Greedy optimization of resistance-based graph robustness with global and local edge insertions

Predari, Maria; Berner, Lukas; Kooij, Robert; Meyerhenke, Henning

DOI
10.1007/s13278-023-01137-1

Publication date
2023

Document Version
Final published version

Published in
Social Network Analysis and Mining

Citation (APA)
Predari, M., Berner, L., Kooij, R., & Meyerhenke, H. (2023). Greedy optimization of resistance-based graph robustness with global and local edge insertions. *Social Network Analysis and Mining*, *13*(1), Article 130. https://doi.org/10.1007/s13278-023-01137-1

Social Network Analysis and Mining (2023) 13:130
https://doi.org/10.1007/s13278-023-01137-1

ORIGINAL ARTICLE

Greedy optimization of resistance-based graph robustness with global and local edge insertions

Maria Predari[1] · Lukas Berner[1] · Robert Kooij[2,3] · Henning Meyerhenke[1]

Received: 6 April 2023 / Revised: 8 May 2023 / Accepted: 11 September 2023

Abstract

The total effective resistance, also called the Kirchhoff index, provides a robustness measure for a graph G. We consider two optimization problems of adding k new edges to G such that the resulting graph has minimal total effective resistance (i.e., is most robust)—one where the new edges can be anywhere in the graph and one where the new edges need to be incident to a specified focus node. 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 an established generic greedy heuristic. 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 larger graphs for which the application of the state-of-the-art greedy approach was impractical before.

Keywords Graph robustness · Optimization problem · Effective resistance · Kirchhoff index · Laplacian pseudoinverse

1 Introduction

The analysis of network topologies has received considerable attention in various fields of science and engineering in the last decades (Barabási and Pósfai 2016; Newman 2018). Its purpose usually is to better understand the functionality, dynamics, and evolution of a network[1] and its components (Barabási and Pósfai 2016). 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 (Freitas et al. 2022). As an example, one may ask whether the network is guaranteed to remain connected if an edge is deleted, e.g., due to failure or an attack. Network robustness is a critical design issue in many areas, including telecommunication (Rueda et al. 2017), power grids (Koç et al. 2014), public transport (Cats et al. 2017), supply chains (Perera et al. 2015) and water distribution (Yazdani and Jeffrey 2011).

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

A preliminary version of this paper appeared in the Proc. of *2022 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (ASONAM) (Predari et al. 2022). 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.

✉ Robert Kooij
r.e.kooij@tudelft.nl

Maria Predari
predarim@hu-berlin.de

Lukas Berner
lukas.berner@hu-berlin.de

Henning Meyerhenke
meyerhenke@hu-berlin.de

[1] Department of Computer Science, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany

[2] Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands

[3] UNIT ICT, Strategy and Policy, TNO (Netherlands Organisation for Applied Scientific Research), P.O. Box 96800, 2509 JE The Hague, The Netherlands

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

Published online: 12 October 2023

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 $X \subset \binom{V}{2} \setminus E$ of size k that optimizes the robustness of G. We call this problem k-GRIP, short for *global robustness improvement problem*. A related task fixes a focus node $v \in V$ from which k edges can be inserted into G to other nodes; we call this problem k-LRIP, short for *local 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 (Barabási and Pósfai 2016; Rueda et al. 2017).

One established measure for k-GRIP, which was shown to be a good robustness indicator in various scenarios (Ellens et al. 2011; Ghosh et al. 2008; Wang et al. 2014), 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 (Angriman et al. 2020), random walks (Lovász 1996), and several centrality measures (Mavroforakis et al. 2015; Brandes and Fleischer 2005). In fact, it works similarly as an objective function for k-LRIP—we are just restricted in the search space to a particular focus node. To compute the total effective resistance, one sums the effective resistance over all vertex pairs in G (for technical details, see Sect. 2). Intuitively, the effective resistance becomes small if there are many short paths between two vertices. Removing an edge in such a case hardly disrupts the connectivity, since there are usually alternative paths. Due to this favorable property, we select total effective resistance in this paper as the robustness measure for k-GRIP and k-LRIP.

The effective graph resistance-based k-GRIP version, recently shown to be $\mathcal{NP}$ hard (Kooij and Achterberg 2023), was already considered by Summers et al. (2015). It was shown in Summers et al. (2017) that k-GRIP for the effective graph resistance is not submodular, hence without an approximation guarantee for the greedy algorithm (more details in Sect. 3). Still, even without an approximation guarantee, this greedy algorithm provides very good empirical results—for small networks it does so in reasonable time. It should be noted that the example given in Summers et al. (2017), which proves that k-GRIP for the effective graph resistance is non-submodular, also proves that k-LRIP is non-submodular for the effective graph resistance.

The greedy algorithm performs k iterations, at each step adding the edge with the 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, this 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 (Mirzasoleiman et al. 2015) has been shown to be potentially much faster—while usually producing solutions of nearly the same quality. It does so by *sampling* from the set of candidates to find the one with highest gain (from the sample instead of from the whole set) in each iteration. Our hypothesis for this paper is that this favorable speed-quality trade-off of stochastic greedy holds for our k-GRIP as well. We also assume that other Laplacian approximation techniques can speed up the required computations. Furthermore, we hope that the techniques that work well for the k-GRIP problem also work well (if adapted properly) for the related k-LRIP problem. Some differences in the speed-quality trade-off are to be expected.

Building upon the generic stochastic greedy approach (Mirzasoleiman et al. 2015), we first devise several heuristic strategies for k-GRIP that leverage both graph- and matrix-related properties (Sect. 4). Our approaches 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 (Sect. 6) 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, testing graphs with $< 57K$ nodes, we produce results that are on average 2–15% away from the greedy solution, while running 3.3–68× faster than the state of the art (SotA). Finally, we demonstrate that we can now process much larger graphs for which the application of the SotA greedy approach was infeasible before.

Besides a better update strategy for our heuristic ColStoch, another extension of this paper compared to its conference version (Predari et al. 2022) consists of the k-LRIP part (Sect. 5). The corresponding experiments in Sect. 6 show that our heuristics (except one) work for this problem similarly well when the graphs are sufficiently large. For example, on graphs with more than 10,000 nodes, one of our new heuristics is $\approx 10\%$ away from the greedy quality, but on average $\approx 2 - 7\times$ faster (depending on k and the graph).

2 Preliminaries

We assume that our input consists first of all of a connected, undirected, and simple graph $G = (V, E)$ with n vertices and m edges. For both k-GRIP and k-LRIP, we also have an integer $k \in \mathbb{Z}_{>0}$ for the number of edges to be added to G; k-LRIP additionally requires the focus node $v \in V$ from which the additional edges are inserted. Our methods can be easily extended to weighted graphs. However, for the sake

of presentation simplicity, we only consider unweighted graphs.

For the remainder, we use several well-known matrix representations of graphs. $\mathbf{L} = \mathbf{D} - \mathbf{A}$ is the $n \times n$ Laplacian matrix of G, where $\mathbf{D}$ is the diagonal matrix of vertex degrees and $\mathbf{A}$ is 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 that its Moore–Penrose pseudoinverse ($\mathbf{L}^\dagger$) is used instead, for which holds: $\mathbf{L}\mathbf{L}^\dagger = \mathbf{L}^\dagger\mathbf{L} = \mathbf{I} - \frac{1}{n} \cdot \mathbf{1}\mathbf{1}^T$ (Gutman and Xiao 2004). 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 $\Omega_G := \binom{V}{2} \setminus E$. For any $X \subset \Omega_G$, 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 a set $X \subset \Omega_G$ 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 (for which lower values indicate higher robustness), which is the sum of pairwise effective resistances $\mathbf{r}_G(\cdot,\cdot)$ between all vertex pairs:

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

Thus, k-GRIP for total effective resistance asks to find the set X of size k that minimizes the resistance of the graph resulting from inserting the edges of X. The notion of effective resistance comes from viewing G as an electrical circuit in which each edge e is a resistor with resistance $1/\mathbf{w}[e]$. Following fundamental electrical laws, the effective resistance $\mathbf{r}(a,b)$ between two vertices a and b is the potential difference between a and b when a unit current is injected into G at a and extracted at b.

The second problem we address is the related k-LRIP problem. It also uses total effective resistance $\mathcal{R}(G)$ as the objective function. The main difference is that it restricts the search space by limiting the insertion of the k edges to a particular focus node $v \in V$ that is part of the input. The set X of edges to insert is selected from the vertex pairs $\Omega_v := \{(v,u) \mid u \in V, \{v,u\} \notin E\}$.

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

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

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 $\mathcal{R}$ by the insertion of $\{a,b\}$ is $\mathcal{R}(G) - \mathcal{R}(G')$ and relies on $\mathbf{L}^\dagger_{G'}$ (Sherman–Morrison formula (Sherman and Morrison 1950)):

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

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

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

where $\mathbf{L}^\dagger_{\mathbf{G}}[:,i]$ is the i^{th} column of $\mathbf{L}^\dagger$. We rewrite Eq. (5) as a function of squared ℓ_2 norms:

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

where $\mathbf{b}_\mathbf{G}(\cdot,\cdot)$ is known as the biharmonic distance of G (Yi et al. 2018; Wei et al. 2021). Finally, we express these distances via the spectral decomposition of $\mathbf{L}^\dagger$ (or $\mathbf{L}$, respectively):

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

where $\mathbf{\Lambda}$ is the diagonal matrix of the eigenvalues of $\mathbf{L}^\dagger$. Similarly,

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

3 Related work

Robustness of networks has been an active research area for decades (Pizzuti and Socievole 2018; Freitas et al. 2022). Several authors have proposed the use of specific network metrics to quantify the robustness of a given network, see, e.g., Rueda et al. (2017), Fiedler (1973), Schneider et al.

(2011), Cetinay et al. (2020). In a recent survey on the topic, Freitas et al. (2022) classify robustness metrics into three types: metrics based on structural properties, such as edge connectivity or diameter; metrics based on the spectrum of the adjacency matrix, such as the spectral radius or spectral gap; and metrics based on the spectrum of the Laplacian matrix, for instance the algebraic connectivity and the effective graph resistance. Here, the algebraic connectivity, i.e., the second smallest eigenvalue λ_2 of the graph's Laplacian (Fiedler 1973), is known to capture the overall connectivity of a graph. This metric is also related to synchronization of networks, including opinion dynamics (Olfati-Saber et al. 2007).

Once the robustness of a network has been established, a natural next step is to determine how robustness can be improved. Schneider et al. (2011) view the relative size of the largest connected component as robustness measure (after removing a certain fraction of the edges) and rewire the edges for robustness improvement. A second approach is to add elements to the network. Several researchers investigated k-GRIP for specific robustness metrics. For instance, Wang and Van Mieghem (2008) considered 1-GRIP, with the robustness metric being the algebraic connectivity. They suggest several strategies, based upon topological and spectral properties of the graph, to decide which single link to add to the network in order to increase the algebraic connectivity as much as possible. Reference (He 2020, Chapter 8) also considered algebraic connectivity for k-GRIP. Under some light conditions, lower bounds for the quality of the greedy solution were obtained. It might be argued that the algebraic connectivity is not a proper robustness metric, because there are examples where adding a link to a graph does not change the algebraic connectivity, see Jun et al. (2010). The $\mathcal{NP}$-hardness of k-GRIP for algebraic connectivity was proved in Mosk-Aoyama (2008). Manghiuc et al. (2020) consider a weighted decision variant of k-GRIP w.r.t. λ_2. They propose an almost-linear time algorithm that augments the graph by k edges such that λ_2 exceeds a specified threshold. A nice overview of algebraic connectivity for k-GRIP is presented in Li et al. (2018).

