \) is the destination of \( e \), \( B[e, a] = -1 \) if \( a \) is the origin of \( e \) and \( B[e, a] = 0 \) otherwise. For undirected graphs, the direction of each edge is specified arbitrarily. Moreover, \( L = B^T B \). It is well-known that \( L \) is not invertible, so its Moore-Penrose pseudoinverse \((L^+)\) is used instead, for which holds: \( LL^+ = L^+L = I - \frac{1}{\lambda_2}11^T \) [13]. Since \( L \) is symmetric, it has an orthonormal basis of eigenvectors \( U = [u_1, \ldots, u_n] \). We write the spectral decomposition as: \( L = \sum_{i=1}^n u_i \lambda_i u_i^T \), where the eigenvectors \( u_2, \ldots, u_n \) correspond to the ordered eigenvalues \( 0 < \lambda_2 \leq \ldots \leq \lambda_n \) (excluding the zero eigenvalue). For a graph \( G \) we use \( L_G \) \([L_G]^\dagger\) to refer to its Laplacian \([\text{Laplacian pseudoinverse}\]). If there is no subscript in our matrix notation, the associated graph is inferred by the context.

Let \( \Omega := \left\{ \frac{1}{2} \right\} \setminus E \). 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 \( r_G(\cdot, \cdot) \) between all vertex pairs:

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

\(^2\)Our methods can easily be extended to weighted graphs. However, for simplicity of notations we only consider non weighted graphs.

Computing \( r_G(a, b) \) can be done via \( L^+ \):

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

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

\[
\mathcal{R}(G) = n \text{tr}(L^+) \tag{3}
\]

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

\[
L_{G'}^\dagger = L_G^\dagger - \frac{1}{1 + r_{G}(a, b)}L_G^\dagger(e_a - e_b)(e_a - e_b)^T L_G^\dagger. \tag{4}
\]

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

\[
\text{gain}(a, b) = n \frac{\| L_G^\dagger[:, a] - L_G^\dagger[:, b] \|^2}{1 + r_{G}(a, b)} \tag{5}
\]

where \( L_G^\dagger[:, a] \) is the \( a \)th column of \( L^+ \). We rewrite Eq. (5) as a function of squared \( \ell_2 \) norms:

\[
\text{gain}(a, b) = n \frac{\| L^+[e_a - e_b] \|^2}{1 + \| B L^+[e_a - e_b] \|^2} = n \frac{b_G(a, b)}{1 + r_G(a, b)}. \tag{6}
\]

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

\[
r_G(a, b) = \| B L^+[e_a - e_b] \|^2 = (e_a - e_b)^T U A^{-1} U^T (e_a - e_b) = \sum_{i=2}^n \frac{(u_i[a] - u_i[b])^2}{\lambda_i} \tag{7}
\]

where \( A \) is the diagonal matrix of eigenvalues. Similarly:

\[
b_G(a, b) = \| L^+[e_a - e_b] \|^2 = (e_a - e_b)^T (L^+)^2 (e_a - e_b) = (e_a - e_b)^T U A^{-2} U^T (e_a - e_b) = \sum_{i=2}^n \frac{(u_i[a] - u_i[b])^2}{\lambda_i^2} \tag{8}
\]

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. Manghiciu 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 \( L \) as a preprocessing. Then, the marginal gains of all vertex pairs are computed via Eq. (5) in \( O(n^2) \) 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 \( O(kn^2) \), which is due to the evaluation of the gain function in \( k \) rounds on \( O(n^2) \) node pairs. The preprocessing takes \( O(n^3) \) time with standard tools. For monotonic submodular problems, the greedy algorithm has an approximation ratio of \( 1-\frac{1}{e} \). 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. The worst-case time complexities of all approaches are shown in Table I.

