OPTIMIZING NETWORK ROBUSTNESS *VIA* KRYLOV SUBSPACES

STEFANO MASSEI[1,*] AND FRANCESCO TUDISCO[2,3]

Abstract. We consider the problem of attaining either the maximal increase or reduction of the robustness of a complex network by means of a bounded modification of a subset of the edge weights. We propose two novel strategies combining Krylov subspace approximations with a greedy scheme and an interior point method employing either the Hessian or its approximation computed *via* the limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm (L-BFGS). The paper discusses the computational and modeling aspects of our methodology and illustrates the various optimization problems on networks that can be addressed within the proposed framework. Finally, in the numerical experiments we compare the performances of our algorithms with state-of-the-art techniques on synthetic and real-world networks.

1. INTRODUCTION

When studying and analyzing a complex network, one of the main questions is how to identify important nodes and robust connections among them, given the network topology and no other external data. There is a broad literature on the subject, with many different models and associated algorithms. As a network can be naturally represented by a matrix, many successful approaches strongly rely on tools from linear algebra and matrix analysis [12, 25].

Spectral models, such as eigenvector centrality [54], PageRank [33], or resistance distance [40], are based on the eigenvalues and eigenvectors of graph matrices and rely on a mutually reinforcing argument, while path-based models, such as Katz centrality [55], subgraph centrality and total communicability [22], use the entries of suitable graph matrix functions and are based on weighted walk counts.

For example, if $\boldsymbol{x} \geq 0$ is the Perron eigenvector of the adjacency matrix A of an undirected graph $G = (V, E)$, then $A\boldsymbol{x} = \lambda\boldsymbol{x}$ with $\lambda > 0$ and hence x_i is proportional to $\sum_j A_{ij}x_j > 0$, for all the nodes $i \in V$. Thus, we can interpret the entries of $\boldsymbol{x}$ as an importance score for the nodes of G, known as *Bonacich centrality* or *eigenvector centrality* [54], where x_i, the importance of node i, is mutually reinforced by the importance of its neighbors.

[1] Department of Mathematics, University of Pisa, Pisa, Italy.
[2] School of Mathematics & Maxwell Institute, University of Edinburgh, Edinburgh, UK.
[3] Gran Sasso Science Institute, L'Aquila, Italy.
*Corresponding author: stefano.massei@unipi.it

Similarly, if we are given a function of the adjacency matrix

$$f(A) = a_0 I + a_1 A + a_2 A^2 + a_3 A^3 + \dots, \tag{1}$$

where the coefficients a_k are nonnegative, we can interpret the diagonal entries of $f(A)$ as node importances. In fact, in a weighted graph G, the weight of a walk from i to j of length k can be defined as $A_{u_0 u_1} A_{u_1 u_2} \dots A_{u_{k-1} u_k} > 0$, where all pairs $u_i u_{i+1}$ are edges and $u_0 = i$, $u_k = j$. Thus, the sum of the weights of all the walks of length k from i to j corresponds to $(A^k)_{ij}$ and the diagonal entry $f(A)_{ii}$ defines the so-called f–*centrality* or *subgraph centrality* score of the node i [24], which corresponds to the weighted sum of all the walks of any length from i and returning to i, *i.e.* the subgraphs containing i. Related to the individual centrality of a node are important notions of network robustness and network connectivity, which can be quantified by the summations $\sum_i x_i$, $\sum_i f(A)_{ii}$ and $\sum_{ij} f(A)_{ij}$, respectively (see *e.g.* [10,20]). These quantities measure the degree of resiliency of a network in the face of accidental failures or deliberate attacks, modeled as edge modification, removal, or insertion. Both spectral and matrix function-based centrality measures have been successfully used in a variety of settings, including discovering relevant proteins in protein-protein interaction networks [19], as well as keystone species in ecological food webs and landscapes [21].

While spectral centralities require the evaluation of one extremal eigenvector and can thus be computed in a relatively cheap way by means of standard sparse numerical eigensolvers, computing the entries of a matrix function can be in general a much more expensive operation, in particular when the matrix is large. This numerical challenge has prompted extensive research work in recent years. Based on Krylov subspace techniques as well as Gauss-Lobatto quadrature formulas, a variety of efficient numerical techniques have been proposed for large-scale sparse networks [2,6,26,27,36,47].

Rather than the problem of their efficient evaluation, in this work we focus on the problem of the optimization of matrix function-based node centrality scores. Roughly, we look for a "small" modification $A+X$ of the current network A that yields the largest centrality increase. Here small means that only a limited number of nonzero entries are allowed in X or, in other terms, that we are allowed to modify only a limited number of edges of the graph. Clearly, the resulting optimization task is more complicated than the centrality evaluation problem, as already simple first-order optimization methods would require evaluating both $f(A + X)$ and its Fréchet derivative for many different choices of X. Based on recent work on low-rank updates of matrix functions and trace estimators [5,15], we propose two strategies based on the efficient approximation of $f(A+X) - f(A)$ and the Fréchet derivative of $f(A)$ along multiple directions, to optimize the robustness measure $\mathrm{Tr}(f(A + X)) := \sum_i f(A + X)_{ii}$, for both the combinatorial (unweighted) case, in which both A and $A + X$ are binary matrices, and the continuous (weighted) case, in which edge weight tuning is allowed. Among the most frequently used functions f we mention the exponential function $f(z) = \exp(z)$, which corresponds to the so-called *natural connectivity* [22]; the hyperbolic sine and consine functions $f(z) = \sinh(z)$, $f(z) = \cosh(z)$, which are often used as a measure of bipartitedness and to define so-called *returnability* [23]; the resolvent function $f(z) = (1-\alpha z)^{-1}$, which defines the so-called *Katz centrality* [24].

The remainder of the paper is structured as follows. In Section 2 we introduce the optimization problems that we are going to analyze. Section 3 describes the greedy algorithm that we propose in the context of unweighted binary graphs and other techniques that will be used for comparison, see Section 3.4. Section 4 is dedicated to the gradient method that we propose for weighted graphs. Finally, Section 5 reports numerical experiments concerning optimization problems on both weighted and unweighted graphs.

1.1. Related work

Optimizing network robustness or network connectivity is in general very challenging, due to the combinatorial nature of the problem. A large body of work has focused on spectral-based scores. The problem of minimizing the largest eigenvalue (spectral radius) of A by a small number of edge and node removals is considered in [48,53,57]. This is shown to be an NP-hard problem which is addressed by a number of heuristics in [48,53] or *via* a semidefinite program with polynomial time complexity in [57]. A similar problem is considered in [39,51],

with the aim of optimizing the network diffusion rate. The works [31, 56] studied the problem of maximizing the algebraic connectivity, *i.e.*, the second smallest eigenvalue of the graph Laplacian, and propose both a convex relaxation-based method and a greedy perturbation heuristic, based on the entries of the Fiedler eigenvector of the initial network. In [9] the problem of modifying network edges to reduce external influence is studied. This is done by controlling the asymptotic consensus value $\boldsymbol{x}^T \boldsymbol{a}$, where $\boldsymbol{x}$ is the eigenvector centrality, *i.e.* the Perron eigenvector of A, and $\boldsymbol{a}$ is a vector of external user consensus coefficients. The eigenvector centrality $\boldsymbol{x}$ is also the subject of [45], where it is observed that, often, modifying a very small subset of edges of a real-world network is enough to drastically change and thus control the eigenvector centrality value of any node in the network. Instead, the Perron eigenvector of the PageRank matrix, so–called PageRank or random walk centrality, is the subject of [30].

Alongside spectral-based coefficients, other network scores have been considered by several authors. For example, [43] deals with the problem of improving both coverage and betweenness centralities by adding a small set of edges to the network. Greedy algorithms for improving coverage and closeness centralities are proposed in [17] and [16], respectively.

Centrality optimization problems for indices defined by means of matrix functions are considered for instance in [3, 10, 32]. These works target the optimization of a number of robustness and connectivity coefficients of the network, by modifying, adding, or removing a small subset of edges. In [32], a semidefinite program-based approach is proposed for the optimization of the total effective resistance, defined as $\text{Tr}(L^+) = \sum_i (L^+)_{ii}$, where L^+ is the pseudo inverse of the graph Laplacian L. In [3, 10], instead, given a suitable function f, a number of heuristics are proposed to efficiently enhance the network natural connectivity, defined as $f^{-1}(\text{Tr}(f(A))/n)$, and the network total communicability $\boldsymbol{1}^T f(A)\boldsymbol{1} = \sum_{ij} f(A)_{ij}$, respectively. Both these two studies show that very good results can be achieved by modifying edges between nodes with high or low centrality values. The recent work [18] proposes to measure the sensitivity of the network communicability, to the addition or removal of certain edges, by looking at the derivatives of $\boldsymbol{1}^T f(A)\boldsymbol{1}$. The latter quantities, called *total network sensitivities*, are defined in terms of evaluations of the Fréchet derivative of $f(A)$. The preprint by Schweitzer [49], which appeared in parallel to the first version of this document, introduces an efficient technique that is able to compute all the total network sensitivities by means of a single evaluation of the Frechét derivative of $f(A)$ in the rank one direction $\boldsymbol{1}\boldsymbol{1}^T$. An analogous technology is applicable for computing the derivatives of the network's natural connectivity.

Building on top of this body of work, we focus here on the optimal modification of the network's natural f-connectivity. In the sequel, we formalize the problem and the algorithms we propose.

2. Optimizing the natural connectivity

Networks strongly rely on their robustness, *i.e.*, the ability to maintain a high degree of connectivity when a portion of the network's structure is damaged or simply altered. An intuitive notion of graph robustness can be expressed in terms of the redundancy of routes between vertices. If we consider a source vertex and a termination vertex, there may be several paths between them. When one path fails, the two vertices can still communicate *via* other alternative routes. Hence, the robustness of the network grows with the number of available alternative routes.Thus, an ideal measure of robustness for a network would be the degree of redundancy of alternative paths, *i.e.* the number of alternative routes of different lengths for all pairs of vertices. However, this number is very difficult to compute.

An alternative definition of robustness, which is usually called "natural connectivity", counts instead the number V of closed walks of any length. Let $G = (V, E)$ be an undirected, possibly weighted graph with $V = \{1, \ldots, n\}$ and entry wise nonnegative symmetric adjacency matrix $A \geq 0$, such that $A_{ij} > 0$ if and only if $ij \in E$. As the number of closed walks of length k from i to itself coincides with the i-th diagonal entry of the

k-th power of the adjacency matrix, we can quantify the natural connectivity by looking at

$$\ln\left(\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{\infty}\frac{(A^k)_{ii}}{k!}\right) = \ln\left(\frac{1}{n}\mathrm{Tr}(\exp(A))\right) = \ln\left(\frac{1}{n}\sum_{i=1}^{n}e^{\lambda_i}\right)$$

where $\lambda_1 \le \cdots \le \lambda_n$ are the eigenvalues of the adjacency matrix A. The scaling factor $1/k!$ is required here in order to have a convergent series and to discount the importance of long walks with respect to short ones. The logarithm and the scaling factor $1/n$ are used to avoid very large numbers as they yield an "average" of the eigenvalues of the adjacency matrix. More in general, we can consider the natural f-connectivity (f-connectivity, in short) as the generalized f-mean of the eigenvalues of A

$$\vartheta(A) = f^{-1}\left(\frac{1}{n}\mathrm{Tr}(f(A))\right) = f^{-1}\left(\frac{1}{n}\sum_{i=1}^{n}f(\lambda_i)\right),$$

where f is a real-valued, increasing, and analytic function on a set containing the spectrum of A.

As f is increasing, it is not difficult to realize that $\vartheta(A)$ itself changes monotonically with the edges of the graph, that is, $\vartheta(A)$ grows if edges are added, and decreases if they are removed. In the following, we assume we are given a *budget k* representing the number of edges, or the cumulative edges' weight, that can be either removed or added to the graph. Thus, we consider the optimization problem of using the given budget to either reduce or increase $\vartheta(A)$ the most.

In matrix terms we can formulate the corresponding optimization problem as follows. Assume we are given the initial graph with adjacency matrix A. We want to find a modification X of the network edges A that either maximizes or minimizes the function $\vartheta(A + X)$, subject to suitable constraints on X which account for the budget and for whether we are removing, adding or modifying the weight of the edges, as detailed next. The constraints on X also depend on whether we are considering weighted or unweighted (binary) networks. To summarize we consider the following three classes of optimization problems.