Reference Papagelis (2015) shows that k-GRIP with the average shortest path length as a robustness metric does not satisfy the submodularity constraint, but accurate greedy solutions can be obtained. Van Mieghem et al. (2011) consider a link removal problem with the spectral radius (largest eigenvalue of adjacency matrix) as a robustness metric and prove this problem is $\mathcal{NP}$-hard. Baras and Hovareshti (2009) consider the problem of adding k links to a given network, such that the number of spanning trees in the graph is maximized.

Effective graph resistance as a robustness measure dates back at least to Ellens et al. (2011). It has been known much longer, however, that effective resistance is proportional to commute times of random walks (Ghosh et al. 2008). Wang et al. (2014) and Pizzuti and Socievole (2018) investigate heuristics for 1-GRIP with effective graph resistance (both for edge insertion *and* removal). Besides deriving theoretical bounds, Wang et al. (2014) 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 trade-off between running time and the robustness gain.

Pizzuti and Socievole (2018, 2023) proposed and evaluated several genetic algorithms to find the optimal edge to add, in order to minimize R_G. Clemente and Cornaro (2020) studied k-GRIP for the effective graph resistance and gave lower bounds for R_G upon the addition of k links, under some mild conditions for k. For $k = 1$, the lower bound in Clemente and Cornaro (2020) clearly outperforms the lower bound in Wang et al. (2014).

The state-of-the-art heuristic for k-GRIP is a greedy algorithm presented by Summers et al. (2015), 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 the robustness measure). To this end, StGreedy computes the full pseudoinverse of $\mathbf{L}$ as a preprocessing step. 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 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 Summers et al. (2017) for a counterexample), the greedy algorithm still often leads to solutions of high quality (Summers and Kamgarpour 2019; Angriman et al. 2021). Stochastic greedy algorithms that improve the time complexity of the standard greedy approach (in a general setting) were proposed in Mirzasoleiman et al. (2015); and Hassidim and Singer (2017). These algorithms use random sampling techniques and reduce the total number of function evaluations (roughly) by a factor of k. They achieve provable approximation guarantees in cases where the greedy algorithm admits them, too.

Also, k-LRIP has been considered by several authors—for different objectives. Shan et al. (2018) consider the node resistance (as robustness metric or rather as a centrality measure), which is the sum of the effective resistance from one source node v to all other nodes. They assume that the k links that are to be added are chosen from the set of non-existing links from the focus node v; not all possible non-existing links. It is shown by the authors that in this setting,

Table 1 Time complexities (assuming standard (pseudo) inversion tools, linear solvers, and eigensolvers used in practice for Laplacians of general graphs) of all approaches involved

	Compute	#Evals × SingleEval	All updates	Memory
StGreedy	$\mathcal{O}(n^3)$	$\mathcal{O}(kn^2) \times \mathcal{O}(n)$	$\mathcal{O}(kn^2)$	$\mathcal{O}(n^2)$
SimplStoch	$\mathcal{O}(n^3)$	$\mathcal{O}'(n^2) \times \mathcal{O}(n)$	$\mathcal{O}(kn^2)$	$\mathcal{O}(n^2)$
ColStoch	$\widetilde{\mathcal{O}}(sm \log n)$	$\mathcal{O}'(n^2) \times \mathcal{O}(n)$	$\widetilde{\mathcal{O}}(ksm \log n)$	$\mathcal{O}((s+\tau)n+m)$
SimplStochJLT	$\widetilde{\mathcal{O}}(m \log n)$	$\mathcal{O}'(n^2) \times \mathcal{O}(\log n)$	$\widetilde{\mathcal{O}}(km \log n)$	$\mathcal{O}((s+\log n)n+m)$
ColStochJLT	$\widetilde{\mathcal{O}}(m \log n \log s)$	$\mathcal{O}'(n^2) \times \mathcal{O}(\log s)$	$\widetilde{\mathcal{O}}(km \log n \log s)$	$\mathcal{O}((\log s+\tau)n+m)$
SpecStoch	$\mathcal{O}(cm)$	$\mathcal{O}'(n^2) \times \mathcal{O}(c)$	$\mathcal{O}(kcm)$	$\mathcal{O}(cn+m)$

Columns correspond to major steps of Algorithm 1. In general, the dominant term comes from the total number of evaluations and their time to be evaluated (second column). The $\widetilde{\mathcal{O}}$-notation hides $\log(1/\epsilon)$ factors, where ϵ is the accuracy threshold of the linear solver. The $\mathcal{O}'$-notation hides $\log(1/\delta)$ factors, where δ determines the sample size in the stochastic candidate selection. Note that we consider the Johnson–Lindenstrauss transform (JLT) parameter η here as a constant. τ is the number of uniform spanning trees (USTs) required for the diagonal approximation in ColStoch, which depends on the diameter of the graph (Angriman et al. 2020). More details in the text

the node resistance is a supermodular set function. Ref. Bergamini et al. (2018) consider k-LRIP with betweenness centrality. In fact, k-LRIP has been studied with a variety of other centrality metrics, such as PageRank (Avrachenkov and Litvak 2006; Olsen and Viglas 2014), closeness centrality (Crescenzi et al. 2016), and eccentricity (Demaine and Zadimoghaddam 2010; Perumal et al. 2013).

Besides using the stochastic greedy algorithm for both k-GRIP and k-LRIP, we intend to accelerate the optimization process by approximation techniques for the effective resistance values. While Shan et al. (2018) also employ a greedy algorithm for k-LRIP, their objective function and acceleration techniques differ from ours.

4 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 existing randomized techniques 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. Functions not defined explicitly in the pseudocode are described in detail in the text. The time and space complexities of all approaches (assuming standard tools) are shown in Table 1.

Algorithm 1 General framework for k-GRIP

```
1:  function GreedyFramework(G, k, δ)
2:      Input: Graph G = (V, E), k ∈ ℕ_{>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}
```

For submodular functions, the greedy framework can be combined with a lazy technique (Minoux 1978) 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. An observation important for us is: "The 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" (Minoux 1989). Based on the above observation and existing, positive results on the lazy greedy approach for k-GRIP (Summers et al. 2015), we also employ this technique and do so by means of a priority queue. Entries in the priority queue are of the form $(e, g(e), r)$, where $e \in \binom{V}{2} \setminus E$, $g(e)$ is the marginal gain of e, and $r \in \mathbb{N}_{>0}$ is the round in which the gain was computed.

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

4.1 SimplStoch

Our first idea is to simply apply the generic randomized technique proposed in generic form by Mirzasoleiman et al. (2015) in the context of k-GRIP. The main idea of Mirzasoleiman et al. (2015) is to not inspect all possible elements for insertion, but only a reduced sample $\mathcal{S}$. For non-negative monotone submodular functions (which does not hold for k-GRIP), the stochastic greedy approach provides an approximation ratio of $1 - e^{-(1-\delta)}$, where $0 \leq \delta \leq 1$ is an accuracy parameter.

Regarding SimplStoch, any edge from $\mathcal{S} \times \mathcal{S}$ is a subset of $\binom{V}{2} \setminus E$; during each iteration of the main loop, we sample uniformly at random $s := \frac{n(n-1)/2-m}{k} \log(\frac{1}{\delta})$ vertex pairs (Line 8 in Algorithm (1)), resulting in $\mathcal{O}((n^2 - m)\log(\frac{1}{\delta}))$ function evaluations overall. Those are performed via the Laplacian pseudoinverse obtained during preprocessing, 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 $k/\log(1/\delta)$. Yet, computing $\mathbf{L}^\dagger$ is still very time- and space-consuming.

4.2 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 columns of $\mathbf{L}^\dagger$. 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 $\mathcal{O}(n^2 \log(\frac{1}{\delta}))$ evaluations over all iterations, similar to SimplStoch. 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.

Moreover, to limit the quality loss, we choose elements of $\mathcal{S}$ following graph-based sampling probabilities (details in Sect. 4.2.1). 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 $\mathcal{S}$. 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.

We compute the columns corresponding to $\mathcal{S}$ 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 $\mathbf{x} \perp \mathbf{1}$. Once the sample set $\mathcal{S} \subset V$ is determined, 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 number.

4.2.1 diag($\mathbf{L}^\dagger$) strategy

Let us now explain the sampling probabilities for selecting $\mathcal{S}$. Following previous studies (Van Mieghem et al. 2017; Wang et al. 2014), 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 our immediate context. Firstly, 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 mean to (pseudo)invert $\mathbf{L}$—with the associated cost. To circumvent these issues, we sample vertices according to their corresponding diagonal entries in $\mathbf{L}^\dagger$. Recall from Sect. 2 that these entries are proportionate to the electrical farness of the corresponding nodes. In other words, 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) (Van Mieghem et al. 2017), which is why we select them with higher probability for an edge insertion.

Computing diag($\mathbf{L}^\dagger$) can be performed in almost-linear time by using the connection of effective resistance to uniform spanning trees (USTs) of G. A UST of G is a spanning tree drawn uniformly at random from the set of all spanning trees of G. Angriman et al. (2020) proposed an algorithm that approximates (effective resistances and) 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 $\mathcal{O}(\log n)$). We provide here some details necessary to understand our new update strategy (Sect. 4.2.2) when an edge is added.

Following fundamental electrical laws, the effective resistance $\mathbf{r}(u, v)$ of vertices u and v is the potential difference between u and v when a unit of current is injected into G at u and extracted at v. According to Ohm's law, whenever there is a potential vector $\mathbf{x} \in \mathbb{R}^{n \times 1}$ on the vertices of G, there is also an electrical flow $\mathbf{f} \in \mathbb{R}^{m \times 1}$ on the edges of the graph, equal to the potential differences and leading from the node with higher to the node with lower potential value. As a consequence, we can express $\mathbf{r}(u, v)$ (for any vertex pair (u, v)) as the sum of current flows on *any* path[2] $\langle u = v_0, v_1, \ldots, v_{k-1}, v_k = v \rangle$ as:

$$\mathbf{r}(u, v) = \sum_{i=0}^{k-1} \mathbf{f}[v_i, v_{i+1}] \tag{9}$$

Note that the sign of the current flow changes if we traverse an edge against the flow direction (and thus the sum may hide negative values when the direction is reversed). Equation (9) can also be written as, see Bollobás (1998)

$$\mathbf{r}(u, v) = 1/N \sum_{i=0}^{k-1} \left(N_{u,v}(v_i, v_{i+1}) - N_{u,v}(v_{i+1}, v_i) \right), \tag{10}$$

where $N_{u,v}(v_i, v_{i+1})$ is the number of spanning trees in which the (unique) path from u to v contains (v_i, v_{i+1}) in that order and N is the number of all spanning trees of the graph G. The main idea of Angriman et al. (2020) is to compute a sufficiently large sample of uniform spanning trees (USTs) in order to approximate the effective resistances according to Eq. (10). The resistance values are then used for approximating the diagonal entries of $\mathbf{L}^\dagger$, together with one column of $\mathbf{L}^\dagger$ derived from solving one linear system.