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

<table>
<thead>
<tr>
<th>Compute</th>
<th>#Evals \times SingleEval</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>STGREEDY</td>
<td>( O(n^2) )</td>
<td>( O(kn^2) \times O(n) )</td>
</tr>
<tr>
<td>SIMPLSTOCH</td>
<td>( O(n^2) )</td>
<td>( O(n^2) \times O(n) )</td>
</tr>
<tr>
<td>COLSTOCH</td>
<td>( O(n \log n) )</td>
<td>( O(n^2) \times O(n) )</td>
</tr>
<tr>
<td>SIMPLSTOCHJLT</td>
<td>( O(n \log n) )</td>
<td>( O(n^2) \times O(n \log n) )</td>
</tr>
<tr>
<td>COLSTOCHJLT</td>
<td>( O(n \log n \log n^2) )</td>
<td>( O(n^2) \times O(n \log n) )</td>
</tr>
<tr>
<td>SPECSTOCH</td>
<td>( O(cn) )</td>
<td>( O(n^2) \times O(c) )</td>
</tr>
</tbody>
</table>

### Algorithm 1 General framework for \( k \)-GRIP

1: function GreedyFramework\((G, k, \delta)\)  
2: Input: Graph \( G = (V, E) \), \( k \in \mathbb{Z}_{>0} \), accuracy \( 0 < \delta < 1 \)  
3: Output: \( G_k \) – graph after \( k \) edge insertions  
4: \( G_0 \leftarrow G \)  
5: ComputeOBJ\((G_0, \ldots)\)  
6: \( s \leftarrow \text{CANDIDATE_SIZE}(m, n, k, \delta) \)  
7: for \( r \leftarrow 0, \ldots, k-1 \) do  
8: \( S \leftarrow \text{CANDIDATES}(S, G_r, \ldots) \)  
9: for each \( (a, b) \in S \times S \) do  
10: \( \text{gain}(a, b) \leftarrow \text{EVAL}(a, b, \ldots) \)  
11: \( (a^*, b^*) \leftarrow \arg\max_{a \in S \times S} \text{gain}(a, b) \)  
12: \( G_{r+1} = G_r \cup \{a^*, b^*\} \)  
13: \( \text{UPDATE}(G_{r+1}, \ldots) \)  
14: return \( G_{r+1} \)
reduced sample $S$. 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 \textsc{SimplStoch}, any sample $S$ is a subset of $\binom{V}{2} \setminus E$. During each iteration of the main loop we sample uniformly at random $s := \frac{n^2 - mn}{k} \log \frac{1}{\delta}$ 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 \textsc{STGreedy}. More precisely, $L^1$ 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, $L^1$ is updated accordingly via Eq. (4) (Line 13). The cost of the main loop for \textsc{SimplStoch} is reduced compared to greedy by a factor of $k$. Yet, computing $L^1$ is still very time- and space-consuming.

B. \textsc{ColStoch}

Our first improvement upon \textsc{SimplStoch} avoids the full pseudoinversion of $L$, 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 $L^1$; precisely those corresponding to vertices $a$ and $b$. That is why, instead of sampling elements from $\binom{V}{2} \setminus E$, \textsc{ColStoch} restricts the sampling process to elements from $V$, the set of $L^1$ columns. Carefully selecting $S$ is critical as it affects the quality of the solution. Even if our problem is not submodular, we choose the default sample size of $s = n \sqrt{\frac{1}{2} \cdot \log\left(\frac{1}{\delta}\right)}$ elements (Line 6), leading to $n^2 \log \frac{1}{\delta}$ evaluations over all iterations, similar to \textsc{SimplStoch}. Moreover, to limit the quality loss, we choose elements of $S$ 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 \textsc{Candidates}() also receives those sampling probabilities in each iteration (Line 8). Once $S$ is determined, we compute all columns of $L^1$ corresponding to vertices in $S$.\footnote{\textbf{4}The only difference here is that we sample pairs of $L^1$ 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.} We do so by solving $s$ linear systems. More precisely, we solve one linear system for each vertex $a \in S$: $Lx = e_a - \frac{1}{n} \cdot 1$, where $1 = (1, \ldots, 1)^T$ and $x \perp 1$. Then, \textsc{ColStoch} performs function evaluations only between vertex pairs in $S \times S$ (Line 10). Finally, to further improve the overall running time, we do not update $L^1_{\mathcal{S}[\cdot, \cdot]}$ for all $a \in S$ at the end of each round (Line 13 of Algorithm 1). Instead, we update individual columns of $L^1$ on demand; only if the corresponding vertices participate in the candidate set $S$ of the following round.