Edge downgrading

Let us assume that we are given a positive budget k and we want to remove or diminish the weight of the edges that yield the greatest decrease in f-connectivity. Given the graph $G = (V, E)$, we then consider the set of admissible modifications

$$\Omega_k(E) = \left\{X : \sum_{ij}|X_{ij}| \le k,\ X = X^T,\ X_{ij} = 0,\ \text{for } ij \notin E\right\}.$$

The downgrading problem for unweighted graphs, more often referred to as *edge breaking* problem [10], is:

$$\min \vartheta(A + X) \ \text{s.t.}\ X \in \Omega_k(E) \text{ and } X_{ij} \in \{-1, 0\} \tag{DG}$$

while for weighted graphs the second constraint is replaced by $-A_{ij} \le X_{ij} \le 0$, *i.e.*

$$\min \vartheta(A + X) \ \text{s.t.}\ X \in \Omega_k(E) \text{ and } -A_{ij} \le X_{ij} \le 0. \tag{DG'}$$

Edge addition

In this setting, we consider the situation where new edges may be introduced in order to increase the f-connectivity of the network. In this case, given a budget k, the set of admissible modifications takes the form

$$\Omega_k(\overline{E}) = \left\{X : \sum_{ij}X_{ij} \le k,\ X = X^T,\ X_{ij} = 0,\ \text{for } ij \in E\right\}.$$

For unweighted graphs, we obtain the following optimal edge addition problem

$$\max \vartheta(A + X) \ \text{s.t.}\ X \in \Omega_k(\overline{E}) \text{ and } X_{ij} \in \{0, 1\}. \tag{AD}$$

To avoid trivial solutions, where all the budget is spent on a single most important edge, when dealing with weighted networks, we further assume we are given a set of maximum weight values U_{ij} that we are allowed to spend on each edge:

$$\max \vartheta(A + X) \text{ s.t. } X \in \Omega_k(\overline{E}) \ 0 \le X_{ij} \le U_{ij}. \tag{AD'}$$

Edge tuning

Finally, in the third problem, we are given the budget k and a weighted graph G, and we look for a modification of the edge weights of a limited set $F \subseteq E$ of the existing edges in order to obtain the largest increase in f-connectivity. We will also consider the case where F includes both existing and non existing edges, to address the scenario where the creation of new links is also allowed; we call this slightly modified problem *edge rewiring*. As for (AD'), we assume a set of maximum weight values U_{ij} is given, to avoid trivial solutions:

$$\max \vartheta(A + X) \text{ s.t. } X \in \Omega_k(F) \text{ and } -A_{ij} \le X_{ij} \le U_{ij}. \tag{TU}$$

2.1. Algorithmic set-up

Before moving on to the proposed algorithmic techniques, we make several preliminary remarks.

First, we note that since f in the definition of ϑ is an increasing function, then so is f^{-1}. Additionally, note that minimizing (resp. maximizing) the natural f-connectivity is equivalent to minimizing (resp. maximizing) the trace variation

$$\varphi_A(X) := \text{Tr}(f(A + X)) - \text{Tr}(f(A)),$$

with respect to X.

Secondly, we observe that the dimensions of the constraint sets that involve all the existing (or non-existing) edges in the graph are usually very large, already for graphs of moderate size. For this reason, in the rest of the paper we further restrict the optimization problems above to a subset of the edges (or non-existing edges) F whose elements are cleverly selected and whose size is kept under control.

The selection of a suitable F may depend on the problem at hand, and we will call this procedure "the search space selection", which will be discussed case-by-case in Sections 3.2 and 6. Note that, in real applications, further constraints on the set of modifiable edges (or non-existing edges) may be imposed by the application set-up: for example, one may have only access to a certain part of the network (as in the case of a street network where most of the roads may not be modifiable). This additional problem-based constraint can be imposed by straightforward modifications of the above optimization problems.

3. EDGE DOWNGRADING AND ADDITION FOR UNWEIGHTED GRAPHS

In this section we propose some heuristic greedy procedures for addressing the optimization problems (DG) and (AD). We begin by describing the general greedy template that is behind our method and other algorithms proposed in the literature. Throughout the discussion we assume to have a budget of k edges.

3.1. The greedy paradigm

The most intuitive greedy strategy for problem (DG) (resp. (AD)) consists of sequentially removing (resp. adding) the edge that attains the largest reduction (resp. increase) of φ_A until k deletions (resp. additions) are performed. Usually, the identification of the jth edge to be either added or removed is made by evaluating or approximating the variation of φ_A on a large number of candidate edges. Even in the case of an exhaustive search of candidates over the whole edge set (or the whole set of missing edges, in the case of (AD)), this greedy procedure is guaranteed to return the optimal solution only for $k = 1$; on the other hand, when $k > 1$, we expect that the selected set of k edges provides a significant modification of φ_A.

When dealing with medium to large networks, the implementation of this greedy procedure poses two major computational issues:

(i) the large number of edges in the search space to be processed in each step, and

---

Algorithm 1 Template of a greedy method for (DG).

1: **procedure** GREEDY_DOWNGRADE(A, k)
2:     Set $\Delta A = 0$
3:     **for** $j = 1, \ldots, k$ **do**
4:         $X_{\mathsf{opt}} \leftarrow 0$, $\delta_{\mathsf{opt}} \leftarrow +\infty$
5:         Select F_j
6:         **for** $(s, t) \in F_j$ **do**
7:             $X \leftarrow -(\mathbf{1}_s \mathbf{1}_t^T + \mathbf{1}_t \mathbf{1}_s^T)$ ▷ rank 2 modification that deletes (s, t)
8:             Compute $\varphi_A(X) = \mathrm{Tr}(f(A + X)) - \mathrm{Tr}(f(A))$
9:             **if** $\varphi_A(X) \leq \delta_{\mathsf{opt}}$ **then**
10:                 $\delta_{\mathsf{opt}} \leftarrow \varphi_A(X)$
11:                 $X_{\mathsf{opt}} \leftarrow X$
12:             **end if**
13:         **end for**
14:         $\Delta A \leftarrow \Delta A + X_{\mathsf{opt}}$
15:         $A \leftarrow A + X_{\mathsf{opt}}$
16:     **end for**
17:     **return** $\delta_{\mathsf{opt}}, \Delta A$
18: **end procedure**

(ii) the cost of evaluating (or approximating) the cost function φ_A.

Concerning (i), we remark that, when the graph is sparse, an exhaustive search would require considering $\mathcal{O}(n)$ edges for problem (DG) and $\mathcal{O}(n^2)$ edges for problem (AD). When such sets have large sizes, this step can be prohibitively expensive. This is circumvented by restricting the search space for the jth edge to an appropriate subset F_j of moderate size. In the case of (DG), $F_j \subseteq E$, while for (AD) $F_j \subseteq V \times V \setminus E$.

Similarly, task (ii) involves $f(A)$ and $f(A + X)$ but cannot be addressed by directly forming these matrix functions as $f(A)$ is dense almost always, even if A is sparse, and computing $f(A)$ directly would require $\mathcal{O}(n^3)$ operations. Even for small to medium-size matrices, as X changes at each greedy step, computing $f(A + X)$ each time would be prohibitively expensive. Efficient greedy methods make use of techniques that approximate the variation φ_A with a reduced computational cost.

In Algorithm 1 we present a general scheme for the above greedy strategy, for the case of (DG). The analogous algorithm for (AD) is obtained with straightforward modifications at lines 7 and 8, by changing the sign of the rank-2 update and reversing the inequality for δ_{opt}, which has to be initially set to $-\infty$ at line 4. Then, in the next two subsections, we will present our proposed strategy for addressing the two points (i) and (ii) above. In particular, we propose the use of a Krylov subspace-based approach for the approximation of the variation φ_A, which will guarantee an accurate approximation with a computational cost of $\mathcal{O}(n)$, as detailed in subsection 3.3.

3.2. Selection of the search spaces

The strategy for selecting the sets F_j has to ensure a feasible size of the search space and that the most meaningful edges are considered. Intuitively, the second requirement is the trickiest as, due to the combinatorial nature of (DG) and (AD), only an exhaustive search space can guarantee it. The latter choice might be computationally viable for problem (DG) where each F_j has at most $\mathcal{O}(n)$ edges, assuming the initial graph is sparse. If n is moderate and the cost of evaluating $\varphi_A(X)$ is at most linear on n, then we consider the following search spaces

$$\begin{cases} F_1 = E \\ F_{j+1} = E \setminus \mathsf{Chosen}(j), \end{cases} \tag{$\mathrm{S_{DG}^{full}}$}$$

with $\mathsf{Chosen}(j) := \{\text{edges selected in the first } j\text{-th steps of Algorithm 1}\}$.

When strategy ($\mathrm{S_{DG}^{full}}$) is too expensive, an alternative is to define a ranking on the set of edges to heuristically identify the most important ones. Here we propose to rank the edges on the basis of the eigenvector centrality

scores of the nodes they connect, as these scores for the nodes are cheap to evaluate for sparse graphs. More specifically, given two edges (v_1, v_2) and (v_3, v_4), we consider the following two rankings $\leq_1$ and $\leq_2$ on $V \times V$:

$$(v_1, v_2) \leq_1 (v_3, v_4) \quad \Longleftrightarrow \quad \mathsf{eigc}(v_1) \cdot \mathsf{eigc}(v_2) \leq \mathsf{eigc}(v_3) \cdot \mathsf{eigc}(v_4),$$

$$(v_1, v_2) \leq_2 (v_3, v_4) \quad \Longleftrightarrow \quad \begin{cases} \min\{\mathsf{eigc}(v_1), \mathsf{eigc}(v_2)\} < \min\{\mathsf{eigc}(v_3), \mathsf{eigc}(v_4)\} \\ \qquad\qquad\qquad \text{or} \\ \min\{\mathsf{eigc}(v_1), \mathsf{eigc}(v_2)\} = \min\{\mathsf{eigc}(v_3), \mathsf{eigc}(v_4)\} \\ \max\{\mathsf{eigc}(v_1), \mathsf{eigc}(v_2)\} \leq \max\{\mathsf{eigc}(v_3), \mathsf{eigc}(v_4)\} \end{cases} .$$

where $\mathsf{eigc}(v)$ denotes the eigenvector centrality of node $v \in V$, *i.e.* the v-th entry x_v of the Perron eigenvector $\boldsymbol{x}$ of the adjacency matrix. The ordering $\leq_1$ is a standard way of inferring centralities for edges from the node scores [3, 52]. However, we note that $\leq_1$ may still assign large importance to edges that connect a node with small centrality with another having a large centrality; this is prevented by $\leq_2$ which thresholds the edge score by the smallest node centrality involved. We observe that $\leq_2$ works better in practice, as shown in the numerical experiments in Section 5.

Finally, given a subset of edges $F \subseteq V \times V$ and a positive integer q, we denote with $[F]_q^{\leq i}$ the subset of F made by its largest q elements according to $\leq_i$, $i = 1, 2$. The following selection strategies maintain a search space of size q at each step of Algorithm 1:

$$\begin{cases} F_1 & = [E]_q^{\leq_1} \\ F_{j+1} & = [E]_{q+j}^{\leq_1} \setminus \mathsf{Chosen}(j) \end{cases} \quad (\mathrm{S}_{\mathrm{DG}}^1) \qquad \begin{cases} F_1 = [V \times V \setminus E]_q^{\leq_1} \\ F_{j+1} = [V \times V \setminus E]_{q+j}^{\leq_1} \setminus \mathsf{Chosen}(j) \end{cases} \quad (\mathrm{S}_{\mathrm{AD}}^1)$$

$$\begin{cases} F_1 & = [E]_q^{\leq_2} \\ F_{j+1} & = [E]_{q+j}^{\leq_2} \setminus \mathsf{Chosen}(j) \end{cases} \quad (\mathrm{S}_{\mathrm{DG}}^2) \qquad \begin{cases} F_1 = [V \times V \setminus E]_q^{\leq_2} \\ F_{j+1} = [V \times V \setminus E]_{q+j}^{\leq_2} \setminus \mathsf{Chosen}(j) \end{cases} \quad (\mathrm{S}_{\mathrm{AD}}^2)$$

where we have used the subscripts DG and AD to emphasize that the corresponding strategy is meant for problem (DG) and (AD), respectively.

Finally, we describe an additional selection strategy for (AD) proposed in [10], a method we will use as benchmark for comparison in our experiments. Let d be the maximum node degree of the graph and denote by $V_d \subseteq V$ the set of d nodes of largest degrees. Then, the selection strategy uses the missing edges contained in $V_d \times V_d$. This is formally expressed with the following equation:

$$\begin{cases} F_1 = V_d \times V_d \setminus E \\ F_{j+1} = V_d \times V_d \setminus \{E \cup \mathsf{Chosen}(j)\} \end{cases} . \quad (\mathrm{S}_{\mathrm{AD}}^3)$$

Note that, strategy $(\mathrm{S}_{\mathrm{AD}}^3)$ only ensures that the search space has cardinality bounded from above by d^2; this might be a very weak property for certain graph topologies, as $|F_j|$ can be very small.

3.3. Updating the trace of $f(A)$

The main computational efforts of Algorithm 1 come from evaluating $\varphi_A(X)$ at line 8. Note that the matrix X at that step of the algorithm is symmetric and has rank 2. Leveraging this key rank property, we can devise a method of cost $\mathcal{O}(n)$ for computing the variation $\varphi_A(X)$, based on the Krylov subspace method in [5]. We start by describing in Section 3.3.1 the proposed Krylov method; then, in Section 3.3.2 we report another approximation of φ_A that has been previously used in the literature and that will be used as a baseline for comparison later.

3.3.1. A Krylov projection method

Let A be a symmetric adjacency matrix, X a symmetric low-rank modification and $f(z)$ a scalar function. In [5] it has been proved that, under mild assumptions, the matrix $\Delta f := f(A + X) - f(A)$ is of low numerical

rank and its approximation can be performed by means of Krylov subspaces. We will see that, with some minor modifications, this also allows to cheaply approximate $\mathrm{Tr}(\Delta f) = \mathrm{Tr}(f(A+X)) - \mathrm{Tr}(f(A))$.