4.2.2 Updating approximate diag($\mathbf{L}^\dagger$) after edge insertions

For updating diag($\mathbf{L}^\dagger$) within 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) and save computations by reusing previously computed USTs corresponding to G_r. This dynamic approximation approach can also be useful in other contexts. The re-used trees are not uniformly distributed in the new graph $G_{r+1} := G_r \cup \{a, b\}$, however, 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 (Wilson 1996). The final sample set is the union of the reweighted USTs (originally from G_r) and the newly sampled USTs in G_{r+1}. We provide the details in the following.

To account for an edge insertion into G, let the set of all spanning trees of G (before the edge insertion) be denoted as $\mathcal{T} = \mathcal{T}_G$. When looking at the potential difference between two nodes u and v within one particular spanning tree T, then the electrical flow induced on each edge on the unique path from s to t in T is $1/N$. Using the principle of superposition for the electrical flow in G, we can then write $\mathbf{r}(u, v) = \sum_{i=0}^{k-1} \mathbf{f}[v_i, v_{i+1}] = \sum_{T \in \mathcal{T}} \sum_{i=0}^{k-1} \mathbf{f}^{(\mathbf{T})}[v_i, v_{i+1}]$, where $\mathbf{f}^{(\mathbf{T})}[\cdot]$ restricts the electrical flow to edges of the respective spanning tree T (edges not in T contribute 0 to the sum). In the following, we use $\mathbf{F}_{(\mathbf{u},\mathbf{v})}(T) := \sum_{i=0}^{k-1} \mathbf{f}^{(\mathbf{T})}[v_i, v_{i+1}]$ as short-hand notation for the sum of the flows. Now let G' be the new graph when an edge $e = \{u, v\}$ is added to the graph G. Let τ be a random variable from the uniform distribution over spanning trees of G. Then, $\mathbf{r}(u, v) = \mathbf{E}\left[\mathbf{F}_{(\mathbf{u},\mathbf{v})}(\tau)\right]$ and we are interested in computing their updated values upon edge insertions.

We define $\mathcal{T}' := \mathcal{T}_{G'}$. Let τ' be a uniformly distributed valued random variable over $\mathcal{T}'$. We consider $\mathbf{F}'_{(\mathbf{u},\mathbf{v})} : \mathcal{T}' \to \mathbb{R}$ and denote by $\mathbf{F}_{(\mathbf{u},\mathbf{v})} = \mathbf{F}'_{(\mathbf{u},\mathbf{v})}|_{\mathcal{T}} : \mathcal{T} \to \mathbb{R}$ its restriction to spanning trees of G.

[2] For the algorithm, it is beneficial to use shortest paths, though

Lemma 1 *Let G' be the graph resulting from inserting $e = \{u, v\}$ into G. Then,*

$$\mathbf{r}_{G'}(u,v) = \frac{\mathbf{r}_G(u,v)}{1+\mathbf{r}_G(u,v)}\mathbf{E}\left[\mathbf{F}'_{(\mathbf{u},\mathbf{v})}(\tau') \mid e \in \tau'\right] + \frac{1}{1+\mathbf{r}_G(u,v)}\mathbf{E}\left[\mathbf{F}_{(\mathbf{u},\mathbf{v})}(\tau') \mid \tau' \in \mathcal{T}\right]. \tag{11}$$

Proof Recall from above that $\mathbf{r}_{G'}(u,v) = \mathbf{E}\left[\mathbf{F}'_{(\mathbf{u},\mathbf{v})}(\tau')\right]$. Also note that for any edge $e' = \{u', v'\}$, it holds that its effective resistance equals the probability to be part of a UST. Now $\mathbf{E}\left[\mathbf{F}'_{(\mathbf{u},\mathbf{v})}(\tau')\right]$ can be computed by distinguishing whether e is contained in τ' or not:

$$\begin{aligned}
\mathbf{E}\left[\mathbf{F}'_{(\mathbf{u},\mathbf{v})}(\tau')\right] &= \mathbf{P}\left(e \in \tau'\right)\mathbf{E}\left[\mathbf{F}'_{(\mathbf{u},\mathbf{v})}(\tau') \mid e \in \tau'\right] \\
&\quad + \mathbf{P}\left(e \notin \tau'\right)\mathbf{E}\left[\mathbf{F}'_{(\mathbf{u},\mathbf{v})}(\tau') \mid e \notin \tau'\right] \\
&= \mathbf{P}\left(e \in \tau'\right)\mathbf{E}\left[\mathbf{F}'_{(\mathbf{u},\mathbf{v})}(\tau') \mid e \in \tau'\right] \\
&\quad + \mathbf{P}\left(e \notin \tau'\right)\mathbf{E}\left[\mathbf{F}_{(\mathbf{u},\mathbf{v})}(\tau') \mid \tau' \in \mathcal{T}\right] \\
&= \mathbf{r}_{G'}(u,v)\mathbf{E}\left[\mathbf{F}'_{(\mathbf{u},\mathbf{v})}(\tau') \mid e \in \tau'\right] \\
&\quad + (1-\mathbf{r}_{G'}(u,v))\mathbf{E}\left[\mathbf{F}_{(\mathbf{u},\mathbf{v})}(\tau') \mid \tau' \in \mathcal{T}\right] \\
&= \frac{\mathbf{r}_G(u,v)}{1+\mathbf{r}_G(u,v)}\mathbf{E}\left[\mathbf{F}'_{(\mathbf{u},\mathbf{v})}(\tau') \mid e \in \tau'\right] \\
&\quad + \frac{1}{1+\mathbf{r}_G(u,v)}\mathbf{E}\left[\mathbf{F}_{(\mathbf{u},\mathbf{v})}(\tau') \mid \tau' \in \mathcal{T}\right],
\end{aligned} \tag{12}$$

using $\mathbf{P}\left(e \in \tau'\right) = \mathbf{r}_{G'}(u,v) = \frac{\mathbf{r}_G(u,v)}{1+\mathbf{r}_G(u,v)}$ (the latter equation follows from (Ranjan et al. 2014, Cor. 3) by setting $u = x = i$ and $v = y = j$) and the fact that $\mathcal{T}$ equals $\mathcal{T}' \setminus \mathcal{T}_e$, where $\mathcal{T}_e$ is the set of trees containing e. □

Adapting the UST Algorithm

The second term in Eq. (12) can be approximated using the USTs of G, which are already available from previous rounds of the algorithm. To approximate the first term, one can sample spanning trees of G' which contain e. For this, we use Algorithm 2, which is a slight adaptation of Wilson's algorithm with a modified starting state. A spanning tree which contains $\{u, v\}$ can be reinterpreted as a forest with two components by removing $\{u, v\}$. Thus, we initialize our version of Wilson's algorithm with a forest T with two components where each component contains only one of u and v. Then, in each iteration we generate a loop-erased random walk from a random vertex until it hits a node in T.

Proposition 2 *The distribution of forests T sampled by Algorithm 2 is the uniform distribution on the set of all spanning trees which contain the edge $\{u, v\}$.*

Proof Avena et al. (2018) reformulate Wilson's algorithm for uniform spanning forests (USFs) and multiple roots (one for each tree in the forest). That is why we set u and v as the roots of two separate trees, and let the algorithm compute a USF with two trees. The two trees in the USF are then linked by the edge $\{u, v\}$, resulting in a spanning tree T' of G'. By the USF property of the two trees above the claim follows. □

Algorithm 2 Algorithm for sampling a UST of G containing a fixed edge $\{a, b\}$

```
function SAMPLING(G, a, b)
    Input: Graph G = (V, E), edge {a, b} ∈ E
    Output: T: UST of G containing {a, b}
    T_1 ← tree consisting of a
    T_2 ← tree consisting of b
    Let x_1, ..., x_{n-2} be an arbitrary ordering of V \ {a, b}
    for i ← 1, ..., n − 2 do
        P ← a random walk from x_i to either T_1 or T_2
        add the loop erasure of P to the tree hit by P
    return T_1 ∪ T_2 ∪ {a, b}
```

Algorithm 3 Compute diag($\mathbf{L}^{\dagger}_{G_r}$) upon edge insertion

1: **function** ApproxUpdateDiag($G_r, r, u, U, t, w, R, B_u$)
2: **Input:** Graph $G_r = G \cup \{a, b\}$, current round r, pivot node u, UST container $U[]$, total # of USTs t, round weights $w[]$, effective resistance estimates $R[]$, BFS Tree B_u
3: **Output:** diag($\widetilde{\mathbf{L}^{\dagger}_{G'}}$)
4: $R_{new}[v] \leftarrow 0\ \forall v \in V \setminus \{u\}$
5: $\omega \leftarrow \mathbf{r}_{G'}(a,b) = \frac{\mathbf{r}_G(a,b)}{1+\mathbf{r}_G(a,b)}$ ▷ computed via $\mathbf{L}^{\dagger}_{G'}[:,a]$ and $\mathbf{L}^{\dagger}_{G'}[:,b]$ (linear systems)
6: **for** $i = 0, \ldots, r-1$ **do**
7: $w[i] \leftarrow (w[i] \cdot (1-\omega))$
8: $U[i]$.resize($\lceil w[i] \cdot t \rceil$) ▷ adjust # of USTs for round i acc. to round weights
9: w.append(ω) ▷ add weight of current round
10: **for** $i \leftarrow 1$ to $\lceil \omega \cdot t \rceil$ **do** ▷ $\lceil \omega \cdot t \rceil$ times
11: $T_i \leftarrow$ Sampling(G_r, a, b) ▷ $\mathcal{O}(m\,\mathrm{diam}(G))$
12: $R_{new} \leftarrow$ Aggregate(T_i, R_{new}, B_u) ▷ $\mathcal{O}(n\,\mathrm{diam}(G))$
13: $U[r]$.append(T_i)
14: $R_{new} \leftarrow R_{new} / \lceil \omega \cdot t \rceil$
15: $R \leftarrow \omega R_{new} + (1-\omega)R$ ▷ Acc. to Lemma 1
16: **for** $v \in V \setminus \{u\}$ **do** ▷ All iterations: $\mathcal{O}(n)$
17: $\widetilde{\mathbf{L}^{\dagger}_{\mathbf{G}'}}[v,v] \leftarrow R[v] - \widetilde{\mathbf{L}^{\dagger}_{\mathbf{G}}}[u,u] + 2\widetilde{\mathbf{L}^{\dagger}_{\mathbf{G}}}[v,u]$
18: **return** diag($\widetilde{\mathbf{L}^{\dagger}_{G'}}$)

Putting the Pieces Together

By applying Eq. (12) to the effective resistance estimates, we obtain Algorithm 3. It obtains an approximation for diag($\mathbf{L}^{\dagger}_{G'}$), where G' is obtained from G by inserting an edge $e = \{a, b\}$. This algorithm is run each time after an edge is added to G. To obtain an initial set of USTs, the algorithm of Angriman et al. (2020) is applied to the original graph G. These USTs are stored in what we call the UST repository, which is used to also store USTs from graphs resulting from a series of edge insertions. All spanning trees together in this repository form a sufficiently large sample of USTs for the graph of the current round. Lines 4 and 5 initialize the vector of new resistance estimates and compute the effective resistance ω of the inserted edge $\{a, b\}$. The latter is necessary to scale the contribution of the USTs from this and previous rounds according to Lemma 1 (Line 15). How many USTs each round contributes is governed by the round weight w; both numbers have to be adapted according to ω (Lines 7 and 8). After sampling and aggregating the new trees as well as updating R (Lines 10 to 15), the new diagonal approximation can be computed and returned.