To update previously computed columns, we use the out-
dated 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.

\textbf{4}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.

\textbf{5}The step is still very time- and space-consuming.

\textbf{6}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. \textsc{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: $b_G(a, b) = \|L^1(e_a - e_b)\|^2$ and $r_G(a, b) = \|B^T L^1(e_a - e_b)\|^2$ (Eq. (6)). Viewing $b_G(a, b)$ and $r_G(a, b)$ as pair-wise distances between vectors in $\{L^1\}_{a \in V}$ and $\{B^T L^1\}_{a \in V}$ (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 $O(\log n)$ random vectors. The JLT lemma states the following:

\textbf{Lemma IV-C.1} (Johnson-Lindenstrauss): Given fixed vectors $u_1, \ldots, u_n \in \mathbb{R}^d$ and $\eta > 0$, let $Q \in \mathbb{R}^{d \times d}$ 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 $1 - 1/n$

\begin{align}
(1 - \eta) ||u_i - u_j||^2 \leq ||Qu_i - Qu_j||^2 \leq (1 + \eta) ||u_i - v_j||^2
\end{align}

(9)

for all pairs $i, j \leq n$.\footnote{\textbf{6}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.}
Using Lemma IV-C.1, we can simply project matrices \( \mathbf{L} \) and \( \mathbf{B}^T \mathbf{L} \) 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 approximation technique with fast linear solvers, one can approximate distances between vertex pairs within a factor of \( (1 \pm \epsilon) \) in \( \mathcal{O}(I(n,m) \log n/n^2) \) 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 \( \| \mathbf{PL}^T (\mathbf{e}_a - \mathbf{e}_b) \|^2 \) and \( \| \mathbf{QBL}^T (\mathbf{e}_a - \mathbf{e}_b) \|^2 \), 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\(^6 \) \( \mathbf{Y} = \mathbf{L}^T \mathbf{P}^T \) to express effective resistances:

\[
\| \mathbf{QBL}^T \mathbf{Y}(\mathbf{e}_a - \mathbf{e}_b) \|^2 = (\mathbf{e}_a - \mathbf{e}_b) \mathbf{L}^T \mathbf{1} \mathbf{1}^T \mathbf{L}^T (\mathbf{e}_a - \mathbf{e}_b) = \| \mathbf{B}^T \mathbf{L}^T (\mathbf{e}_a - \mathbf{e}_b) \|^2
\]

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

We can integrate the JLT approximation both in the context of \textsc{ColStoch} and \textsc{SimpleStoch} (having \textsc{ColStochJLT} and \textsc{SimpleStochJLT} respectively). Let us consider the case of \textsc{ColStochJLT}: Again, the update step is performed after selecting set \( \mathcal{S} \) (just after Line 8). Indeed, we compute the vectors in \( \{ \mathbf{L}^T \}_a \in \mathcal{S} \) and \( \{ \mathbf{B}^T \mathbf{L}^T \}_a \in \mathcal{S} \) for \( G_0 \), where \( |\mathcal{S}| = \sqrt{n} \log \frac{\lambda}{\lambda_0} \). Since, later, we only perform evaluations for pairs in \( \mathcal{S} \times \mathcal{S} \), it suffices to consider projections onto \( \log \) rows (via \( \mathbf{P} \in \mathbb{R}^{\log n \times n} \) and \( \mathbf{Q} \in \mathbb{R}^{\log m \times n} \)). During the main loop of Algorithm 1 we perform the same number of overall function evaluations as in \textsc{ColStoch}, that is \( \mathcal{O}(n^2) \). However, now a single function evaluation for an arbitrary vertex pair takes \( \mathcal{O}(\log n) \) via the formula gain\((a,b) \approx \left\| \mathbf{PL}^T (\mathbf{e}_a - \mathbf{e}_b) \right\|^2 \) (up to a relative error of \( \eta \)). For the update step, we need to sample new projections \( \mathbf{P} \) and \( \mathbf{Q} \) and recompute the two matrices \( \mathbf{PL}^T \) and \( \mathbf{BQ}^T \mathbf{Y} \).