Let us assume $X = \boldsymbol{U}_X B_X \boldsymbol{U}_X^*$ with $\boldsymbol{U}_X \in \mathbb{R}^{n \times s}$, $B_X = B_X^* \in \mathbb{R}^{s \times s}$ and denote by $\mathcal{K}_m(A, \boldsymbol{U}_X)$ the m-th order *Krylov subspace* generated by A and the (block) vector $\boldsymbol{U}_X$:

$$\mathcal{K}_m(A, \boldsymbol{U}_X) := \mathrm{Span}\{\boldsymbol{U}_X, A\boldsymbol{U}_X, \ldots, A^{m-1}\boldsymbol{U}_X\},$$

where Span indicates the column span. If m steps of the Arnoldi process on A and $\boldsymbol{U}_X$ can be carried out without breakdowns, then it returns an orthonormal basis $\boldsymbol{\mathcal{U}}_m = [\boldsymbol{U}_1 | \ldots | \boldsymbol{U}_m] \in \mathbb{R}^{n \times ms}$ of $\mathcal{K}_m(A, \boldsymbol{U}_X)$ which verifies the following block Arnoldi relation [35]:

$$A\boldsymbol{\mathcal{U}}_m = \boldsymbol{\mathcal{U}}_m \mathcal{H}_m + \boldsymbol{U}_{m+1} H_{m+1,m} \boldsymbol{E}_m^T, \tag{2}$$

with a $ms \times ms$ block tridiagonal matrix $\mathcal{H}_m$, a $s \times s$ matrix $H_{m+1,m}$, and $\boldsymbol{E}_m^T = [0 | \ldots | 0 | I_s] \in \mathbb{R}^{s \times ms}$, where I_s denotes the $s \times s$ identity matrix. An approximation of Δf is given by

$$\Delta f \approx \Delta_m f := \boldsymbol{\mathcal{U}}_m [f(\mathcal{H}_m + \boldsymbol{W}_m B_X \boldsymbol{W}_m^*) - f(\mathcal{H}_m)] \boldsymbol{\mathcal{U}}_m^*, \tag{3}$$

where $\boldsymbol{W}_m := \boldsymbol{\mathcal{U}}_m^* \boldsymbol{U}_X \in \mathbb{R}^{ms \times s}$. The algorithm proposed in [5], reported in Algorithm 2, builds — incrementally in m — the Arnoldi relations (2) and their corresponding quantities $\Delta_m f$. We remark that the matrix $\Delta_m f$ is kept in the factored form $\Delta_m f = \boldsymbol{\mathcal{U}}_m \widetilde{\Delta}_m f \, \boldsymbol{\mathcal{U}}_m^*$ where $\widetilde{\Delta}_m f := f(\mathcal{H}_m + \boldsymbol{W}_m B_X \boldsymbol{W}_m^*) - f(\mathcal{H}_m) \in \mathbb{R}^{ms \times ms}$. The method stops when the heuristic stopping criterion

$$\|\Delta_m f - \Delta_{m-\ell} f\|_2 = \left\| \widetilde{\Delta}_m f - \begin{bmatrix} \widetilde{\Delta}_{m-\ell} f & 0 \\ 0 & 0 \end{bmatrix} \right\|_2 \leq \epsilon$$

is satisfied for a prescribed tolerance ϵ and a positive integer ℓ; in our implementation we set $\ell = 2$. We emphasize that this is just one (arguably, the simplest) of a variety of possible choices for the stopping criterion. Alternative and more accurate methods for computing error estimates of block Arnoldi methods for matrix functions can be used, as discussed for example in [11, 29].

Concerning the approximation error, the method is exact when $f(z)$ is a low degree polynomial; more precisely, $\Delta f = \Delta_m f$ when $f \in \mathcal{P}_{m-1}$, where $\mathcal{P}_{m-1}$ denotes the set of polynomials of degree at most $m-1$. For a more general f, the error norm is linked to the best polynomial approximation of f on a set Π containing the convex hull of the spectrum of A and $A+X$ [5, Theorem 4.1].

We remark that, if the goal is to approximate $\mathrm{Tr}(\Delta f)$, then we can avoid the evaluation of matrix functions at all. Indeed, for computing $\mathrm{Tr}(\Delta_m f) = \mathrm{Tr}(f(\mathcal{H}_m + \boldsymbol{W}_m B_X \boldsymbol{W}_m^*)) - \mathrm{Tr}(f(\mathcal{H}_m))$ it is sufficient to retrieve the eigenvalues of the small symmetric matrices $\mathcal{H}_m$ and $\mathcal{H}_m + \boldsymbol{W}_m B_X \boldsymbol{W}_m^*$, and then apply the function f to them. Since only the approximate eigenvalues are needed here, we replace the Arnoldi method with the Lanczos method for computing the projected matrices. Moreover, a tighter approximation bound is obtained for this particular case, namely [15, Theorem 3]:

$$|\mathrm{Tr}(\Delta f) - \mathrm{Tr}(\Delta_m f)| \leq 4n \min_{p \in \mathcal{P}_{2m}} \max_{z \in \Pi} |f(z) - p(z)|.$$

We report the pseudocode of the procedure for approximating the variation $\mathrm{Tr}(f(A+X)) - \mathrm{Tr}(f(A))$ in Algorithm 3.

Under the assumptions that matrix-vector products with the matrix A cost $\mathcal{O}(n)$, that the rank of X is r, and that k_{it} iterations of the Arnoldi method have been executed before detecting convergence, the cost of Algorithm 2 is $\mathcal{O}(nr^2 k_{\mathrm{it}}^2 + r^3 k_{\mathrm{it}}^4)$. The term of complexity $\mathcal{O}(nr^2 k_{\mathrm{it}}^2)$ comes from the full re-orthogonalization applied in the Arnoldi procedure; moreover, computing $\widetilde{\Delta}_m$ requires the evaluation of two functions of $rm \times rm$ symmetric matrices, which typically needs $\mathcal{O}(r^3 m^3)$, and this yields the term of complexity $\mathcal{O}(r^3 k_{\mathrm{it}}^4)$. An analogous analysis applies to Algorithm 3 that is of complexity $\mathcal{O}(nr k_{\mathrm{it}} + r^3 k_{\mathrm{it}}^4)$; the major difference with

Algorithm 2, is that at, each iteration, the Lanczos method only orthogonalizes with respect to the last two block vectors of the orthonormal basis, and that the eigenvalues of two $rm \times rm$ symmetric matrices are computed in place of their matrix functions. Note that, when calling Algorithm 3 from Algorithm 1 we always have $r = 2$.

Algorithm 2 Low-rank approximation of $f(A + X) - f(A)$.

1: **procedure** FUN_UPDATE($A, \boldsymbol{U}_X, B_X, f, \ell, \epsilon$)
2:     **for** $m = 1, \ldots, m_{\max}$ **do**
3:         Compute (incrementally) the Arnoldi relation for $\mathcal{K}_m(A, \boldsymbol{U}_X)$ by means of the Arnoldi method;
        store $\boldsymbol{\mathcal{U}}_m = [\boldsymbol{U}_1 | \ldots | \boldsymbol{U}_m]$ and $\mathcal{H}_m$
4:         $\boldsymbol{W}_m \leftarrow \boldsymbol{\mathcal{U}}_m^* \boldsymbol{U}_X$
5:         $\widetilde{\Delta}_m f \leftarrow f(\mathcal{H}_m + \boldsymbol{W}_m B_X \boldsymbol{W}_m^*) - f(\mathcal{H}_m)$
6:         **if** $m > \ell$ and $\left\| \widetilde{\Delta}_m f - \begin{bmatrix} \widetilde{\Delta}_{m-\ell} f & 0 \\ 0 & 0 \end{bmatrix} \right\|_2 \leq \epsilon$ **then**
7:             **break**
8:         **end if**
9:     **end for**
10:     **return** $\boldsymbol{\mathcal{U}}_m, \widetilde{\Delta}_m f$
11: **end procedure**

Algorithm 3 Approximation of $\text{Tr}(f(A + X) - f(A))$.

1: **procedure** TRACE_FUN_UPDATE($A, \boldsymbol{U}_X, B_X, f, \ell, \epsilon$)
2:     **for** $m = 1, \ldots, m_{\max}$ **do**
3:         Compute (incrementally) the Arnoldi relation for $\mathcal{K}_m(A, \boldsymbol{U}_X)$ by means of the Lanczos method;
        store $\boldsymbol{\mathcal{U}}_m = [\boldsymbol{U}_1 | \ldots | \boldsymbol{U}_m]$ and $\mathcal{H}_m$
4:         $\boldsymbol{W}_m \leftarrow \boldsymbol{\mathcal{U}}_m^* \boldsymbol{U}_X$
5:         Compute the eigenvalues $\widetilde{\lambda}_j$ of $\mathcal{H}_m + \boldsymbol{W}_m B_X \boldsymbol{W}_m^*$
6:         Compute the eigenvalues λ_j of $\mathcal{H}_m$
7:         $\Delta_m \lambda \leftarrow \sum_j f(\widetilde{\lambda}_j) - f(\lambda_j)$
8:         **if** $m > \ell$ and $|\Delta_m \lambda - \Delta_{m-\ell} \lambda| < \epsilon$ **then**
9:             **break**
10:         **end if**
11:     **end for**
12:     **return** $\Delta_m \lambda$
13: **end procedure**

3.3.2. Approximation via eigendecomposition update

The algorithm *make it or break it* (MIOBI) proposed in [10] approximates the difference of traces by means of a first-order approximation of the largest eigenpairs of $A + X$. More specifically, given a positive integer h, the procedure starts by computing the eigenpairs $(\lambda_1, \boldsymbol{u}_1), \ldots, (\lambda_h, \boldsymbol{u}_h)$ of A, corresponding to the h eigenvalues of largest magnitudes. For each X, the authors of [10] observe that the dominant h eigenpairs $\widehat{\lambda}_j, \widehat{\boldsymbol{u}_j}$ of $A + X$ can be written as

$$\begin{aligned} \widehat{\lambda}_j &= \widetilde{\lambda}_j + \mathcal{O}(\|X\|^2) \\ \widehat{\boldsymbol{u}}_j &= \widetilde{\boldsymbol{u}}_j + \mathcal{O}(\|X\|^2) \end{aligned} \quad \text{with} \quad \begin{aligned} \widetilde{\lambda}_j &= \lambda_j + \boldsymbol{u}_j^* X \boldsymbol{u}_j, \\ \widetilde{\boldsymbol{u}}_j &= \boldsymbol{u}_j + \sum_{i=1, i \neq j}^h \frac{\boldsymbol{u}_i^* X \boldsymbol{u}_j}{\lambda_i - \lambda_j} \boldsymbol{u}_i. \end{aligned}$$

Thus, it is proposed to consider the pairs $(\widetilde{\lambda}_j, \widetilde{\boldsymbol{u}}_j)$ as approximations of $(\widehat{\lambda}_j, \widehat{\boldsymbol{u}}_j)$, *i.e.*, to neglect the high-order terms $\mathcal{O}(\|X^2\|)$. This approach is particularly useful when X is a perturbation with small norm. The

resulting procedure is of the same form as Algorithm 1, with two main modifications: at line 8 the formula $\sum_{j=1}^{h} f(\widetilde{\lambda}_j) - f(\lambda_j)$ is used to approximate the trace update $\mathrm{Tr}(f(A + X)) - \mathrm{Tr}(f(A))$; then at line 15 both formulas for $\widetilde{\lambda}_j, \widetilde{u}_j$ are used to approximate the h dominant eigenpairs of $A + X_{\mathsf{opt}}$. Overall, this yields an algorithm with an iteration cost of $\mathcal{O}(|F_j|h + nh^2)$.

3.4. Algorithms for edge downgrading and edge addition

We are now ready to formally introduce the methods that we propose for solving (DG), (AD):

GREEDY_KRYLOV_BREAK: Algorithm 1 combined with TRACE_FUN_UPDATE for evaluating the difference of traces at line 8 and using the strategy ($\mathrm{S}_{\mathrm{DG}}^2$) for selecting the sets F_j.

GREEDY_KRYLOV_MAKE: Algorithm 1 combined with TRACE_FUN_UPDATE for evaluating the difference of traces at line 8 and using the strategy ($\mathrm{S}_{\mathrm{AD}}^2$) for selecting the sets F_j.

Moreover, to provide a comparison with the performance of state-of-the-art greedy schemes, we consider the following methods:

MIOBI: Greedy method proposed in [10] that uses (3.3.2) for evaluating the difference of traces and the selection strategies ($\mathrm{S}_{\mathrm{DG}}^{\mathrm{full}}$) and ($\mathrm{S}_{\mathrm{AD}}^3$) for (DG) and (AD), respectively.

EIGENV: Method proposed in [3] that consists in deleting or adding the k edges with the largest eigenvector centrality scores — with respect to $\leq_1$ — in E and $V \times V \setminus E$, respectively. The dominant part of its cost is given by the computation of the dominant eigenvector of the adjacency matrix; in our implementation this is done by means of the Matlab function `eigs`.

4. EDGE DOWNGRADING, ADDITION, AND TUNING FOR WEIGHTED GRAPHS

When considering the solution of (DG'), (AD'), and (TU), one needs to deal with a constrained continuous optimization problem involving the objective function $\varphi_A(X)$. Similarly to what has been done for unweighted graphs in Section 3, we keep the size of the problem under control by imposing that we are allowed to modify only a subset F of the edges (or the missing edges), with cardinality n_F. With this constraint, we have that φ_A can be seen as a function of n_F variables $\varphi_A : \mathbb{R}^{n_F} \to \mathbb{R}$, which correspond to the variation of the weights of the edges in F. In particular, the matrix X has rank bounded by $2n_F$; i.e., to efficiently evaluate $\varphi_A(X)$ we can rely on Algorithm 3, as far as $2n_F \ll n$.