4.3 *StochJLT

In this section, we propose an improvement to SimplStoch that exploits the following observation: to evaluate the gain function for an arbitrary vertex pair $\{a, b\}$, we only require to compute the squared ℓ_2-norm of two distance vectors: $\mathbf{b}^2_G(a,b) = \left\|\mathbf{L}^{\dagger}(\mathbf{e}_a - \mathbf{e}_b)\right\|^2$ and $\mathbf{r}_G(a,b) = \left\|\mathbf{B}^T\mathbf{L}^{\dagger}(\mathbf{e}_a - \mathbf{e}_b)\right\|^2$ (Eqs. (7–8)). Viewing $\mathbf{b}^2_G(a,b)$ and $\mathbf{r}_G(a,b)$ as pairwise 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 transform (JLT) (Johnson 1984). In this case, pairwise distances among vectors are nearly preserved if we project the vectors onto a low-dimensional subspace, spanned by $\mathcal{O}(\log n/\eta^2)$ random vectors. The JLT lemma, in the improved version by Dasgupta and Gupta (2003), can be stated as:

Lemma 3 *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 Gaussian matrix with entries from* $N(0,1)$ *and* $q > 24 \log n/\eta^2$. *Then with probability at least* $1 - 1/n$

$$(1-\eta)\left\|\mathbf{u_i} - \mathbf{u_j}\right\|^2 \leq \left\|\mathbf{Qu_i} - \mathbf{Qu_j}\right\|^2 \leq (1+\eta)\left\|\mathbf{u_i} - \mathbf{v_j}\right\|^2 \tag{13}$$

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

Using Lemma 3, we can simply project matrices $\mathbf{L}^{\dagger}$ and $\mathbf{BL}^{\dagger}$ 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 $\mathbf{b}_G^2(a,b)$ 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. One can avoid the solution of two sets of Laplacian systems by expressing the effective resistances directly via the projection of (squared) biharmonic distances onto the lower-dimensional space. More precisely, one only solves $\mathbf{LY}=\mathbf{P}^T-\frac{1}{n}\mathbf{1}\mathbf{1}^T\mathbf{P}^T$. Due to $\mathbf{L}^\dagger\cdot\frac{1}{n}\mathbf{1}\mathbf{1}^T=\mathbf{O}$ (the zero matrix), it follows $\mathbf{Y}=\mathbf{L}^\dagger\mathbf{P}^T$, so that we can express effective resistances as follows:

$$\begin{aligned}\left\|\mathbf{QBYP}(\mathbf{e}_a-\mathbf{e}_b)\right\|^2&=(\mathbf{e}_a-\mathbf{e}_b)^T\mathbf{P}^T\mathbf{Y}^T\mathbf{B}^T\mathbf{Q}^T\mathbf{QBYP}(\mathbf{e}_a-\mathbf{e}_b)\\&=(\mathbf{e}_a-\mathbf{e}_b)^T\mathbf{L}^\dagger\mathbf{B}^T\mathbf{BL}^\dagger(\mathbf{e}_a-\mathbf{e}_b)=\left\|\mathbf{BL}^\dagger(\mathbf{e}_a-\mathbf{e}_b)\right\|^2,\end{aligned}\tag{14}$$

where we assume that $\mathbf{Q}$ and $\mathbf{P}$ are orthonormal matrices. Note that there are formulations of the JLT with orthonormal matrices, including very early ones (Johnson 1984; Frankl and Maehara 1988). The formulation in Lemma 3 with random Gaussian entries is only "almost" orthogonal; this condition is usually sufficient in practice (Achlioptas 2003). In our case, this would mean that the equality in Eq. (14) becomes "approximately equal", which would be sufficient for our heuristics as well.

We can integrate the JLT approximation both in the context of ColStoch and SimplStoch (having ColStochJLT and SimplStochJLT, respectively). For both approaches, we set $\eta := 0.55$ in our experiments and thus consider it as a constant in the time complexity statements regarding *StochJLT. 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 $\{\mathbf{BL}^\dagger\}_{a\in\mathcal{S}}$ for G_0, where $s := |\mathcal{S}| = n\sqrt{\frac{1}{k}\cdot\log(\frac{1}{\delta})}$. 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

$$\text{gain}(a,b)\approx\frac{\left\|\mathbf{PL}^\dagger(\mathbf{e}_a-\mathbf{e}_b))^2\right\|}{1+\left\|\mathbf{QBYP}(\mathbf{e}_a-\mathbf{e}_b)^2\right\|}\tag{15}$$

(up to a relative error of $(1+\eta)$). For the update step, we need to sample new projections $\mathbf{P}$ and $\mathbf{Q}$ and recompute the two matrices $\mathbf{PL}^\dagger$ and $\mathbf{QBYP}$. The dominant cost of the approach is due to the main loop, which takes $\mathcal{O}'(n^2\log s)$ time. For SimplStochJLT, the time complexity is $\mathcal{O}'(n^2\log n)$.

4.4 SpecStoch

As the last approach in this section we propose to exploit the spectral expression of the gain function. More precisely, we combine the spectral expressions of effective resistance and (squared) biharmonic distance (Eqs. (8) and (7)) to write Eq. (5) as

$$\text{gain}(a,b)=n\cdot\frac{\sum_{i=2}^n\frac{1}{(\lambda_i)^2}\cdot(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{1+\sum_{i=2}^n\frac{1}{\lambda_i}\cdot(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}.\tag{16}$$

Equation (16) benefits from the fact that both effective resistance and (squared) biharmonic 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. (16) using standard low-rank techniques (Bozzo and Franceschet 2012) and new bounds for both distances. To do so, we exploit the fact that the bulk of the eigenvalues tends to concentrate away from the smallest eigenvalues (Chung and Lu 2004). Moreover, we compute only a small number of eigenpairs on the lower side of the spectrum of $\mathbf{L}$. We expect that the smaller eigenpairs have a larger influence on the sums of Eq. (16): for small i, contributions are accentuated by a large weight, $\frac{1}{\lambda_i^2}$ (recall that we index the eigenvalues ordered non-decreasingly). At the same time, the entries of eigenvector $\mathbf{u_i}$ fluctuate slowly, so we should carefully select $\{a,b\}$ to avoid near-zero contributions. On the other hand, for large i, the eigenvectors $\mathbf{u_i}$ fluctuate rapidly, since they correspond to high-frequency modes of the spectrum (Spielman 2012). Their contribution to Eq. (16) is undermined by $\frac{1}{\lambda_i^2}$ (small for large i). The above observations suggest that for a new edge insertion $\{a,b\}$, the focus should be on eigenpairs corresponding to small i.

We now show how to derive bounds for $\mathbf{b}_G^2(a,b)$. First we break Eq. (8) into partial sums where $c\le n$ is a cut-off value.

$$\begin{aligned}
\mathbf{b}_G^2(a,b) &= \sum_{i=2}^{c} \frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{\lambda_i^2} + \sum_{i=c+1}^{n} \frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{{\lambda_i}^2} \\
&\leq \sum_{i=2}^{c} \frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{{\lambda_i}^2} + \frac{1}{{\lambda_c}^2} \sum_{i=c+1}^{n} (\mathbf{u_i}[a]-\mathbf{u_i}[b])^2 \\
&\leq \sum_{i=2}^{c} \frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{{\lambda_i}^2} + \frac{1}{{\lambda_c}^2}(2 - \sum_{i=2}^{c}(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2) \\
&= \frac{2}{{\lambda_c}^2} + \sum_{i=2}^{c}(\frac{1}{{\lambda_i}^2} - \frac{1}{{\lambda_c}^2})(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2.
\end{aligned} \tag{17}$$

The first inequality holds for large enough eigenvalues (≥ 1), since $\lambda_c \leq \lambda_{c+i}$ and $\frac{1}{(\lambda_c)^2} \geq \frac{1}{(\lambda_{c+i})^2}$ for any i. Moreover, the third line comes from the following observation:

$$\begin{aligned}
&\sum_{i=2}^{n}(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2 = \sum_{i=1}^{n}(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2 \\
&= \sum_{i=1}^{n}\mathbf{u_i}[a]^2 + \sum_{i=1}^{n}\mathbf{u_i}[b]^2 - 2\sum_{i=1}^{n}\mathbf{u_i}[a]\mathbf{u_i}[b] \\
&= \left\|\mathbf{u_a^T}\right\|^2 + \left\|\mathbf{u_b^T}\right\|^2 - 2\mathbf{U}[a,:]\mathbf{U^T}[:,b] = 2
\end{aligned} \tag{18}$$

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

$$\begin{aligned}
\mathbf{b}_G^2(a,b) &= \sum_{i=2}^{c} \frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{\lambda_i^2} + \sum_{i=c+1}^{n} \frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{{\lambda_i}^2} \\
&\geq \sum_{i=2}^{c} \frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{{\lambda_i}^2} + \frac{1}{{\lambda_n}^2} \sum_{i=c+1}^{n} (\mathbf{u_i}[a]-\mathbf{u_i}[b])^2 \\
&\geq \sum_{i=2}^{c} \frac{(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{{\lambda_i}^2} + \frac{1}{{\lambda_n}^2}(2 - \sum_{i=2}^{c}(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2) \\
&= \frac{2}{{\lambda_n}^2} + \sum_{i=2}^{c}(\frac{1}{{\lambda_i}^2} - \frac{1}{{\lambda_n}^2})(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2,
\end{aligned} \tag{19}$$

where the inequality in the third line holds, since $\lambda_n \geq \lambda_{c+i}$ for any i. Following the above, we can easily derive similar bounds for $\mathbf{r}_G(a,b)$. Plugging those bounds together, we can approximate Eq. (16) using the following inequality:

$$\begin{aligned}
&\frac{\frac{2}{{\lambda_c}^2} + \sum_{i=2}^{c}(\frac{1}{{\lambda_i}^2} - \frac{1}{{\lambda_c}^2})(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{1 + \frac{2}{\lambda_n} + \sum_{i=2}^{c}(\frac{1}{\lambda_i} - \frac{1}{\lambda_n})(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2} \leq \text{gain}(a,b) \\
&\leq \frac{\frac{2}{{\lambda_n}^2} + \sum_{i=2}^{c}(\frac{1}{{\lambda_i}^2} - \frac{1}{{\lambda_n}^2})(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}{1 + \frac{2}{\lambda_c} + \sum_{i=2}^{c}(\frac{1}{\lambda_i} - \frac{1}{\lambda_c})(\mathbf{u_i}[a]-\mathbf{u_i}[b])^2}.
\end{aligned} \tag{20}$$

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 (Paige 1980), which often takes only $\mathcal{O}(cm)$ time for sparse matrices (Koch 2011), 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). Assuming "well-behaved" eigenvalues, single function evaluations in Line 10 require only $\mathcal{O}(c)$ time using the bounds in Eq. (20). 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. Under our assumptions, the overall complexity of SpecStoch is $\mathcal{O}'(n^2c + kcm)$ and in case both $c \in \mathcal{O}(1)$ and $k \in \mathcal{O}(1)$, the overall time complexity becomes $\mathcal{O}'(n^2)$.

5 Heuristics for *k*-LRIP

Recall the idea of the k-LRIP problem: consider a fixed *focus node* v. How can the robustness of the graph be improved when we restrict the edges that may be added to the graph to those that are incident to v? This problem is a local variant of k-GRIP in the sense that we can only add edges local to v. Still, we take a global view of the graph and try to improve the total graph resistance with no special consideration for v.

Now assume there is a set F of focus nodes and for each $v \in F$ we want to solve the k-LRIP problem independently. Then, it makes sense to run the preprocessing steps of our algorithms just once and re-use the results when solving k-LRIP for each $v \in F$.