D. \textsc{SimpleStoch}

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:

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

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 \( \mathbf{L}^\dagger \).

\( ^6 \)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 \( \eta = 0.55 \).
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 $L$ (corresponding to $G_0$). We do so using standard iterative methods, such as the Lanczos algorithm [27], which often takes $O(cm)$ time for sparse matrices, depending on the desired accuracy and eigenvalue distribution. During the main loop, the algorithm performs $O(n^2)$ function evaluations (dictated by the stochastic approach). Single function evaluations in Line 10 require only $O(c)$ 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 whose vertex count is $< 57K$. 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 $k = 2, 5, 20, 50, 100$ 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 (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.

Additionally, we perform configuration experiments to determine the quality of the gain approximation via Eq. (15) for SIMPLSTOCH. 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 $100$. 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 Slepe [15] library.

\begin{table}[h]
\centering
\caption{Summary of graph instances, providing (in order) network name, vertex count, and edge count.}
\begin{tabular}{|c|c|c|}
\hline
Graph & $|V|$ & $|E|$ \\
\hline
inf-power & 4K & 6K \\
facebook-ego-combined & 4K & 8.8K \\
web-spam & 4K & 37K \\
Wiki-Vote & 7K & 100K \\
p2p-Grunetelld09 & 8K & 2.6K \\
p2p-Grunetelld04 & 10K & 39K \\
web-indochina & 11K & 47K \\
cac-Hepph & 11K & 117K \\
web-webbase-2001 & 16K & 25K \\
axriv-astro-ph & 17K & 196K \\
as-caida20071105 & 26K & 53K \\
cit-HeptTh & 27K & 352K \\
ia-email-EU & 32K & 54.4K \\
\hline
\end{tabular}
\end{table}

\begin{table}[h]
\centering
\caption{Quality and Speedup of SIMPLSTOCH (RELATIVE TO STGREEDY) FOR DIFFERENT APPROXIMATION BOUND.}
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
SIMPLSTOCH & & Relative Quality & & & & \\
& & $\delta = .9$ & $\delta = .99$ & $\delta = .9$ & $\delta = .99$ & \\
& $k = 2$ & $k = 5$ & $k = 20$ & $k = 50$ & $k = 100$ \\
\hline
$\delta = .9$ & 0.9662 & 0.9610 & 0.9696 & 0.9840 & 0.9898 \\
$\delta = .99$ & 0.9239 & 0.9241 & 0.9442 & 0.9559 & 0.9694 \\
\hline
\end{tabular}
\end{table}

\begin{table}[h]
\centering
\caption{Relative Speedup of SIMPLSTOCH (RELATIVE TO TOCHGREEDY) FOR DIFFERENT APPROXIMATION BOUND.}
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
SIMPLSTOCH & & Relative Speedup & & & & \\
& & $\delta = .9$ & $\delta = .99$ & $\delta = .9$ & $\delta = .99$ & \\
& $k = 2$ & $k = 5$ & $k = 20$ & $k = 50$ & $k = 100$ & \\
\hline
$\delta = .9$ & 2.6 & 2.6 & 2.7 & 2.7 & 3.1 & \\
$\delta = .99$ & 4.0 & 3.9 & 4.2 & 4.1 & 4.6 & \\
\hline
\end{tabular}
\end{table}
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, SIMPLSTOCHLT and COLSTOCHLT 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× (SIMPLSTOCHLT) and 68× (COLSTOCHLT) faster than STGREEDY (Fig. 2(b)). The scaling of COLSTOCHLT is worse than that of SIMPLSTOCHLT for large $k$. This is due to the update step of Algorithm 1, where COLSTOCHLT 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.\footnote{The time limit is set to 43,200 seconds (12 hours).} The best approaches for large graphs are COLSTOCHLT and COLSTOCH. Both of them produce the highest quality results and COLSTOCHLT 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.

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 $L[i, i]$ 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., COLSTOCHLT.

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 G"org.

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