We perform the efficient optimization of φ_A via two tailored implementations of an Interior-Point method. The first one approximates the Hessian of the objective function by means of the Limited-memory BFGS algorithm (L-BFGS), which iteratively updates the approximation via rank-2 corrections and only requires the evaluation of the objective function and its gradient. The second one, approximates the true Hessian by means of a Krylov approach. Note that the second approach involves the computation of the second derivatives while the first approach does not. The evaluations of $\varphi_A(X)$ are computed by means of Algorithm 3 as in the discrete setting. The gradient and Hessian computations, instead, require additional analysis as they can be prohibitively expensive if done in a naive way. We devote the remainder of this section to briefly review the L-BFGS algorithm and to the description of numerical methods, presented in Algorithm 5 and Algorithm 8, to efficiently evaluate the gradient and the Hessian of φ_A. The ultimate procedures obtained by combining the Interior-Point method with Algorithm 3 for the objective function evaluation, Algorithm 5 for the gradient, and either L-BFGS or Algorithm 8 for the Hessian, are denoted by KRYLOV_LBFGS and KRYLOV_HESSIAN, respectively. Our implementation of the Interior-Point method relies on the Matlab function `fmincon` that allows us to specify handle functions for the evaluation of the objective function, the gradient, and the Hessian approximation strategy.

4.1. The L-BFGS algorithm

The Limited-memory BFGS algorithm is a variation of the Broyden–Fletcher–Goldfarb–Shanno (BFGS) optimization scheme that reduces the amount of memory storage and operations per step of the original algorithm. For the sake of completeness, we briefly review the main points of L-BFGS in the following and refer the reader to [46] for more details. L-BFGS belongs to the family of quasi-Newton methods, a class of descent-direction unconstrained optimization schemes that, given an objective function φ, uses a search direction of the form $\boldsymbol{d}_k = -B_k \nabla \varphi(\boldsymbol{x}_k)$, with B_k positive definite, to compute the new approximate minimizer for $\min \varphi$ as $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$, with α_k chosen through a suitable line search step. The standard first-order gradient descent method is obtained for $B_k = I$ for all k, while the Newton method is obtained by choosing $B_k = \nabla^2 \varphi(\boldsymbol{x}_k)^{-1}$, the inverse of the Hessian at $\boldsymbol{x}_k$. Rather than inverting the Hessian, which can be computationally prohibitive, BFGS computes an approximation of $\nabla^2 \varphi(\boldsymbol{x}_k)^{-1}$ by performing a rank-2 correction of the previous approximation $B_k = B_{k-1} + R_{k-1}$, with the parameters in the rank-2 matrix R_{k-1} chosen to ensure that (a) B_k is positive definite, and (b) B_k satisfies the secant equation with respect to approximation points $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k-1}$. This update rule brings down the $\mathcal{O}(n_F^3)$ cost of Newton's scheme to $\mathcal{O}(n_F^2)$ and, more importantly, avoids the computations of second derivatives. To further reduce the cost per step, L-BFGS introduces a "history parameter" m and, starting from $B_0 = \gamma_0 I$, it updates B_k only for m steps and then resets B_k to a multiple of the identity $B_k = \gamma_k I$, every m steps. This operation allows one to further reduce cost and memory storage of the method to $\mathcal{O}(m n_F)$, which effectively coincides with $\mathcal{O}(n_F)$ when $m \ll n_F$. In our experiments, we set $m = 10$. The pseudocode for L-BFGS is illustrated in Algorithm 4.

In order to apply the L-BFGS approach to the constrained problems (DG'), (AD'), and (TU), we modify the objective function by introducing a logarithmic barrier for the inequality constraints, following a standard Interior-Point method approach (see *e.g.* [7,8]). In (TU), for example, the objective function is modified into

$$\varphi_\mu(X) := -\varphi_A(X) + \mu \sum_{ij} \big\{ \log(U_{ij} - X_{ij}) + \log(X_{ij} - A_{ij}) \big\}, \qquad \text{with } \mu > 0. \tag{4}$$

L-BFGS is then applied to the unconstrained problem $\min \varphi_\mu$, and the parameter μ is reduced throughout the L-BFGS iterations so that the solution of the approximated problem (4) approaches that of (TU) as the method approaches convergence. In our experiments, the above Interior-Point method approach with L-BFGS is run by means of Matlab's `fmincon` function, with optimization parameters `HessianApproximation=lbfgs` and `HistorySize=10`.

4.2. Gradient approximation *via* Krylov methods

We now look at the gradient of φ_A, for a Fréchet differentiable f. Let us denote by $\text{ind} : F \to \{1, \ldots, n_F\}$ an ordering map on the set F and observe that the derivative with respect to the ijth component of the matrix X is $\partial_{ij} f(A + X) = L_f(A + X, \mathbf{1}_i \mathbf{1}_j^T)$, where $\mathbf{1}_i$ denotes the indicator vector of the node i, $(\mathbf{1}_i)_j = 1$ if $i = j$ and zero otherwise, and $L_f(A + X, \mathbf{1}_i \mathbf{1}_j^T)$ indicates the Fréchet derivative of f at $A + X$, applied to the matrix $\mathbf{1}_i \mathbf{1}_j^T$. Moreover, $L_f(A + X, \mathbf{1}_i \mathbf{1}_j^T + \mathbf{1}_j \mathbf{1}_i^T) = L_f(A + X, \mathbf{1}_i \mathbf{1}_j^T) + L_f(A + X, \mathbf{1}_j \mathbf{1}_i^T)$ and, since $A + X$ is symmetric, it holds $L_f(A + X, \mathbf{1}_i \mathbf{1}_j^T) = L_f(A + X, \mathbf{1}_j \mathbf{1}_i^T)^T$. Putting it all together we have that

$$(\nabla \varphi_A(X))_{\text{ind}(i,j)} = 2 \operatorname{Tr}(L_f(A + X, \mathbf{1}_i \mathbf{1}_j^T)), \qquad \forall (i,j) \in F. \tag{5}$$

In recent work by Schweitzer [49], it has been shown the following identity

$$\operatorname{Tr}(L_f(A + X, \mathbf{1}_i \mathbf{1}_j^T)) = f'(A^T + X^T)_{ij} = f'(A + X)_{ij}, \tag{6}$$

where f' denotes the first derivative of f. Equation (6) is of key importance because it enables us to simplify the calculation of the gradient from computing $\mathcal{O}(n_F^2)$ actions of the Fréchet derivative to evaluating $\mathcal{O}(n_F^2)$ entries of a single matrix function. Moreover, if the quantities $f'(A)_{ij}$ are already given, then we can approximate the

---

Algorithm 4 Pseudocode of L-BFGS for unconstrained optimization problem $\min_{\boldsymbol{x}} \varphi(\boldsymbol{x})$.

1: **procedure** LBFGS($\boldsymbol{x}_0$, γ_0, m, ϵ, maxiter)
2:     $B_0 = \gamma_0 I$
3:     **for** $k = 0, 1, \ldots,$ maxiter **do**
4:         $\boldsymbol{d}_k = -B_k \nabla\varphi(\boldsymbol{x}_k)$
5:         $\alpha_k \leftarrow$ line search using $\{\varphi, \boldsymbol{x}_k, \boldsymbol{d}_k\}$
6:         $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$
7:         **if** $\|\nabla\varphi(\boldsymbol{x}_{k+1})\| < \epsilon$ **then**
8:             **break**
9:         **end if**
10:        **if** $k+1$ is a multiple of m **then**
11:            $\gamma_{k+1} \leftarrow$ scalar approximation of B_k
12:            $B_{k+1} = \gamma_{k+1}I$
13:        **else**
14:            $B_{k+1} \leftarrow$ update B_k using the L-BFGS rule
15:        **end if**
16:    **end for**
17:    **return** $\boldsymbol{x}_{k+1}$
18: **end procedure**

---

difference $f'(A+X)_{ij} - f'(A)_{ij}$ with Algorithm 2. The procedure for evaluating the gradient, for a general f, is reported in Algorithm 5. The cost of the latter is the one of Algorithm 2 plus extracting n_F entries from the low-rank matrix $\boldsymbol{\mathcal{U}}_X \widetilde{\Delta} \boldsymbol{\mathcal{U}}_X^*$; under the assumption that matvecs with A cost $\mathcal{O}(n)$, that X has rank always bounded by $r \leq n_F$, and that Algorithm 2 takes k_{it} iterations to converge (so that $\widetilde{\Delta} \in \mathbb{R}^{(r \cdot k_{\mathrm{it}}) \times (r \cdot k_{\mathrm{it}})}$), we get an overall complexity of $\mathcal{O}((n + n_F)r^2 k_{\mathrm{it}}^2 + r^3 k_{\mathrm{it}}^4)$.

Note that, when $f(z) = e^z = f'(z)$, the evaluation of the objective function and of the gradient simply rely on incorporating the computation of the quantity $\mathrm{Tr}(\widetilde{\Delta})$ in Algorithm 2. The latter has an additional cost of only $\mathcal{O}(r \cdot k_{\mathrm{it}})$ flops.

Algorithm 5 Approximation of $\nabla\varphi_A(X)$.

1: **procedure** GRADIENT_EVAL(A, $\boldsymbol{U}_X$, B_X, $\{f'(A)_{ij}\}_{(i,j)\in\mathrm{ind}^{-1}(\{1,\ldots,n_F\})}$, f', ℓ, ϵ)
2:     $[\boldsymbol{\mathcal{U}}_X, \widetilde{\Delta}] \leftarrow$ FUN_UPDATE(A, $\boldsymbol{U}_X$, B_X, f', ℓ, ϵ)
3:     **for** $h = 1, \ldots, n_F$ **do**
4:         $(i,j) \leftarrow \mathrm{ind}^{-1}(h)$
5:         $\Delta_h \leftarrow \boldsymbol{\mathcal{U}}_X(i, \ :) \cdot \widetilde{\Delta} \cdot \boldsymbol{\mathcal{U}}_X(:, \ j)$
6:         $\nabla\varphi_A(X)_h \leftarrow 2(f'(A)_{ij} + \Delta_h)$
7:     **end for**
8:     **return** $\nabla\varphi_A(X)$
9: **end procedure**

4.3. Hessian evaluation *via* Krylov methods

By taking the partial derivatives of (6) we get the following expression for the Hessian's entries:

$$\left(H\varphi_A(X)\right)_{\mathrm{ind}(i,j),\mathrm{ind}(h,k)} = 2\left(L_{f'}(A+X, \mathbf{1}_i\mathbf{1}_j^T)\right)_{hk} \qquad \forall(i,j),(h,k)\in F. \tag{7}$$

In particular, (7) tells us that computing the Hessian requires extracting $\mathcal{O}(n_F)$ entries from $\mathcal{O}(n_F)$ Frchet derivatives along rank 1 directions. Fortunately, the rank 1 property of the direction implies the low-rank

approximability of $L_{f'}(A + X, \mathbf{1}_i\mathbf{1}_j^T)$ that in turn enables us to leverage an efficient Krylov subspace technique [38], as discussed next.

To simplify the exposition we temporarily replace f' with f and we describe how to efficiently evaluate quantities of the form $L_f(M, \mathbf{1}_i\mathbf{1}_j^T)$, for a given symmetric matrix M and a given function f. The evaluation of the Fréchet derivative in a certain direction can be recast as evaluating the function of a specific augmented matrix. More precisely, applying the well-known formula in [42, Theorem 2.1] to our framework, yields

$$f\left(\begin{bmatrix} M & \mathbf{1}_i\mathbf{1}_j^T \\ 0 & M \end{bmatrix}\right) = \begin{bmatrix} f(M) & L_f(M, \mathbf{1}_i\mathbf{1}_j^T) \\ 0 & f(M) \end{bmatrix} \tag{8}$$

so that we can look at extracting the $(1,2)$ sub-block of (8). Since

$$\begin{bmatrix} 0 & L_f(M, \mathbf{1}_i\mathbf{1}_j^T) \\ 0 & 0 \end{bmatrix} = f\left(\begin{bmatrix} M & 0 \\ 0 & M \end{bmatrix} + \begin{bmatrix} 0 & \mathbf{1}_i\mathbf{1}_j^T \\ 0 & 0 \end{bmatrix}\right) - f\left(\begin{bmatrix} M & 0 \\ 0 & M \end{bmatrix}\right)$$

and $\begin{bmatrix} 0 & \mathbf{1}_i\mathbf{1}_j^T \\ 0 & 0 \end{bmatrix}$ is of rank 1, we expect $L_f(M, \mathbf{1}_i\mathbf{1}_j^T)$ to be well approximated by a low-rank matrix. This property is exploited in [38, Algorithm 2], where a projection method that makes use of tensorized Krylov subspaces has been proposed. The latter incrementally builds orthonormal bases $\boldsymbol{\mathcal{U}}_m, \boldsymbol{\mathcal{V}}_m$ for $\mathcal{K}_m(M, \mathbf{1}_i)$ and $\mathcal{K}_m(M, \mathbf{1}_j)$, respectively, by means of two Arnoldi processes. The associated Arnoldi relations

$$M\boldsymbol{\mathcal{U}}_m = \boldsymbol{\mathcal{U}}_m\mathcal{H}_m + \boldsymbol{U}_{m+1}H_{m+1,m}\mathbf{1}_m^T, \qquad M\boldsymbol{\mathcal{V}}_m = \boldsymbol{\mathcal{V}}_m\mathcal{G}_m + \boldsymbol{U}_{m+1}G_{m+1,m}\mathbf{1}_m^T,$$

directly provide the expression of the projected augmented matrix

$$\begin{bmatrix} \boldsymbol{\mathcal{U}}_m^* & 0 \\ 0 & \boldsymbol{\mathcal{V}}_m^* \end{bmatrix} \begin{bmatrix} M & \mathbf{1}_i\mathbf{1}_j^T \\ 0 & M \end{bmatrix} \begin{bmatrix} \boldsymbol{\mathcal{U}}_m & 0 \\ 0 & \boldsymbol{\mathcal{V}}_m \end{bmatrix} = \begin{bmatrix} \mathcal{H}_m & \mathbf{1}_i\mathbf{1}_j^T \\ 0 & \mathcal{G}_m \end{bmatrix}.$$

Thus, the method computes the quantities

$$L_{f,m}^{(i,j)} := \boldsymbol{\mathcal{U}}_m\widetilde{L}_{f,m}^{(i,j)}\boldsymbol{\mathcal{V}}_m^*, \qquad \widetilde{L}_{f,m}^{(i,j)} := f\left(\begin{bmatrix} \mathcal{H}_m & \mathbf{1}_i\mathbf{1}_j^T \\ 0 & \mathcal{G}_m \end{bmatrix}\right)_{(1,2)}, \tag{9}$$

where the subscript $(1,2)$ refers to the extraction of the $(1,2)$ sub-block, as an approximation of $L_f(M, \mathbf{1}_i\mathbf{1}_j^T)$. The method then stops when the heuristic stopping criterion

$$\left\|\widetilde{L}_{f,m}^{(i,j)} - \begin{bmatrix} \widetilde{L}_{f,m-\ell}^{(i,j)} & 0 \\ 0 & 0 \end{bmatrix}\right\|_2 \leq \epsilon$$

is verified, for a prescribed tolerance ϵ and a positive integer ℓ. In our implementation we set $\ell = 2$. For an alternative and more reliable stopping criterion see Section 5 of [37]. The full procedure is reported in Algorithm 6.