In the following subsections, we will describe how we adapt the heuristics from Sect. 4 to k-LRIP. Let us mention a few general aspects first. Since we still optimize for the total graph resistance, the formulas derived for k-GRIP can generally be re-used; the gain only becomes a function of one (fixed focus) node now. Also the basic structure of Algorithm 1 remains the same in general. Some changes to note: recall from Sect. 2 that the set of all candidates is Ω_v. A candidate edge $e = \{v, b\}$ from this set is uniquely identified by b. That is why a equals v in Lines 9 - 12. Moreover, in Line 9 we sample from $\mathcal{S}$ instead of $\mathcal{S} \times \mathcal{S}$.

Compared to k-GRIP, the candidate set for k-LRIP is considerably smaller. This reduces the number of evaluations in each iteration of the main loop (per focus node). Table 2 shows the time and space complexities of all approaches for k-LRIP. For some heuristics, depending on the density of the graph, the dominant term becomes either the total number of evaluations (second column) or the update step (third column). If m is considerably larger than n, SimplStoch may actually provide the best overall time complexity.

Table 2 Time complexities (assuming standard (pseudo) inversion tools, linear solvers, and eigensolvers used in practice for Laplacians of general graphs) of all approaches involved for k-LRIP for one focus node

	Compute	#Evals × SingleEval	All updates	Memory
StGreedy	$\mathcal{O}(\frac{n^3}{\lvert F\rvert})$	$\mathcal{O}(kn)\times\mathcal{O}(n)$	$\mathcal{O}(kn^2)$	$\mathcal{O}(n^2)$
SimplStoch	$\mathcal{O}(\frac{n^3}{\lvert F\rvert})$	$\mathcal{O}'(n)\times\mathcal{O}(n)$	$\mathcal{O}(kn^2)$	$\mathcal{O}(n^2)$
ColStoch	$\widetilde{\mathcal{O}}(\frac{sm\log n}{\lvert F\rvert})$	$\mathcal{O}'(n)\times\mathcal{O}(n)$	$\widetilde{\mathcal{O}}(ksm\log n)$	$\mathcal{O}((s+\tau)n)+m)$
SimplStochJLT	$\widetilde{\mathcal{O}}(\frac{m\log n}{\lvert F\rvert})$	$\mathcal{O}'(n)\times\mathcal{O}(\log n)$	$\widetilde{\mathcal{O}}(km\log n)$	$\mathcal{O}((s+\log n)n+m)$
ColStochJLT	$\widetilde{\mathcal{O}}(\frac{m\log n\log s}{\lvert F\rvert})$	$\mathcal{O}'(n)\times\mathcal{O}(\log s)$	$\widetilde{\mathcal{O}}(km\log n\log s)$	$\mathcal{O}((\log s+\tau)n+m)$
SpecStoch	$\mathcal{O}(\frac{cm}{\lvert F\rvert})$	$\mathcal{O}'(n)\times\mathcal{O}(c)$	$\mathcal{O}(kcm)$	$\mathcal{O}(cn+m)$

Columns correspond to major steps of Algorithm 1. The $\widetilde{\mathcal{O}}$-notation hides $\log(1/\epsilon)$ factors, where ϵ is the accuracy threshold of the linear solver. The $\mathcal{O}'$-notation hides $\log(1/\delta)$ factors, where δ determines the sample size in the stochastic candidate selection. Note that we consider the JLT parameter η as a constant. The time complexity of the compute step is amortized over all focus nodes F. More details in the text

5.1 SimplStoch

In the case of SimplStoch, preprocessing includes the computation of the full pseudoinverse. When solving k-LRIP for multiple focus nodes, we store a copy of the pseudoinverse before we start the main loop of Algorithm 1. This copy is used to skip the computation of $\mathbf{L}^\dagger$ for the other focus nodes, reducing the time complexity of the Compute step to $\mathcal{O}(\frac{n^3}{|F|})$ per focus node (when amortized over all focus nodes). This approach is also applied to StGreedy.

Regarding sampling, we still want to inspect a subset $\mathcal{S}$ of Ω_v. During each iteration of the main loop, we now sample uniformly at random $s := \frac{n-1-\deg(v)}{k}\log(\frac{1}{\delta})$ vertices (Line 8 in Algorithm 1), resulting in $\mathcal{O}'(n)$ function evaluations overall; they are performed (as in k-GRIP) via Eq. (4) applied to $\mathbf{L}^\dagger$ (obtained during preprocessing). When an edge is added to the graph, $\mathbf{L}^\dagger$ is updated in the same way.

5.2 ColStoch

For ColStoch, $\mathcal{S}$ is sampled from Ω_v as well. The sample size is $s := \frac{n-1-\deg(v)}{k}\log(\frac{1}{\delta})$, which is also the same size as $\mathcal{S}$ in the case of SimplStoch. The concept of sampling only specific vertices (and thus reducing the required number of columns of $\mathbf{L}^\dagger$) that we described for k-GRIP has no significance here, since all edges already have one incident node (and therefore column of $\mathbf{L}^\dagger$) in common. Hence, the sets from which we sample for SimplStoch and ColStoch from k-GRIP are the same when considering a fixed focus node v. The remaining difference is that we are still using graph-based sampling probabilities as described in Sect. 4.2.1 (instead of uniform sampling as in SimplStoch) and do not compute the full pseudoinverse; instead, we solve linear systems for each column of $\mathbf{L}^\dagger$ corresponding to $\mathcal{S}$ again, including the lazy update strategy described for k-GRIP.

The preprocessing in ColStoch consists of (i) setting up a linear solver that computes the required columns of $\mathbf{L}^\dagger$ and (ii) computing the initial sampling probabilities for $\mathcal{S}$, which involves approximating $\mathrm{diag}(\mathbf{L}^\dagger)$. The initial states of both the solver and $\mathrm{diag}(\mathbf{L}^\dagger)$ are stored as a copy and can then be used to setup ColStoch before the main loop instead of re-computing them.

Regarding the running time of the main loop, ColStoch may be slower than SimplStoch due to the additional time for approximating $\mathrm{diag}(\mathbf{L}^\dagger)$. The overall time needs to consider the preprocessing as well—how costly that is with the different methods depends (mostly) on the graph size and its density. We expect ColStoch to provide higher quality results than SimplStoch, though, since we are using graph-based probabilities instead of uniform sampling, as discussed in Sect. 4.2.1.

5.3 *StochJLT

As in the case of k-GRIP, we calculate $\mathbf{r}_G(v,b)$ and $\mathbf{b}^2_G(v,b)$ using the JLT technique. In *StochJLT, preprocessing involves setting up the linear solver and computing the projection with the two matrices $\mathbf{P}$ and $\mathbf{Q}$. Again, results can be stored and used to initialize the solver for the next focus node (with G reset to its original state).

5.4 SpecStoch

As for k-GRIP, the gain function only depends on the spectrum of $\mathbf{L}$. The integration of this approach into Algorithm 1 is similar to k-GRIP: in the compute step, the first c eigenpairs and the largest eigenpair of $\mathbf{L}$ are computed

using iterative solvers, usually taking $\mathcal{O}(cm)$ time. These are then stored for setting up the next focus node. Then, in the main loop, we use the eigenpairs to compute the gain in Eval. When adding an edge to the graph, we compute the eigenpairs again (Line 13) and (as before) bootstrap the new solution process with the previous round to speed up the computation.

Since we are restricted to a fixed focus node in k-LRIP, the search space (and number of calls to Eval) is reduced when compared to k-GRIP. However, for SpecStoch, this has less of an effect on the overall running time than for the other heuristics, since in SpecStoch a single evaluation is rather cheap and the expensive computations are shifted to the Compute and Update steps (where we compute eigenpairs). Hence, we expect that SpecStoch performs worse for k-LRIP than it does for k-GRIP.

6 Experimental results

We conduct experiments to demonstrate the performance of our contributions compared to StGreedy. All algorithms are implemented in C++, using the NetworKit (Staudt et al. 2016) graph APIs. Our test machine for k-GRIP is a shared-memory server with a 2x 18-Core Intel Xeon 6154 CPU and a total of 1.5 TB RAM. For k-LRIP, we use a machine with a Intel Xeon 6126 CPU and 192 GB RAM. To ensure reproducibility, experiments are managed by SimexPal (Angriman et al. 2019). 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 (Leskovec and Krevl 2014) and NR (Rossi and Ahmed 2015), including application-relevant power grid and road networks, see Table 3. In this context, we consider small graphs those whose vertex count is < 10K and medium graphs those whose vertex count is above that but below 57K. The largest graph has around 129K nodes. To evaluate the quality of the solutions, we measure gain improvements: $\mathcal{R}(G) - \mathcal{R}(G_k)$. To this end, after selecting a new edge $\{a, b\}$ for insertion, gain(a, b) is computed via a Laplacian system, for all approaches. This allows us to compare the results of different approaches in fair manner. Our code and the experimental pipeline are available at https://github.com/hu-macsy/2023-kgrip-klrip.

We organize our experimental evaluation in three groups: first, we present experiments for configuring parameters. Second, we evaluate all approaches for k-GRIP in terms of quality and running time. Third, we evaluate all approaches for k-LRIP.

Table 3 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

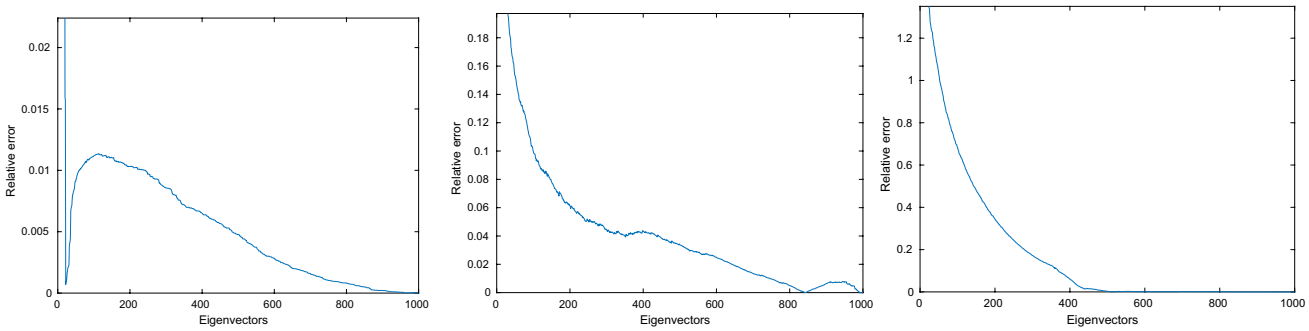
6.1 Configuration experiments

We start by evaluating the performance of SimplStoch for different accuracy values on the small and medium graphs of Table 3. Following the experiments in Mirzasoleiman et al. (2015), we set the accuracy parameter δ to 0.9 and 0.99 (which are reasonable values according to the experiments of Mirzasoleiman et al. (2015) and our own preliminary experiments). In Table 4, we see that there is a clear trade-off between quality and running time, controlled by the accuracy parameter. Still, even for a large δ, 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 δ at 0.9 for small and medium graphs and 0.99 for larger ones.

Additionally, we perform configuration experiments to determine the quality of the gain approximation via Eq. (20) for SpecStoch. To do so, we randomly select a vertex pair and compute Eq. (20) for different numbers 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

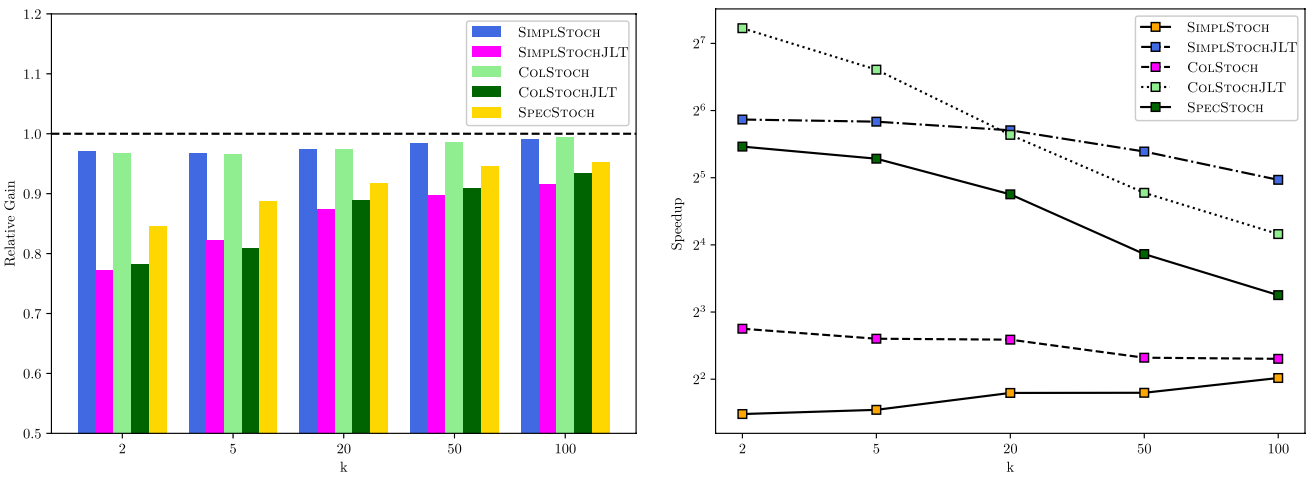
Table 4 Quality and speedup of SimplStoch (relative to StGreedy) for different approximation bounds

SimplStoch	Relative Quality				
	$k = 2$	$k = 5$	$k = 20$	$k = 50$	$k = 100$
$\delta = 0.9$	0.9662	0.9610	0.9696	0.9810	0.9898
$\delta = 0.99$	0.9239	0.9241	0.9442	0.9559	0.9694
SimplStoch	Relative Speedup				
	$k = 2$	$k = 5$	$k = 20$	$k = 50$	$k = 100$
$\delta = 0.9$	2.6	2.6	2.7	2.7	3.1
$\delta = 0.99$	4.0	3.9	4.2	4.1	4.6



(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. (20) for different number of eigenvectors



(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

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 decided to use the sparse LU solver from the Eigen library (Guennebaud et al. 2010) for small and medium graphs and the LAMG solver (Livne and Brandt 2011) 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

Table 5 Runtime results in seconds for k-GRIP for $k = 2$ and $k = 100$ for medium graphs

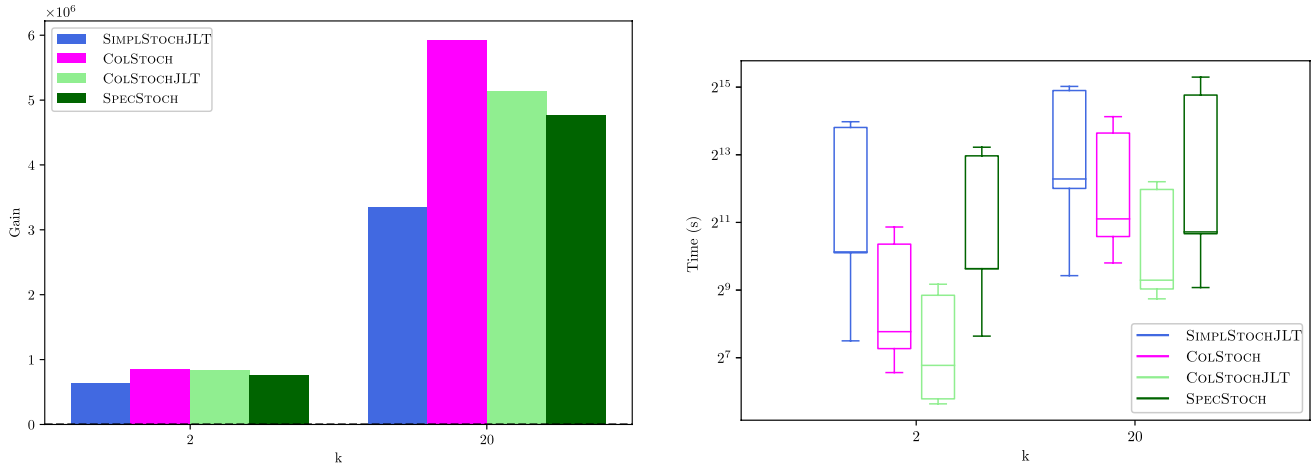
k=2						
Algorithm	SIMPL STOCH	SIMPL JLT	COL STOCH	COL JLT	SPEC STOCH	STGREEDY
inf-power	50.0	**1.6**	10.1	3.2	4.0	118.3
facebook-ego-combined	18.6	1.7	5.8	**0.7**	4.1	46.0
web-spam	29.2	3.4	17.8	**1.8**	5.6	68.5
Wiki-Vote	110.1	9.0	65.7	**5.4**	12.7	357.6
p2p-Gnutella09	137.3	13.0	94.6	**8.1**	16.3	296.0
p2p-Gnutella04	452.5	38.2	297.0	**28.1**	40.1	1163.2
web-indochina	489.6	10.6	95.9	**1.9**	15.2	1700.3
ca-HepPh	479.2	24.1	261.1	**15.1**	31.8	1312.5
web-webbase-2001	1634.4	20.1	292.5	**2.4**	25.8	6402.5
arxiv-astro-ph	1696.5	166.4	1426.0	**135.4**	165.4	5628.3
as-caida20071105	6664.3	93.7	1434.5	**8.0**	88.9	17544.7
cit-HepTh	4956.7	893.4	6973.9	**816.0**	871.2	13818.5
ia-email-EU	11719.3	108.5	2491.7	**5.2**	101.8	32679.4
k=100						
Algorithm	SIMPL STOCH	SIMPL JLT	COL STOCH	COL JLT	SPEC STOCH	STGREEDY
inf-power	41.5	**3.4**	125.3	140.1	63.6	594.6
facebook-ego-combined	19.9	**14.3**	19.1	23.5	56.6	139.3
web-spam	31.0	**18.3**	50.8	32.5	80.2	79.3
Wiki-Vote	121.4	**45.1**	122.1	50.4	133.0	428.9
p2p-Gnutella09	141.3	**51.5**	173.7	54.7	114.5	314.3
p2p-Gnutella04	448.2	**129.3**	580.5	133.7	164.6	1298.2
web-indochina	512.6	**18.3**	137.8	73.6	161.1	3524.8
ca-HepPh	498.4	**88.6**	439.2	136.9	245.1	1499.1
web-webbase-2001	1520.0	**23.9**	295.4	74.5	209.9	14802.7
arxiv-astro-ph	1730.6	469.1	2649.4	523.3	**467.6**	7711.7
as-caida20071105	7712.0	130.2	1630.8	**113.1**	475.7	18350.9
cit-HepTh	4932.2	1960.1	13094.1	2122.5	**1544.7**	11253.3
ia-email-EU	11820.4	136.0	3000.7	**65.7**	428.6	32771.2

For each instance the fastest solver is emphasized in bold

(required by SPECSTOCH), we use the Slepc library (Hernandez et al. 2005).

6.2 Results for *k*-GRIP

We first compare our approaches on the small and medium graphs of Table 3, 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. 2a). Moreover, SPECSTOCH, SIMPLSTOCHJLT and COLSTOCHJLT are away by 9%, 14% and 15%, respectively. Regarding running time, the JLT-based approaches are the fastest, being on average 48× (SIMPLSTOCHJLT) and 68× (COLSTOCHJLT) faster than STGREEDY (Fig. 2b). 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 (mostly) independent of k. Overall, SPECSTOCH may be the best approach for medium graphs in a wide variety of applications since it produces good quality results and is on average 26× faster than STGREEDY. Detailed runtime results are available in Table 5. A disadvantage of SPECSTOCH is that the running



(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

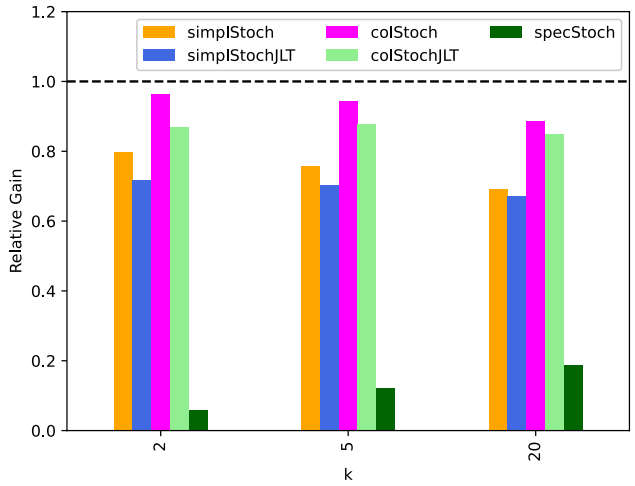


Fig. 4 Aggregated quality results (using geometric mean) of k-LRIP on small and medium graphs ($n < 57K$) for different k. Results are relative to StGreedy

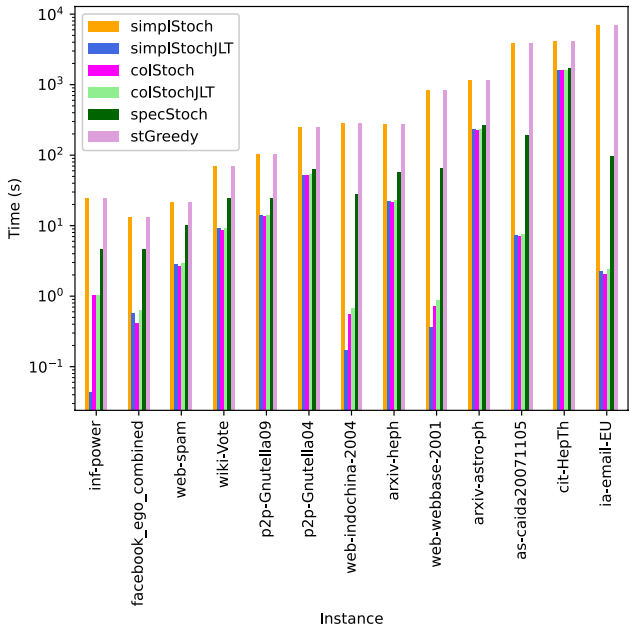


Fig. 5 Preprocessing times for different graphs, taking the arithmetic mean over all k. See Table 3 for size information

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 3. For this experiment, we report absolute values since we do not have a clear reference. With a time limit of 12 hours, StGreedy always times out. These results show that a cubic approach such as StGreedy becomes impractical once the number of nodes in the graph exceeds a certain threshold (such as 57K in our tests). The best approaches for large graphs are ColStochJLT and ColStoch. Both of them produce the highest quality results, with ColStoch slightly ahead. ColStochJLT is the fastest approach, requiring on average 2 [resp. 20] minutes for $k = 2$ [resp. $k = 20$]. 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.