We point out that, the approximation error associated with the sequence $L_{f,m}^{(i,j)}$, $m = 1, 2, \ldots$, decays at least as the best polynomial approximation error of f' on the convex hull of the spectrum of M, which we denote by Π. More precisely, a direct consequence of [38, Corollary 1] is the following bound:

$$\left\|L_f(M, \mathbf{1}_i\mathbf{1}_j^T) - L_{f,m}^{(i,j)}\right\|_F \leq 2 \min_{p \in \mathcal{P}_{m-1}} \max_{z \in \Pi} |f'(z) - p(z)|.$$

Note that, the cost analysis of Algorithm 6 is very similar to the one of Algorithm 2. In particular, under the assumptions that matvecs with M cost $\mathcal{O}(n)$, and that the Arnoldi procedure takes k_{it} iterations before detecting convergence, Algorithm 6 costs $\mathcal{O}(n \cdot k_{\text{it}}^2 + k_{\text{it}}^4)$.

Algorithm 6 Approximation of $L_f(M, \mathbf{1}_i \mathbf{1}_j^T)$.

1: **procedure** FRECHET_EVAL($M, i, j, f, \ell, \epsilon$)
2:     **for** $m = 1, \ldots, m_{\max}$ **do**
3:         Compute (incrementally) the Arnoldi relation for $\mathcal{K}_m(M, \mathbf{1}_i), \mathcal{K}_m(M, \mathbf{1}_j)$ by means of the Arnoldi method; store $\boldsymbol{\mathcal{U}}_m, \boldsymbol{\mathcal{V}}_m, \mathcal{H}_m$ and $\mathcal{G}_m$
4:         $\widetilde{L}_{f,m}^{(i,j)} \leftarrow f\left(\begin{bmatrix} \mathcal{H}_m & \mathbf{1}_i \mathbf{1}_j^T \\ 0 & \mathcal{G}_m \end{bmatrix} \right)_{(1,2)}$
5:         **if** $m > \ell$ and $\left\| \widetilde{L}_{f,m}^{(i,j)} - \begin{bmatrix} \widetilde{L}_{f,m-\ell}^{(i,j)} & 0 \\ 0 & 0 \end{bmatrix} \right\|_2 \leq \epsilon$ **then**
6:             **break**
7:         **end if**
8:     **end for**
9:     **return** $\boldsymbol{\mathcal{U}}_m, \boldsymbol{\mathcal{V}}_m, \widetilde{L}_{f,m}^{(i,j)}$
10: **end procedure**

4.3.1. Multiple evaluations of $L_f(M, \mathbf{1}_i \mathbf{1}_j^T)$

Evaluating (7) requires to approximate the quantities $L_{f'}(A, \mathbf{1}_i \mathbf{1}_j^T)$ for all edges $(i,j) \in F$, with $|F| = n_F \ll n$. In principle, running Algorithm 6 n_F times (on each pair $(i,j) \in F$) performs the sought evaluation. On the other hand, it is possible to enhance the efficiency by avoiding redundant computations due to the repetition of the same nodes in edges of F and thus the same Krylov subspaces. Denote by $V(F)$ the set of nodes that are linked by the edges in F, i.e., $V(F) := \{i \in V : \exists j \in V \text{ such that } (i,j) \in F\}$ and for any such node $i \in V(F)$ let $V_i(F)$ be the set of nodes that are connected to i via an edge in F, i.e., $V_i(F) := \{j \in V : (i,j) \in F\} \subseteq V(F)$. We proceed as follows:

(i) For each $i \in V(F)$ we compute and store the Arnoldi relation

$$M \boldsymbol{\mathcal{U}}_{m_i}^{(i)} = \boldsymbol{\mathcal{U}}_{m_i}^{(i)} \mathcal{H}_{m_i}^{(i)} + \boldsymbol{U}_{m_i+1}^{(i)} H_{m_i+1,m_i}^{(i)} \mathbf{1}_{m_i}^T$$

for $\mathcal{K}_{m_i}(M, \mathbf{1}_i)$ where m_i is such that $\left\| \widetilde{L}_{f,m_i}^{(i,j)} - \begin{bmatrix} \widetilde{L}_{f,m_i-\ell}^{(i,j)} & 0 \\ 0 & 0 \end{bmatrix} \right\|_2 \leq \epsilon$, for all $j \in V_i(F)$.

(ii) While doing (i), for each pair $(i,j) \in F$, we store $\widetilde{L}_{f,m_{(i,j)}}^{(i,j)}$ where $m_{(i,j)}$ is the smallest integer such that $\left\| \widetilde{L}_{f,m_{(i,j)}}^{(i,j)} - \begin{bmatrix} \widetilde{L}_{f,m_{(i,j)}-\ell}^{(i,j)} & 0 \\ 0 & 0 \end{bmatrix} \right\|_2 \leq \epsilon$. Note that, $m_{(i,j)} \leq m_i$ which may yield a cheaper trace evaluation for that particular $(i,j) \in F$.

The procedure which implements these enhancements is reported in Algorithm 7.

Let us denote by $n_V := |V(F)|$ and $k_{\mathrm{it}} = \max_{(i,j) \in F} m_{(i,j)}$ and assume that matvecs with M cost $\mathcal{O}(n)$. Then, the complexity of Algorithm 7 is determined by n_V times the one of Algorithm 6, i.e., $\mathcal{O}(n_V(n \cdot k_{\mathrm{it}}^2 + k_{\mathrm{it}}^4))$. We remark that Algorithm 7 requires to store $\mathcal{O}(n_V \cdot k_{\mathrm{it}})$ vectors of length n to represent all the Krylov bases; this might not be feasible for a large value of n_V, i.e., a large search space F.

Finally, the procedure that evaluates the Hessian of $\varphi_A(X)$ is reported in Algorithm 8. The latter consists in one call to Algorithm 7 and extracting n_F entries from a matrix of rank $r \cdot k_{\mathrm{it}}$, $\mathcal{O}(n_F)$ times; this yields a complexity estimate of $\mathcal{O}(n_V(n \cdot k_{\mathrm{it}}^2 + k_{\mathrm{it}}^4) + n_F^2 k_{\mathrm{it}}^2)$.

5. NUMERICAL EXPERIMENTS WITH UNWEIGHTED GRAPHS

We test the performance of GREEDY_KRYLOV_BREAK and GREEDY_KRYLOV_MAKE, introduced in Section 3.4, with respect to their effectiveness in manipulating the graph natural connectivity, i.e. $f(z) = e^z$, and their running time on 22 real-world unweighted networks. Details about the networks' size are reported in Table 1; in particular, the corresponding adjacency matrices are of size $|V| \times |V|$ and have at most $2|E|$ nonzero entries. Those

---

Algorithm 7 Approximation of $L_f(M, \mathbf{1}_i\mathbf{1}_j^T) \; \forall (i,j) \in F$.

1: **procedure** MULTIPLE_FRECHET_EVAL(M, F, f, ℓ, ϵ)
2:     NC $\leftarrow F$, $m = 1$ ▷ NC is the set of not converged edges
3:     **while** NC$\neq \emptyset$ **do**
4:         **for** $i \in V(F)$ **do**
5:             **if** $\exists j \in V_i(F)$ such that $(i,j) \in$ NC **then**
6:                 Compute (incrementally) the Arnoldi relation for $\mathcal{K}_m(M, \mathbf{1}_i)$ by means of the Arnoldi method;
                store $\boldsymbol{\mathcal{U}}_m^{(i)}$ and $\mathcal{H}_m^{(i)}$
7:             **end if**
8:         **end for**
9:         **for** $(i,j) \in$ NC **do**
10:             $\widetilde{L}_{f,m}^{(i,j)} \leftarrow f\left(\begin{bmatrix} \mathcal{H}_m^{(i)} & \mathbf{1}_i\mathbf{1}_j^T \\ 0 & \mathcal{H}_m^{(j)} \end{bmatrix} \right)_{(1,2)}$
11:             **if** $m > \ell$ and $\left\| \widetilde{L}_{f,m}^{(i,j)} - \begin{bmatrix} \widetilde{L}_{f,m-\ell}^{(i,j)} & 0 \\ 0 & 0 \end{bmatrix} \right\|_2 \leq \epsilon$ **then**
12:                 NC $\leftarrow$ NC$\backslash(i,j)$, $m_{(i,j)} \leftarrow m$
13:             **end if**
14:         **end for**
15:         $m \leftarrow m + 1$
16:     **end while**
17:     **return** $\boldsymbol{\mathcal{U}}_m^{(h)}$ for all nodes $h \in V(F)$ and $\widetilde{L}_{f,m_{(i,j)}}^{(i,j)}$ for all edges $(i,j) \in F$
18: **end procedure**

Algorithm 8 Approximation of $H\varphi_A(X)$.

1: **procedure** HESSIAN_EVAL(A, $\boldsymbol{U}_X$, B_X, F, f', ℓ, ϵ)
2:     $\{\boldsymbol{\mathcal{U}}_{\text{ind}(i,j)}, \widetilde{L}^{(i,j)}\}_{(i,j)\in F} \leftarrow$ MULTIPLE_FRECHET_UPDATE($A + \boldsymbol{U}_X B_X \boldsymbol{U}^*$, F, f', ℓ)
3:     **for** $s = 1, \ldots, n_F$ **do**
4:         $(i, \; j) \leftarrow \text{ind}^{-1}(s)$
5:         **for** $t = s, \ldots, n_F$ **do**
6:             $(h, \; k) \leftarrow \text{ind}^{-1}(t)$
7:             $H\varphi_A(X)_{st} \leftarrow 2 \, \boldsymbol{\mathcal{U}}_s(h, \; 1:m_{(i,j)}) \, \widetilde{L}^{(i,j)} \, \boldsymbol{\mathcal{U}}_s(k, \; 1:m_{(i,j)})^*$
8:             $H\varphi_A(X)_{ts} \leftarrow H\varphi_A(X)_{st}$
9:         **end for**
10:     **end for**
11:     **return** $H\varphi_A(X)$
12: **end procedure**

---

listed on the left-hand side of Table 1 include social networks of geolocated reciprocated Twitter mentions within UK cities (Cardiff, Edinburgh), coauthorship networks (ca-AstroPh, ca-CondMat, ca-HephTh, netscience), a protein-protein interactions network (yeast) and a public transports network (London). All these networks are publicly available *via* public repositories, as reported in [4, 13, 34, 50]. All the networks listed on the right-hand side of Table 1 are road networks of different cities in the world [28]. Our implementation is written using MATLAB and is available at the public repository https://github.com/COMPiLELab/krylov_robustness, together with all the datasets above.

In the proposed experiments we compare with state-of-the-art methods MIOBI and EIGENV, that have been recalled in Section 3.4. In particular, MIOBI uses 25 eigenpairs to compute the approximate trace variation as described in Section 3.3.2 and the search spaces S_{DG}^{full}, S_{AD}^3 for problems DG and AD, respectively. If not stated otherwise, the f-connectivity is considered with respect to be the matrix exponential function, *i.e.*, $f = \exp$.

To assess the impact of the various methods on the natural connectivity of a network we consider the *magnitude of the relative trace variation* that, given the returned modification of the adjacency matrix X, we

TABLE 1. Number of vertices and edges of the unweighted graphs used for the numerical tests. On the left: social, collaboration, transportation, and PPI networks; on the right: graphs representing road networks.

Dataset	$|V|$	$|E|$	Dataset	$|V|$	$|E|$
Cardiff	2685	4444	Anaheim	416	634
CollegeMsg	1893	13835	Austin	7388	10591
Edinburgh	1645	2146	Barcelona	930	1798
as_735	6474	12572	Birmingham	14578	20913
ca-AstroPh	17903	19972	ChicagoRegional	12979	20627
ca-CondMat	21363	91286	DC	9522	14807
ca-HephTh	8638	24806	Hawaii	21774	26007
London	369	430	Philadelphia	13389	21246
netscience	379	914	RhodeIsland	51642	66650
socEpinions1	75877	405739	Rome	3353	4831
yeast	2224	6609	Sydney	32956	38787

define as:

$$\Delta T(X) := \frac{|\mathrm{Tr}(f(A + X)) - \mathrm{Tr}(f(A))|}{|\mathrm{Tr}(f(A))|}.$$

To obtain an estimate of the denominator $\mathrm{Tr}(f(A))$ we have employed the stochastic trace estimator HUTCH++ [44] combined with the `expmv` algorithm from [1] to evaluate the action of the matrix exponential on Rademacher random vectors.

Finally, to evaluate the scalability of the approaches we report their computational times in seconds. The latter do not include the time spent for estimating $\mathrm{Tr}(f(A))$ at the beginning, as this operation is not required by the greedy procedures.

The experiments have been performed on a laptop with a dual-core Intel Core i7-7500U 2.70 GHz CPU, 256 KB of level 2 cache, 16 GB of RAM, and operating system Ubuntu 22.04.2. The algorithms are implemented in MATLAB and tested under MATLAB2022b, with MKL BLAS version 2019.0.3 utilizing both cores.

5.1. Downgrading for unweighted graphs

As a first experiment, we measure the quantity ΔT when solving problem (DG) with a fixed budget of $k = 50$ edges to be removed. The parameter q, used by the method GREEDY_KRYLOV_BREAK to determine its search space, is set to the value 250.

The performances of GREEDY_KRYLOV_BREAK, MIOBI, and EIGENV are compared over both road and general networks. The results reported in the left part of Table 2 show that MIOBI and GREEDY_KRYLOV_BREAK always outperform EIGENV on road networks and our GREEDY_KRYLOV_BREAK achieves the best score on 6 out of 11 case studies. Also, for general graphs, MIOBI and GREEDY_KRYLOV_BREAK provide the best scores although the results reported in the right part of Table 2 show a balanced situation: on 7 out of 11 case studies, the difference between the scores of the methods is less than 2%. The most evident gain of the top method is measured for the medium-size graph `ca-HephTh` and the small graph `netscience`. In view of the significantly lower costs of MIOBI and EIGENV (see Sect. 5.1.1), these results suggest that GREEDY_KRYLOV_BREAK can be a valid competitor for the road networks dataset only.

5.1.1. Trace reduction and scalability with respect to the budget size

Now we consider a second numerical test where we let the budget size k range in the set $\{10 \cdot j\}$, $j = 1, \ldots, 10$, and we measure both the relative trace variation and the time consumption of the methods. Further, we investigate how the parameter q, that determines the size of the search space, affects the performance of

TABLE 2. Magnitude of the relative trace variation obtained with the three methods GREEDY_KRYLOV_BREAK (GKB), MIOBI, EIGENV considered for the downgrading of unweighted graphs on road networks (left) and general networks (right), with a budget of $k = 50$ edges. The column denoted with $\cap$ shows the number of edges that have been commonly chosen by all the methods.

| | **Downgrading** | | | | | | | | |
	GKB	MIOBI	EIGENV	$\cap$		GKB	MIOBI	EIGENV	$\cap$
Anaheim	**0.123**	0.0956	0.0775	6	Cardiff	0.974	**0.974**	0.973	43
Austin	0.00863	**0.00943**	0.00564	6	CollegeMsg	**0.773**	0.771	0.771	45
Barcelona	0.0871	**0.0900**	0.0634	10	Edinburgh	0.326	**0.335**	0.240	19
Birmingham	0.00364	**0.00478**	0.00234	4	as735	0.965	**0.966**	**0.966**	45
ChicagoRegional	**0.00530**	0.00501	0.00317	4	AstroPh	**0.751**	0.751	0.751	49
DC	**0.00682**	0.00643	0.00417	5	CondMat	**0.858**	0.854	0.854	45
Hawaii	0.00273	**0.00287**	0.00198	7	HepTh	**0.958**	0.847	0.847	5
Philadelphia	**0.00348**	0.00340	0.00236	2	London	**0.158**	0.151	0.119	12
RhodeIsland	**0.00125**	0.00124	0.000752	2	netscience	0.704	**0.814**	0.744	18
Rome	**0.0161**	0.0158	0.0101	3	Epinions1	0.581	**0.587**	**0.587**	41
Sydney	0.00148	**0.00250**	0.00109	2	yeast	**0.878**	0.871	0.865	36

GREEDY_KRYLOV_BREAK by considering three implementations of this method for $q = 50, 250, \min\{1000, |E| - k\}$. As case studies, we select 6 road networks: `Anaheim`, `Birmingham`, `ChicagoRegional`, `Hawaii`, `RhodeIsland`, and `Rome`. Figure 1 reports the magnitude of the relative trace variations attained by the five methods, as the budget increases. The method GREEDY_KRYLOV_BREAK with the largest search space attains the highest scores on all the examples apart from `Birmingham`, where the returned trace variation is comparable with the one of MIOBI. There is no clear winner between MIOBI and GREEDY_KRYLOV_BREAK with $q = 500$, while GREEDY_KRYLOV_BREAK with $q = 50$ and EIGENV always provide the 4th and the 5th scores.

The computational times shown in Figure 2 confirm that the cost of all algorithms has a linear scaling with respect to the parameter k. Also, their dependence on n is linear, but the hidden constant determines significantly different running times. In particular, in all case studies the three implementations of GREEDY_KRYLOV_BREAK are the most expensive, then we have MIOBI and, finally, eigenv that is the cheapest method. As expected, reducing the parameter q improves the timings of GREEDY_KRYLOV_BREAK, however, in view of the scores in Figure 1, the convenience of a smaller search space is questionable. Overall, these results suggest that GREEDY_KRYLOV_BREAK is preferable in a scenario where the robustness reduction matters more than the computing time.

5.2. Addition for unweighted graphs

Here we consider analogous tests to those performed in the previous section, for the optimization problem (AD). This time, we compare MIOBI and EIGENV with the performance of our GREEDY_KRYLOV_MAKE with $q = \min\{1000, |E|\}$. In the left and right parts of Table 3 it is reported the magnitude of the relative trace variation, obtained with a budget $k = 50$, for road and general networks, respectively. For all road networks, GREEDY_KRYLOV_MAKE is the clear-cut winner and outperforms the second-highest score of a factor between 1.5 and 5. For general networks, GREEDY_KRYLOV_MAKE obtains either the best or near-best score on 10 out of 11 examples, although the gain with respect to the competitors is often more limited than for road networks.

Then, we investigate the impact of varying the budget size k in the range $10, 20, \ldots, 100$ on the trace variation and the computational time for the road networks considered in section 5.1.1. Also in this case, we consider three different sizes for the search space of GREEDY_KRYLOV_MAKE, corresponding to the choices of the parameter q in the set of values $50, 250, \min\{1000, |E|\}$. Figure 3 reports the magnitude of the relative trace variation
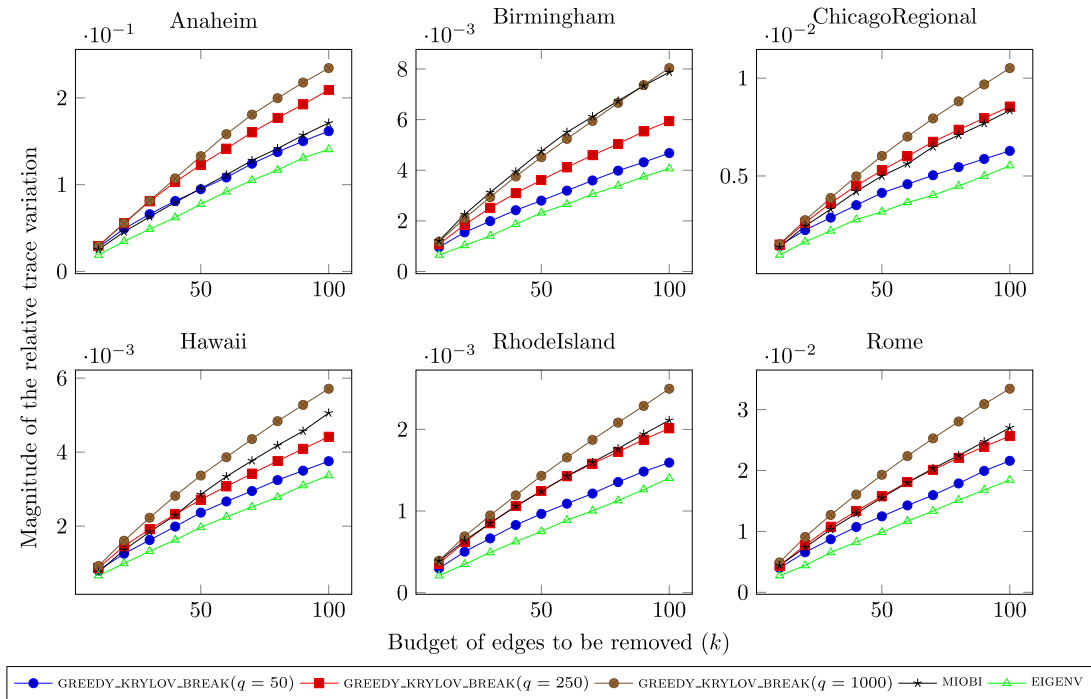
FIGURE 1. Magnitude of the relative trace variation for downgrading as the budget increases.
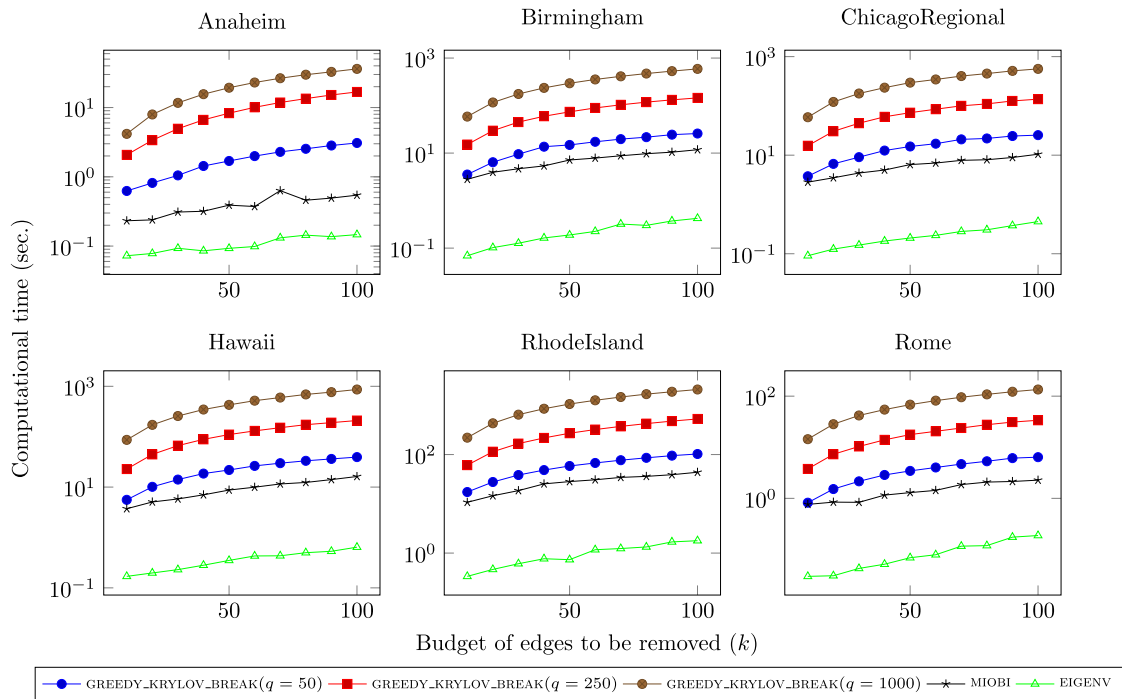

FIGURE 2. Computational times (seconds) of the methods for downgrading as the budget increases.

TABLE 3. Magnitude of the relative trace variation obtained with the three methods GREEDY_KRYLOV_MAKE (GKM), MIOBI, EIGENV considered for the addition of edges to unweighted graphs on road networks (left) and general networks (right), with a budget of $k = 50$ edges. The column denoted with $\cap$ shows the number of edges that have been commonly chosen by all the methods.

	Addition								
	GKM	MIOBI	EIGENV	$\cap$		GKM	MIOBI	EIGENV	$\cap$
Anaheim	**42.4**	12.8	15.9	31	Cardiff	**24.0**	20.6	20.6	37
Austin	**3.49**	2.34	2.34	37	CollegeMsg	**5.14**	5.04	5.04	43
Barcelona	**29.5**	12.3	12.3	30	Edinburgh	**54.2**	19.9	19.9	24
Birmingham	**1.36**	0.353	0.372	19	as735	1.12	**2.31**	**2.31**	10
ChicagoRegional	**1.64**	0.558	0.613	21	AstroPh	**1.28**	1.27	1.27	39
DC	**2.13**	0.197	0.495	19	CondMat	4.66	**4.94**	**4.94**	30
Hawaii	**1.13**	0.0745	0.271	16	HepTh	**3.16**	2.78	2.78	39
Philadelphia	**1.43**	0.162	0.293	14	London	**70.9**	26.9	26.9	36
RhodeIsland	**0.469**	0.150	0.150	27	netscience	**52.1**	30.5	30.5	25
Rome	**6.33**	2.34	2.17	22	Epinions1	1.04	**1.13**	**1.13**	28
Sydney	**0.794**	0.274	0.274	38	yeast	**23.7**	20.4	20.4	32

and highlights a crucial difference with respect to the downgrading problem: For any size of the search space, GREEDY_KRYLOV_MAKE outperforms significantly its competitors on all case studies. We also note that, in contrast to the downgrading case, EIGENV has either comparable or better performances than MIOBI on all case studies. Moreover, the computational times reported in Figure 4 demonstrate that by choosing the smallest size of the search space ($q = 50$), the cost of GREEDY_KRYLOV_MAKE becomes comparable to the one of MIOBI. This is also due to the fact that, for the addition problem, the search space of MIOBI might be significantly larger than in the downgrading case. Therefore, we conclude that GREEDY_KRYLOV_MAKE should be the method of choice for problem (AD), unless a very strict limitation on the time consumption has to be applied.