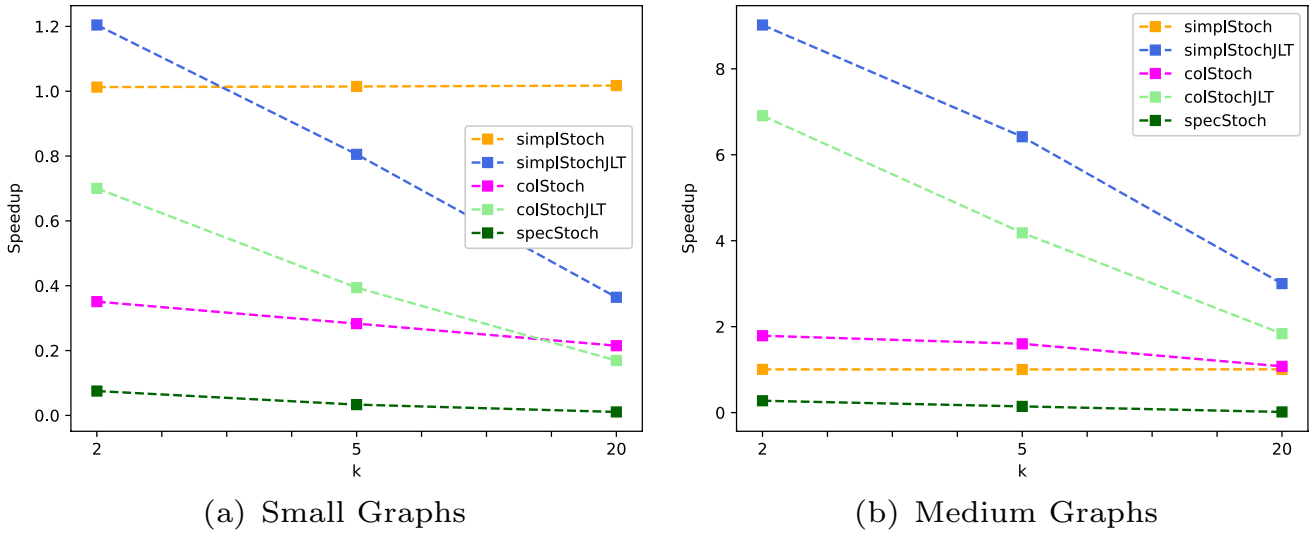


Fig. 6 Aggregated speedup results of k-LRIP on small and medium graphs for different k. Results are relative to StGreedy

6.3 Results for k-LRIP

For k-LRIP, we use the same parameter settings determined in Sect. 6.1 and choose 25 focus nodes at random to run our algorithms on, with a 6-hour time limit for each experiment. We evaluate $k \in \{2, 5, 20\}$, which means that we have to compute up to $20 \cdot 25 = 500$ Update steps overall—up to 5× more than for k-GRIP. At the same time, the number of Eval computations is reduced as described in Sect. 5. (One could of course increase k and decrease the number of focus nodes at the same time and reach about the same number of Update and Eval calls.)

Quality and speedup shown in this section are the geometric mean of the results for all focus nodes (in relation to the baseline StGreedy). Absolute running times are aggregated using the arithmetic mean. When comparing the running time of k-LRIP, we compute the running time for a focus node by taking the actual execution time of the main loop of Algorithm 1 for that focus node and add to this $\frac{1}{|F|}$ ($\frac{1}{25}$ in our case) of the preprocessing time, such that the preprocessing time is amortized over all focus nodes.

For the evaluation, we first compare the solution quality for the small and medium graphs of Table 3, see Fig. 4. Here, ColStoch produces the best results, followed by ColStochJLT. Depending on k, ColStoch produces results that are on average 4%–12% away from StGreedy. The SimplStoch* results are 20%–30% away from StGreedy, showing that our graph-based sampling technique applied in ColStoch does improve the quality of the solution. SpecStoch appears to be not competitive.

Next, we take a look at the preprocessing time for our approaches (Fig. 5). As expected, we can see a clear difference between the approaches that compute the full pseudoinverse (stGreedy and simplStoch) and those that set up a linear solver. The preprocessing time for the solver-based heuristics depends on the density of the graph. A good example of this observation is the difference in preprocessing time for the two instances *web-indochina-2004* and *arxiv-heph*. Both graphs have about the same number of nodes, but *arxiv-heph* contains about 2.5 times more edges, which increases the preprocessing time for ColStoch, ColStochJLT and SimplStochJLT by an order of magnitude. Still, the solver setup is considerably faster than calculating $\mathbf{L}^\dagger$, being up to three orders of magnitude faster for larger, sparse graphs. Generally, the preprocessing of the *JLT variants is slightly slower than without JLT, since we have to set up the projection as well. Computing the eigenpairs for SpecStoch is faster than calculating $\mathbf{L}^\dagger$, but slower than setting up linear solvers. One should keep in mind, though, that for SpecStoch this preprocessing computation is mostly the time to calculate the eigenpairs, which is the same computation required for the edge insertion update for SpecStoch.

Finally, we compare the running time of our approaches. We split the speedup results into two figures for small and medium graphs, respectively (Fig. 6). For both cases, SpecStoch has an average speedup of less than one. This is due to the large number of eigenpair computations required, which are slow, as we have seen in preprocessing. For this reason, most experiments with medium graphs and $k = 20$ did not finish for SpecStoch.

Regarding the other heuristics: for small graphs, SimplStoch is the fastest algorithm, with an average speedup of 1.01. The other algorithms are slower than StGreedy, because computing $\mathbf{L}^\dagger$ for a small graph is still fast enough in practice and the update step generally is fast as well.

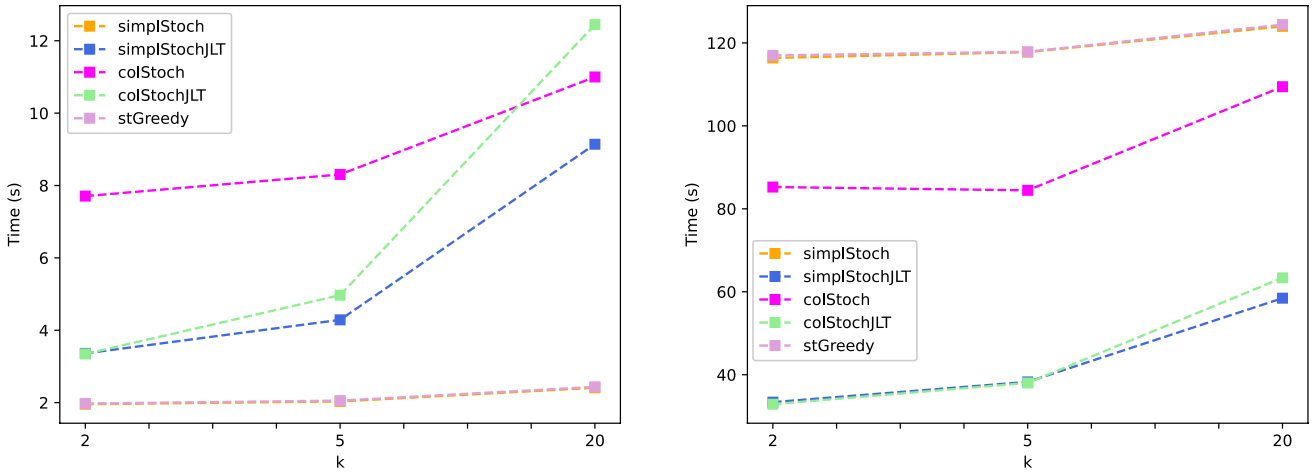


Fig. 7 Aggregated running time results of k-LRIP on small and medium graphs for different k. Results for SpecStoch are orders of magnitude larger and not shown here for readability

Considering that all approaches finish in at most 12 seconds (Fig. 7a), StGreedy is fast enough, so that these small graphs do not require (and do not benefit from) more complicated heuristics.

For the medium graphs, SimplStochJLT is the fastest approach with a speedup of up to 9× for $k = 2$. This is to be expected, since the JLT strategy generally reduces computation time. The second fastest solution is ColStochJLT, which is explained by the additional time required to approximate $\text{diag}(\mathbf{L}^\dagger)$. ColStoch is still faster than SimplStoch for small k, but for $k = 20$ both are almost equal. We also notice that for the *cit-HepTh* graph, which is considerably denser than all other graphs ($m = 352$K), the solver-based heuristics (simplStochJLT, colStochJLT and colStoch) time out, while the $\mathbf{L}^\dagger$-based heuristics do not. The reason for this is that the time complexity of the solver update step depends on m.

Even though the preprocessing itself is more expensive for SimplStoch, once $\mathbf{L}^\dagger$ is computed, the update step is considerably cheaper than in the case of linear solvers and as such SimplStoch is competitive for larger $k \cdot |F|$, where there are many updates, as long as computing $\mathbf{L}^\dagger$ is feasible. Of course, for large enough graphs, one cannot compute $\mathbf{L}^\dagger$ in reasonable time as we have seen for the large graphs in k-GRIP.

Overall, based on these results the choice of the *best* heuristic depends on k, $|F|$, and the density of the graph. In general, there is a trade-off between running time and quality. For the fastest solution, one should choose SimplStochJLT. When quality is the larger concern, ColStoch provides good results. With ColStochJLT, there is also an option in the middle, providing good quality and time.

7 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 already 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, offers an alternative (on average 28× faster than greedy). For larger graphs and whenever high quality is desirable, 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.

Similar results can be observed for the related k-LRIP problem. Some differences occur, though: for small graphs (roughly 10,000 nodes or less), StGreedy is fast enough since the running time and space consumption of the pseudoinversion are still tolerable and can be amortized over the numerous focus nodes. When the graphs become larger, our new heuristics pay off for k-LRIP as well—except SpecStoch, which is dominated in terms of quality *and* running time.

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.

Acknowledgements We are grateful for coding support in early development stages by HU Berlin student Matthias Görg. Under the supervision of MP and HM, he also developed important ideas for the $\mathrm{diag}(\mathbf{L}^{\dagger})$ update strategy. Moreover, we thank Massimo Achterberg from Delft University of Technology for helpful discussions on several aspects of the paper.

Author contribution HM and MP designed the k-GRIP part of the research project. MP devised and implemented the k-GRIP heuristics, carried out the corresponding experiments, and wrote the k-GRIP part of the manuscript. HM, RK, and LB designed the k-LRIP part of the research project. LB adapted and implemented the heuristics for k-LRIP, carried out the corresponding experiments, and wrote the k-LRIP part of the manuscript. HM and RK wrote non-technical parts of the manuscript and edited all the other parts.

Declarations

Conflict of interest Competing interests The authors declare no competing interests.

References

Achlioptas D (2003) Database-friendly random projections: Johnson-Lindenstrauss with binary coins. J Comput Syst Sci 66(4):671–687

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

Angriman E, Predari M, van der Grinten A, Meyerhenke H (2020) Approximation of the diagonal of a Laplacian's pseudoinverse for complex network analysis. In: ESA 2020, Italy, vol 173, pp 6:1–6:24

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

Avena L, Castell F, Gaudillière A, Mélot C (2018) Random forests and networks analysis. J Stat Phys 173:985–1027

Avrachenkov K, Litvak N (2006) The effect of new links on google pagerank. Stoch Model 22(2):319–331

Barabási A-L, Pósfai M (2016) Network science. Cambridge University Press, Cambridge

Baras JS, Hovareshti P (2009) Efficient and robust communication topologies for distributed decision making in networked systems. In: Proceedings of the 48h IEEE conference on decision and control (CDC) held jointly with 2009 28th Chinese control conference, pp 3751–3756

Bergamini E, Crescenzi P, D'angelo G, Meyerhenke H, Severini L, Velaj Y (2018) Improving the betweenness centrality of a node by adding links. ACM J Exp Algorithmics 23:1–32

Bollobás B (1998) Modern graph theory. Graduate texts in mathematics, corrected. Springer, Heidelberg

Bozzo E, Franceschet M (2012) Effective and efficient approximations of the generalized inverse of the graph Laplacian matrix with an application to current-flow betweenness centrality. arXiv:1205.4894

Brandes U, Fleischer D (2005) Centrality measures based on current flow. In: STACS. Springer, Berlin, pp 533–544

Cats O, Koppenol G-J, Warnier M (2017) Robustness assessment of link capacity reduction for complex networks: application for public transport systems. Reliab Eng Syst Saf 167:544–553

Cetinay H, Mas-Machuca C, Marzo JL, Kooij R, Van Mieghem P (2020) Comparing destructive strategies for attacking networks. Springer, Cham, pp 117–140

Chung F, Lu L (2004) Complex graphs and networks. American Mathematical Society, Providence

Clemente GP, Cornaro A (2020) Bounding robustness in complex networks under topological changes through majorization techniques. Eur Phys J B 93(114):1–12

Crescenzi P, D'angelo G, Severini L, Velaj Y (2016) Greedily improving our own closeness centrality in a network. ACM Trans Knowl Discov Data (TKDD) 11(1):1–32

Dasgupta S, Gupta A (2003) An elementary proof of a theorem of Johnson and Lindenstrauss. Random Struct Algorithms 22(1):60–65

Demaine ED, Zadimoghaddam M (2010) Minimizing the diameter of a network using shortcut edges. In: Algorithm theory-SWAT 2010: 12th Scandinavian symposium and workshops on algorithm theory, Bergen, Norway, June 21–23, 2010. Proceedings 12. Springer, pp 420–431

Ellens W, Spieksma F, Van Mieghem P, Jamakovic A, Kooij R (2011) Effective graph resistance. Linear Algebra Appl 435(10):2491–2506

Fiedler M (1973) Algebraic connectivity of graphs. Czechoslov Math J 23:298–305

Frankl P, Maehara H (1988) The Johnson-Lindenstrauss lemma and the sphericity of some graphs. J Combin Theory Ser B 44(3):355–362

Freitas S, Yang D, Kumar S, Tong H, Chau DH (2022) Graph vulnerability and robustness: a survey. IEEE Trans Knowl Data Eng

Ghosh A, Boyd S, Saberi A (2008) Minimizing effective resistance of a graph. SIAM Rev 50(1):37–66

Guennebaud G, Jacob B et al (2010) Eigen v3. http://eigen.tuxfamily.org

Gutman I, Xiao W (2004) Generalized inverse of the Laplacian matrix and some applications. Bulletin Classe Des Sciences Mathematiques et Naturalles 129:15–23

Hassidim A, Singer Y (2017) Robust guarantees of stochastic greedy algorithms. In: Proc of the 34th intl conference on machine learning, vol 70. PMLR, pp 1424–1432

He Z (2020) Performance of complex networks. PhD thesis. Delft University of Technology

Hernandez V, Roman JE, Vidal V (2005) SLEPc: a scalable and flexible toolkit for the solution of eigenvalue problems. ACM Trans Math Softw 31(3):351–362

Johnson WB (1984) Extensions of Lipschitz mappings into Hilbert space. Contemp Math 26:189–206

Jun W, Barahona M, Yue-Jin T, Hong-Zhong D (2010) Natural connectivity of complex networks. Chin Phys Lett 27(7):078902

Koç Y, Warnier M, Van Mieghem P, Kooij RE, Brazier FM (2014) A topological investigation of phase transitions of cascading failures in power grids. Physica A 415:273–284

Koch E (2011) The Lanczos method. In: Pavarini E, Vollhardt D, Koch E, Lichtenstein A (eds) The LDA+DMFT approach to strongly correlated materials. Forschungszentrum Jülich, Jülich

Kooij RE, Achterberg MA (2023) Minimizing the effective graph resistance by adding links is NP-hard. arXiv:2302.12628

Leskovec J, Krevl A (2014) Stanford large network dataset collection. SNAP Datasets. http://snap.stanford.edu/data

Li G, Hao ZF, Huang H, Wei H (2018) Maximizing algebraic connectivity via minimum degree and maximum distance. IEEE Access 6:41249–41255

Livne O, Brandt A (2011) Lean algebraic multigrid (LAMG): fast graph Laplacian linear solver. SIAM J Sci Comput 34

Lovász LM (1996) Random walks on graphs: a survey. Combinatorica 1–46

Manghiuc B, Peng P, Sun H (2020) Augmenting the algebraic connectivity of graphs. In: 28th European Symp. on Algorithms, ESA, volume 173 of LIPIcs. Schloss Dagstuhl, pp 70:1–70:22

Mavroforakis C, Garcia-Lebron R, Koutis I, Terzi E (2015) Spanning edge centrality: large-scale computation and applications. In: Proc. of the 24th Intl. Conference on World Wide Web. Intl. World Wide Web Conferences Steering Committee, pp 732–742

Minoux M (1978) Accelerated greedy algorithms for maximizing submodular set functions. In: Stoer J (ed) Optimization techniques. Springer, Berlin, pp 234–243

Minoux M (1989) Networks synthesis and optimum network design problems: models, solution methods and applications. Networks 19(3):313–360

Mirzasoleiman B, Badanidiyuru A, Karbasi A, Vondrák J, Krause A (2015) Lazier than lazy greedy. In: Proceedings of the 29th AAAI conference on artificial intelligence, AAAI'15. AAAI Press, pp 1812–1818

Mosk-Aoyama D (2008) Maximum algebraic connectivity augmentation is NP-hard. Oper Res Lett 36(6):677–679

Newman M (2018) Networks, 2nd edn. Oxford University Press, Oxford

Olfati-Saber R, Fax JA, Murray RM (2007) Consensus and cooperation in networked multi-agent systems. Proc IEEE 95(1):215–233

Olsen M, Viglas A (2014) On the approximability of the link building problem. Theoret Comput Sci 518:96–116

Paige C (1980) Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem. Linear Algebra Appl 34:235–258

Papagelis M (2015) Refining social graph connectivity via shortcut edge addition. ACM Trans Knowl Discov Data. https://doi.org/10.1145/2757281

Perera S, Bell MGH, Bliemer MCJ (2015) Modelling supply chains as complex networks for investigating resilience: an improved methodological framework

Perumal S, Basu P, Guan Z (2013) Minimizing eccentricity in composite networks via constrained edge additions. In: MILCOM 2013-2013 IEEE military communications conference. IEEE, pp 1894–1899

Pizzuti C, Socievole A (2018) A genetic algorithm for enhancing the robustness of complex networks through link protection. In: International conference on complex networks and their applications. Springer, pp 807–819

Pizzuti C, Socievole A (2023) Incremental computation of effective graph resistance for improving robustness of complex networks: a comparative study. In: Cherifi H, Mantegna RN, Rocha LM, Cherifi C, Micciche S (eds) Complex networks and their applications XI. Springer, Cham, pp 419–431

Predari M, Kooij R, Meyerhenke H (2022) Faster greedy optimization of resistance-based graph robustness. In: 2022 IEEE/ACM international conference on advances in social networks analysis and mining (ASONAM), Los Alamitos, CA, USA, Nov 2022. IEEE Computer Society, pp 1–8

Ranjan G, Zhang Z, Boley D (2014) Incremental computation of pseudo-inverse of Laplacian and applications—8th international conference, COCOA 2014, Wailea, Maui, HI, USA, December 19–21, 2014, Proceedings. Lecture notes in computer science. In: Zhang Z, Wu L, Xu W, Du D (eds) Combinatorial optimization, vol 8881. Springer, pp 729–749

Rossi RA, Ahmed NK (2015) The network data repository with interactive graph analytics and visualization. In: Proceedings of the 29th AAAI conference on artificial intelligence, AAAI'15. AAAI Press, pp 4292–4293

Rueda DF, Calle E, Marzo JL (2017) Robustness comparison of 15 real telecommunication networks: structural and centrality measurements. J Netw Syst Manag 25(2):269–289

Schneider CM, Moreira AA, Andrade JS, Havlin S, Herrmann HJ (2011) Mitigation of malicious attacks on networks. Proc Natl Acad Sci 108(10):3838–3841

Shan L, Yi Y, Zhang Z (2018) Improving information centrality of a node in complex networks by adding edges. In: Proceedings of the 27th international joint conference on artificial intelligence, IJCAI-18, pp 3535–3541. International joint conferences on artificial intelligence organization, 7. https://www.ijcai.org/proceedings/2018/0491.pdf

Sherman J, Morrison WJ (1950) Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. Ann Math Stat 21(1):124–127

Spielman D (2012) Spectral graph theory. In: Combinatorial scientific computing, vol 18. CRC Press, Boca Raton, p 18

Staudt CL, Sazonovs A, Meyerhenke H (2016) NetworKit: a tool suite for large-scale complex network analysis. Netw Sci 4(4):508–530

Summers T, Shames I, Lygeros J, Dörfler F (2015) Topology design for optimal network coherence. In: 2015 European control conference (ECC). IEEE, pp 575–580

Summers T, Shames I, Lygeros J, Dorfler F (2017) Correction to "Topology design for optimal network coherence". https://personal.utdallas.edu/~ths150130/papers/ECC_Correction.pdf

Summers TH, Kamgarpour M (2019) Performance guarantees for greedy maximization of non-submodular controllability metrics. In : 17th European Control Conf., ECC. IEEE, pp 2796–2801

Van Mieghem P, Devriendt K, Cetinay H (2017) Pseudoinverse of the Laplacian and best spreader node in a network. Phys Rev E 96:032311

Van Mieghem P, Stevanović D, Kuipers F, Li C, van de Bovenkamp R, Liu D, Wang H (2011) Decreasing the spectral radius of a graph by link removals. Phys Rev E 84:016101

Wang H, Van Mieghem P (2008) Algebraic connectivity optimization via link addition. In: Proceedings of the 3rd international conference on bio-inspired models of network, information and computing systems, BIONETICS'08, Brussels, BEL, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). https://www.nas.ewi.tudelft.nl/people/Huijuan/Huijuan_paper/Bionetics2008_AlgebraicConnectivity.pdf

Wang X, Pournaras E, Kooij RE, Mieghem PV (2014) Improving robustness of complex networks via the effective graph resistance. Eur Phys J B 87:1–12

Wei Y, Li R-h, Yang W (2021) Biharmonic distance of graphs. arXiv preprint arXiv:2110.02656

Wilson DB (1996) Generating random spanning trees more quickly than the cover time. In *Proc of the 28th annual ACM symposium on theory of computing*, STOC'96. Association for Computing Machinery, pp 296–303

Yazdani A, Jeffrey P (2011) Complex network analysis of water distribution systems. Chaos (Woodbury N.Y), 21:016111

Yi Y, Shan L, Li H, Zhang Z (2018) Biharmonic distance related centrality for edges in weighted networks. In: IJCAI, pp 3620–3626