6. NUMERICAL EXPERIMENTS WITH WEIGHTED GRAPHS: TUNING, REWIRING, ADDITION

Finally, we present results on a set of weighted networks in order to test the performance of the proposed method for the edge tuning problem (TU), as well as weighted edge-addition and edge rewiring, where we simultaneously tune the weight of existing edges and add new ones.

While in certain applications the set F of edges (or missing edges) that we are allowed to modify is given a-priori, in our setup we will assume only the cardinality of the set F is fixed, *i.e.* we are free to select a set of $n_F \geq 1$ modifiable edges (or edges to be added) and we need to form F by choosing which ones are those that are best suited to maximize the natural connectivity. This is a more challenging scenario and, clearly, the case in which the set F is specified by external constraints is retrieved as a special case.

In order to find an optimal set F, we propose to measure how sensitive the f-connectivity, with $f \in \{\exp, \sinh\}$, with respect to changes in the weight of a certain edge (i, j) is. To this end, one should look at the magnitude of the corresponding gradient entry $2f'(A + X)_{ij}$ and select the edges corresponding to the largest gradient. However, inspecting these quantities for all edges (or missing edges) can be too expensive for large networks. Thus, in our experiments, we proceed as follows: first, we select a set of n_P candidate edges, with $n_P > n_F$, chosen as the most important, with respect to a suitable edge-ordering, among existing and/or non-existing edges; then, we identify F on the basis of the evaluations of the gradient over the n_P candidate edges.

In our experiments, we test KRYLOV_LBFGS and KRYLOV_HESSIAN on a set of electric power grid networks from different countries, as listed in Table 4. All the considered network datasets were collected from an Open
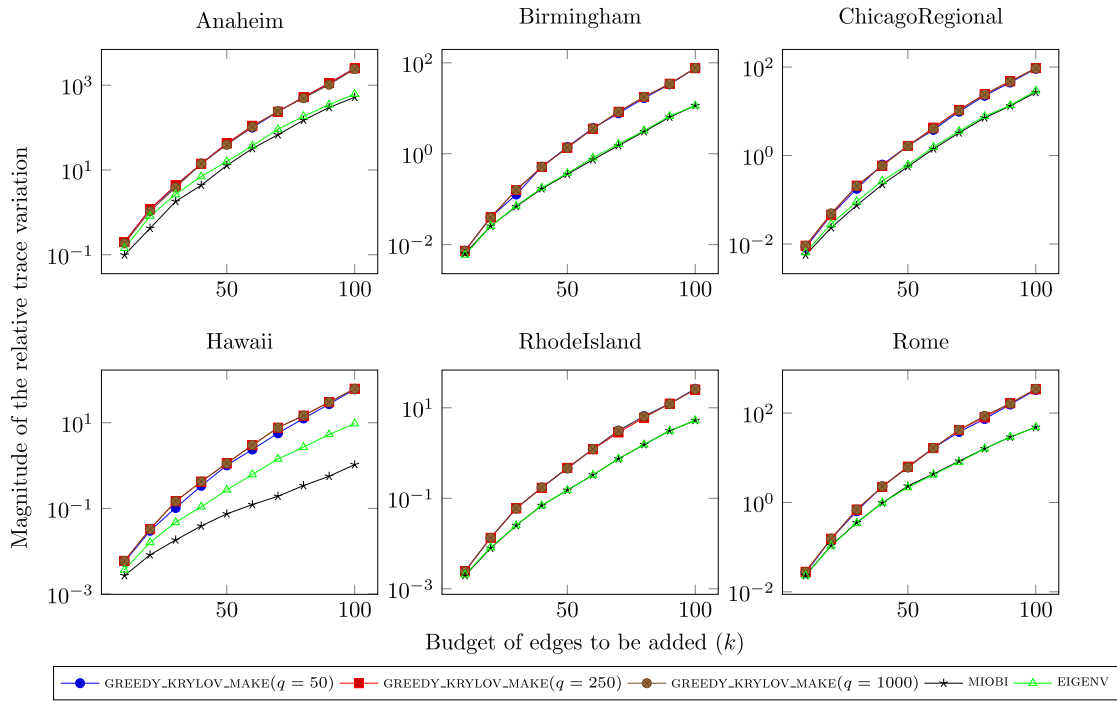
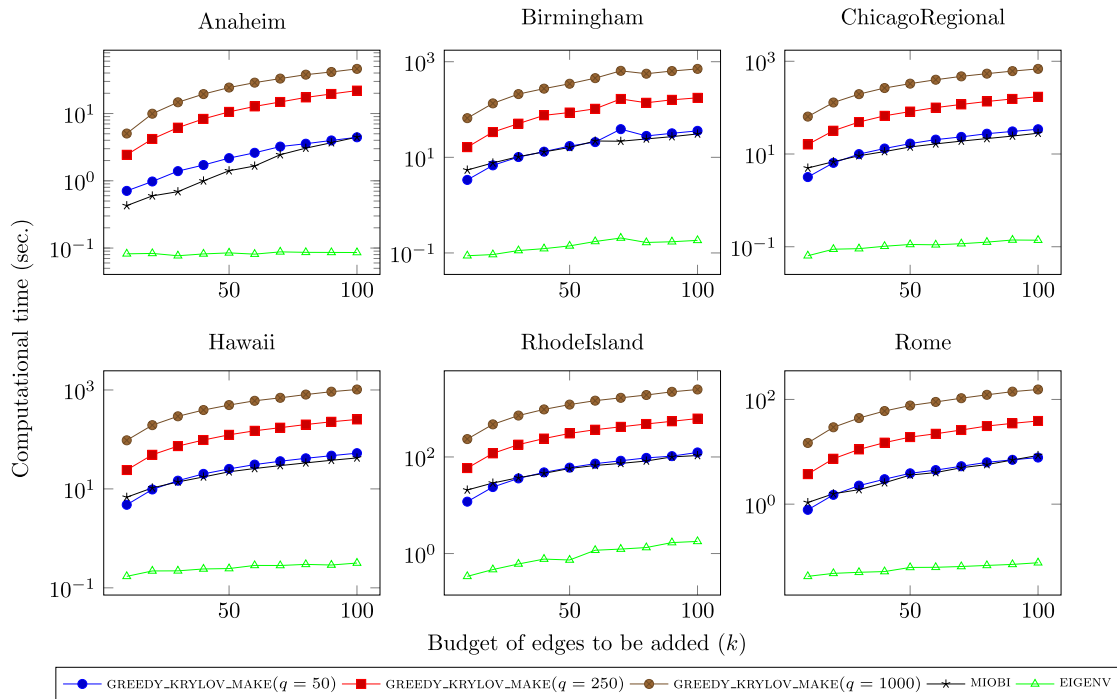FIGURE 3. Magnitude of the relative trace variation for addition as the budget increases.

FIGURE 4. Computational times (seconds) of the methods for addition as the budget increases.

TABLE 4. Number of vertices and edges of the weighted graphs of power grids used for the numerical tests. In bracket the percentage of voltages that has been added in our preprocessing stage.

Dataset	$|V|$	$|E|$
Austria	149	169 (0%)
Denmark	96	105 (0%)
England	504	603 (0.7%)
Germany	1903	2371 (0.4%)
Italy	858	1092 (2.8%)
India	3228	4323 (0.4%)
Mexico	552	743 (2.8%)
Poland	299	390 (0.3%)
Portugal	185	247 (1.5%)
Sweden	268	336 (2.4%)

Street Map project by the Complex Network Group at Telecom Sud-Paris [14]. Each node represents a power station and edges represent wired connections, weighted by their voltage capacity. A small number (in most cases less than 1%) of edge voltage capacity data was missing in the original datasets. For those edges we artificially set the voltage capacity as the average of the neighbors. In all the tests of this section, we consider the total weight budget $k = 10$.

Concerning the selection of the edges in F, we propose three different approaches that deal with different scenarios, as listed below.

Tuning. This approach applies to the case where we are only allowed to modify edges with an initial non-zero weight. We select the candidate edges as the first $n_P = 100$ existing edges with respect to $\leq_1$; then, we set F as the $n_F = 30$ edges, among the candidates, with the largest value of the gradient.

Rewiring. This approach applies to the case where we are allowed to both modify existing edges and add new ones. We select two sets C_1 and C_2 of 50 candidate edges each, as the first 50 existing edges with respect to $\leq_2$ and the first 50 non-existing edges with respect to $\leq_2$, respectively. The resulting set of $n_P = 100$ candidate pairs is then used to form F by choosing $n_F = 30$ elements from the union of the 15 edges in C_1 and non-existing edges in C_2 with the largest value of the gradient.

Addition. This approach applies to the case where we are only allowed to add new edges. We select the candidate edges as the first $n_P = 100$ non-existing edges with respect to $\leq_2$; then, we set F as the $n_F = 30$ edges, among the candidates, with the largest value of the gradient.

Table 5 shows the relative trace variation and the execution time (in seconds), obtained with KRYLOV_LBFGS and KRYLOV_HESSIAN, for all the datasets and the three problem cases above. The values of ΔT obtained with the two methods are very close, indeed their difference is more than the 10% of the highest value only in two cases: Austria (Rewiring) and Portugal (Addition). Figure 5 shows the geographical location of the modified and added edges on the power network of Denmark, obtained with KRYLOV_LBFGS, where red edges denote edges whose weight has been diminished by the algorithm, green edges are edges whose weight was increased, and yellow lines denote edges that were added. As expected, rewiring is always the most effective procedure, resulting in the largest increase in natural connectivity, as it combines edge tuning and edge addition in a simultaneous optimization mechanism. In particular, we see from Figure 5 that the set of edges modified and added by Rewiring is a subset of those that are modified and added by the other two approaches.

Empirically, we observe that KRYLOV_HESSIAN always converges in less iterations; however, the latter are more expensive and there is no clear winner between the two methods, in terms of speed; KRYLOV_LBFGS is faster on 16 examples while KRYLOV_HESSIAN is faster on 14.
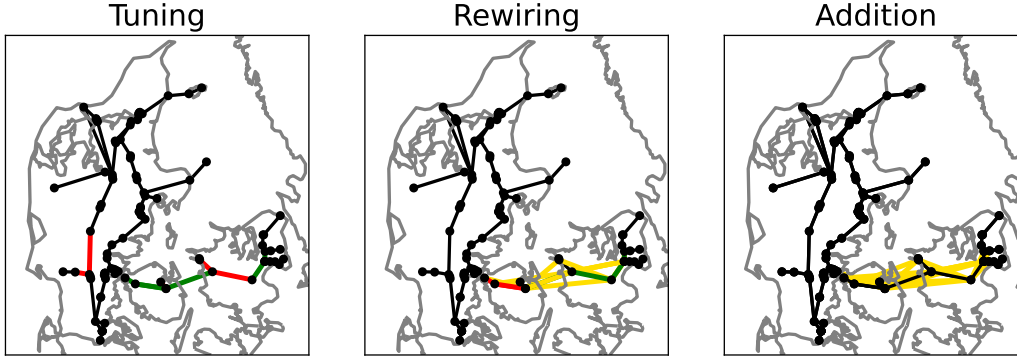
FIGURE 5. Results of KRYLOV_LBFGS on the electric power grid network of Denmark. Red lines correspond to edges whose weight was decreased; Green lines correspond to edges whose weight was increased; Yellow lines denote edges that have been added from scratch.

TABLE 5. Magnitude of the relative trace variation (ΔT), execution time in seconds (t), and number of iterations (k_{it}), for the tuning, rewiring, and addition optimization problems solved with KRYLOV_LBFGS and KRYLOV_HESSIAN approaches, for weighted graphs associated with power grid networks and for $f = \exp$.

	Exponential																	
	TUNING						REWIRING						ADDITION					
	LBFGS			HESSIAN			LBFGS			HESSIAN			LBFGS			HESSIAN		
	ΔT	t	k_{it}	ΔT	t	it	ΔT	t	k_{it}	ΔT	t	k_{it}	ΔT	t	k_{it}	ΔT	t	k_{it}
Austria	0.76	0.2	29	**0.76**	3.2	15	2.89	0.4	34	**3.2**	1.3	12	**0.55**	0.2	20	**0.55**	1.5	11
Denmark	**1.22**	0.2	47	1.2	1.5	14	**6.42**	0.1	34	**6.42**	1.4	12	**0.71**	0.2	28	**0.71**	1.2	13
England	**0.32**	3.2	34	**0.32**	3.9	14	0.8	9.9	54	**0.8**	4.5	16	0.2	1.3	19	**0.22**	2.1	13
Germany	0.19	6.5	42	**0.19**	4.9	17	**0.53**	2.4	45	**0.53**	2.1	12	**0.08**	1.2	22	**0.08**	2.0	12
India	**0.12**	9.0	38	**0.12**	6.5	15	**0.34**	7.5	53	0.33	4.4	16	**0.06**	2.3	17	**0.06**	2.6	10
Italy	**0.41**	7.3	37	**0.41**	4.1	12	**1.62**	4.6	46	**1.62**	2.7	12	**0.16**	2.3	27	**0.16**	2.0	12
Mexico	**0.58**	7.7	40	**0.58**	4.2	12	1.66	13.6	47	**1.66**	4.5	12	0.2	1.4	18	**0.21**	2.0	11
Poland	**0.62**	2.6	37	**0.62**	2.1	13	**1.76**	1.8	34	**1.76**	1.7	10	**0.31**	1.8	31	**0.31**	2.1	14
Portugal	**1.02**	0.5	42	**1.02**	1.6	14	**3.46**	0.5	27	**3.46**	1.4	12	**0.6**	0.6	35	0.47	1.8	19
Sweden	0.6	1.0	33	**0.62**	1.6	13	2.49	0.8	26	**2.49**	1.4	12	0.33	1.2	33	**0.33**	1.5	13

The numerical test is repeated with $f = \sinh$ and the corresponding results are reported in Table 6. On all case studies KRYLOV_LBFGS and KRYLOV_HESSIAN yields almost equal variations of the f-connectivity. The most significant differences are observed on England (Addition), Germany (Rewiring), Mexico (Addition), and Poland (Tuning). Similar comments to the exponential case apply to the reported computational times.

7. CONCLUSIONS

We have proposed two strategies, based on Krylov subspace approximations, for optimizing the natural connectivity of a graph. The first one is a greedy heuristic method that is well suited to contexts where we either add or remove unweighted edges on a large-scale graph. Despite being computationally more expensive than state-of-the-art alternatives, in the context of the addition problem our approach significantly outperforms the increase of the natural connectivity. The second proposed strategy combines Krylov subspace approximation and an interior point scheme using either the Hessian or its L-BFGS approximation, to address continuous

TABLE 6. Magnitude of the relative trace variation (ΔT), execution time in seconds (t), and number of iterations (k_{it}), for the tuning, rewiring, and addition optimization problems solved with KRYLOV_LBFGS and KRYLOV_HESSIAN approaches, for weighted graphs associated with power grid networks and for $f = \sinh$.

							Hyperbolic sine											
	TUNING						REWIRING						ADDITION					
	LBFGS			HESSIAN			LBFGS			HESSIAN			LBFGS			HESSIAN		
	ΔT	t	k_{it}	ΔT	t	k_{it}	ΔT	t	k_{it}	ΔT	t	k_{it}	ΔT	t	k_{it}	ΔT	t	k_{it}
Austria	**13.98**	0.9	41	13.9	1.4	9	**67.14**	0.4	26	67.14	1.4	10	19.85	0.3	15	**19.85**	1.0	9
Denmark	**15.51**	0.3	38	15.5	1.1	11	123.74	0.2	23	**123.75**	1.3	11	16.72	0.1	19	**16.74**	1.0	8
England	**3.68**	10.0	40	3.68	6.0	13	**9.7**	4.2	35	9.7	2.9	10	3.29	2.0	20	**3.67**	2.2	10
Germany	2.72	3.0	12	**2.74**	2.1	7	9.26	2.6	39	**9.87**	2.1	11	1.19	1.9	24	**1.32**	1.8	10
India	**6.86**	6.8	24	6.86	4.9	12	**32.21**	5.0	29	32.17	3.2	9	5.26	3.5	24	**5.26**	2.7	9
Italy	5.48	11.1	47	**5.54**	4.1	12	19.46	3.2	20	**19.46**	3.2	10	2.8	2.1	30	**2.8**	2.4	10
Mexico	**5.02**	4.9	25	5.01	3.0	9	**12.75**	7.4	33	12.72	3.1	9	2.05	3.6	35	**2.6**	3.2	15
Poland	6.23	1.4	17	**6.44**	1.7	9	14.45	2.7	28	**14.46**	2.2	10	**4.05**	3.2	27	4.0	1.5	7
Portugal	9.67	1.1	27	**9.68**	2.6	10	40.63	0.8	24	**41.3**	1.3	10	**6.41**	0.9	29	6.4	1.5	12
Sweden	**6.9**	1.7	27	6.9	7.2	34	**43.36**	1.8	24	43.09	1.6	8	**5.27**	1.7	25	5.27	1.5	8

optimization problems that include edge tuning and rewiring. To the best of our knowledge, this is the first attempt to tackle the optimization of the natural connectivity with first and second order methods, and the reported experiments demonstrate the feasibility of the approach at least for graphs up to medium size.

Finally, we highlight that the proposed computational strategies are quite flexible as they can be adapted with minor changes to the optimization of other matrix function based measures on graphs and it is conceptually easy to incorporate further constraints on the set of modifiable edges.

REFERENCES

[1] A.H. Al-Mohy and N.J. Higham, Computing the action of the matrix exponential, with an application to exponential integrators. *SIAM J. Sci. Comput.* **33** (2011) 488–511.

[2] H. Alqahtani and L. Reichel, Multiple orthogonal polynomials applied to matrix function evaluation. *BIT Numer. Math.* **58** (2018) 835–849.

[3] F. Arrigo and M. Benzi, Updating and downdating techniques for optimizing network communicability. *SIAM J. Sci. Comput.* **38** (2016) B25–B49.

[4] V. Batagelj and A. Mrvar, Pajek datasets collection. http://vlado.fmf.uni-lj.si/pub/networks/data/ (2006).

[5] B. Beckermann, D. Kressner and M. Schweitzer, Low-rank updates of matrix functions. *SIAM J. Matrix Anal. Appl.* **39** (2018) 539–565.

[6] M. Bellalij, L. Reichel, G. Rodriguez and H. Sadok, Bounding matrix functionals via partial global block lanczos decomposition. *Appl. Numer. Math.* **94** (2015) 127–139.

[7] R.H. Byrd, M.E. Hribar and J. Nocedal, An interior point algorithm for large-scale nonlinear programming. *SIAM J. Optim.* **9** (1999) 877–900.

[8] R.H. Byrd, J.C. Gilbert and J. Nocedal, A trust region method based on interior point techniques for nonlinear programming. *Math. Program.* **89** (2000) 149–185.

[9] H. Chan and L. Akoglu, Optimizing network robustness by edge rewiring: a general framework. *Data Min. Knowl. Discov.* **30** (2016) 1395–1425.

[10] H. Chan, L. Akoglu and H. Tong, Make it or break it: Manipulating robustness in large networks. In *Proceedings of the 2014 SIAM International Conference on Data Mining.* SIAM (2014) 325–333.

[11] T. Chen, A. Greenbaum, C. Musco and C. Musco, Error bounds for lanczos-based matrix function approximation. *SIAM J. Matrix Anal. Appl.* **43** (2022) 787–811.

[12] F. Chung, F.R. Chung, F.C. Graham and L. Lu, *Complex Graphs and Networks.* American Mathematical Soc., Number 107 (2006).

[13] S. Cipolla, F. Durastante and F. Tudisco, Nonlocal pagerank. *ESAIM Math. Model. Numer. Anal.* **55** (2021) 77–97.

[14] ComplexNetTSP PowerGrids, Highvoltage power grid networks. https://github.com/ComplexNetTSP/Power_grids/tree/v1.0.0 (2023).

[15] A. Cortinovis, D. Kressner and S. Massei, Divide-and-conquer methods for functions of matrices with banded or hierarchical low-rank structure. *SIAM J. Matrix Anal. Appl.* **43** (2022) 151–177.

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

[17] G. D'Angelo, M. Olsen and L. Severini, Coverage centrality maximization in undirected networks. In Vol. 33 *Proceedings of the AAAI Conference on Artificial Intelligence* (2019) 501–508.

[18] O. De la Cruz Cabrera, J. Jin, S. Noschese and L. Reichel, Communication in complex networks. *Appl. Numer. Math.* **172** (2022) 186–205.

[19] E. Estrada, Virtual identification of essential proteins within the protein interaction network of yeast. *Proteomics* **6** (2006) 35–40.

[20] E. Estrada and N. Hatano, Statistical-mechanical approach to subgraph centrality in complex networks. *Chem. Phys. Lett.* **439** (2007) 247–251.

[21] E. Estrada and Ö. Bodin, Using network centrality measures to manage landscape connectivity. *Ecol. Appl.* **18** (2008) 1810–1825.

[22] E. Estrada and N. Hatano, Communicability in complex networks. *Phys. Rev. E* **77** (2008) 036111.

[23] E. Estrada and N. Hatano, Returnability in complex directed networks (digraphs). *Linear Algebra Appl.* **430** (2009) 1886–1896.

[24] E. Estrada and D.J. Higham, Network properties revealed through matrix functions. *SIAM Rev.* **52** (2010) 696–714.

[25] E. Estrada and P.A. Knight, *A First Course in Network Theory.* Oxford University Press, USA (2015).

[26] C. Fenu, D. Martin, L. Reichel and G. Rodriguez, Block Gauss and anti-Gauss quadrature with application to networks. *SIAM J. Matrix Anal. Appl.* **34** (2013) 1655–1684.

[27] P. Fika and M. Mitrouli, Aitken's method for estimating bilinear forms arising in applications. *Calcolo* **54** (2017) 455–470.

[28] T.N. for Research Core Team, https://github.com/bstabler/TransportationNetworks (2023).

[29] A. Frommer, K. Lund and D.B. Szyld, Block krylov subspace methods for functions of matrices. *Electron. Trans. Numer. Anal.* **47** (2017) 100–126.

[30] K. Garimella, G. De Francisci Morales, A. Gionis and M. Mathioudakis, Reducing controversy by connecting opposing views. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining.* ACM (2017) 81–90.

[31] A. Ghosh and S. Boyd, Growing well-connected graphs. In *Proceedings of the 45th IEEE Conference on Decision and Control.* IEEE (2006) 6605–6611.

[32] A. Ghosh, S. Boyd and A. Saberi, Minimizing effective resistance of a graph. *SIAM Rev.* **50** (2008) 37–66.

[33] D.F. Gleich, Pagerank beyond the web. *SIAM Rev.* **57** (2015) 321–363.

[34] P. Grindrod and T. Lee, Comparison of social structures within cities of very different sizes. *R. Soc. Open Sci.* **3** (2016) 150526.

[35] M.H. Gutknecht, Block Krylov space methods for linear systems with multiple right-hand sides: An introduction, edited by A.H. Siddiqi, I.S. Duff and O. Christensen. In *Modern Mathematical Models, Methods and Algorithms for Real World Systems.* New Delhi, Anamaya (2007) 420–447.

[36] N. Hale, N.J. Higham and L.N. Trefethen, Computing A^α, $\log(A)$, and related matrix functions by contour integrals. *SIAM J. Numer. Anal.* **46** (2008) 2505–2523.

[37] P. Kandolf, A. Koskela, S.D. Relton and M. Schweitzer, Computing low-rank approximations of the Fréchet derivative of a matrix function using Krylov subspace methods. *Numer. Linear Algebra Appl.* **28** (2021) e2401.

[38] D. Kressner, A Krylov subspace method for the approximation of bivariate matrix functions. In *Structured Matrices in Numerical Linear Algebra*, Vol. 30 of *Springer INdAM Series.* Springer, Cham (2019) 197–214.

[39] L.T. Le, T. Eliassi-Rad and H. Tong, Met: A fast algorithm for minimizing propagation in large graphs with small eigen-gaps. In *Proceedings of the 2015 SIAM International Conference on Data Mining.* SIAM (2015) 694–702.

[40] U. Luxburg, A. Radl and M. Hein, Getting lost in space: Large sample analysis of the resistance distance. *Adv. Neural Inf. Process. Syst.* **23** (2010).

[41] S. Massei and F. Tudisco, Matlab code for "Optimizing network robustness via Krylov subspaces". https://github.com/COMPiLELab/krylov_robustness (2023).

[42] R. Mathias, A chain rule for matrix functions and applications. *SIAM J. Matrix Anal. Appl.* **17** (1996) 610–620.

[43] S. Medya, A. Silva, A. Singh, P. Basu and A. Swami, Group centrality maximization via network design. In *Proceedings of the 2018 SIAM International Conference on Data Mining.* SIAM (2018) 126–134.

[44] R.A. Meyer, C. Musco, C. Musco and D.P. Woodruff, Hutch++: optimal stochastic trace estimation. In *Symposium on Simplicity in Algorithms (SOSA)*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA (2021) 142–155.

[45] V. Nicosia, R. Criado, M. Romance, G. Russo and V. Latora, Controlling centrality in complex networks. *Sci. Rep.* **2** (2012) 218.

[46] J. Nocedal and S.J. Wright, *Numerical Optimization*. Springer (1999).

[47] S. Pozza and F. Tudisco, On the stability of network indices defined by means of matrix functions. *SIAM J. Matrix Anal. Appl.* **39** (2018) 1521–1546.

[48] S. Saha, A. Adiga, B.A. Prakash and A.K.S. Vullikanti, Approximation algorithms for reducing the spectral radius to control epidemic spread. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM (2015) 568–576.

[49] M. Schweitzer, Sensitivity of matrix function based network communicability measures: Computational methods and a priori bounds. Preprint: `arXiv:2303.01339` (2023).

[50] S.N.A.P. (SNAP), sparse networks collection. `http://snap.stanford.edu/data/index.html` (2023).

[51] H. Tong, B.A. Prakash, T. Eliassi-Rad, M. Faloutsos and C. Faloutsos, Gelling, and melting, large graphs by edge manipulation. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. ACM (2012) 245–254.

[52] F. Tudisco and D.J. Higham, Node and edge nonlinear eigenvector centrality for hypergraphs. *Commun. Phys.* **4** (2021) 201.

[53] P. Van Mieghem, D. Stevanović, F. Kuipers, C. Li, R. Van De Bovenkamp, D. Liu and H. Wang, Decreasing the spectral radius of a graph by link removals. *Phys. Rev. E* **84** (2011) 016101.

[54] S. Vigna, Spectral ranking. *Netw. Sci.* **4** (2016) 433–445.

[55] S. Wasserman and K. Faust, Social Network Analysis: Methods and Applications (1994).

[56] Z. Yu, C. Wang, J. Bu, X. Wang, Y. Wu and C. Chen, Friend recommendation with content spread enhancement in social networks. *Inf. Sci.* **309** (2015) 102–118.

[57] Y. Zhang, A. Adiga, A. Vullikanti and B.A. Prakash, Controlling propagation at group scale on networks. In *2015 IEEE International Conference on Data Mining*. IEEE (2015) 619–628.