Improved Large-Scale Graph Learning through Ridge Spectral Sparsification

Daniele Calandriello [1 2] **Ioannis Koutis** [3] **Alessandro Lazaric** [4] **Michal Valko** [1]

Abstract

The representation and learning benefits of methods based on graph Laplacians, such as *Laplacian smoothing* or *harmonic function solution for semi-supervised learning* (SSL), are empirically and theoretically well supported. Nonetheless, the exact versions of these methods scale poorly with the number of nodes n of the graph. In this paper, we combine a spectral sparsification routine with Laplacian learning. Given a graph $\mathcal{G}$ as input, our algorithm computes a sparsifier in a *distributed* way in $\mathcal{O}(n\log^3(n))$ time, $\mathcal{O}(m\log^3(n))$ work and $\mathcal{O}(n\log(n))$ memory, using only $\log(n)$ rounds of communication. Furthermore, motivated by the regularization often employed in learning algorithms, we show that constructing sparsifiers that preserve the spectrum of the Laplacian *only up to* the regularization level may drastically reduce the size of the final graph. By constructing a spectrally-similar graph, we are able to bound the error induced by the sparsification for a variety of downstream tasks (e.g., SSL). We empirically validate the theoretical guarantees on Amazon co-purchase graph and compare to the state-of-the-art heuristics.

1. Introduction

Graphs are a very effective data structure to represent relationships between entities (e.g., social and collaboration networks, influence graphs). Over the years, many machine learning problems have been defined and solved exploiting the graph representation, such as *graph-regularized least squares* (LapRLS, Belkin et al. 2005), *Laplacian smoothing* (LapSmo, Sadhanala et al. 2016) graph *semi-supervised learning* (SSL, Chapelle et al. 2010; Zhu et al. 2003), *laplacian embedding* (LE, Belkin & Niyogi 2001, and *spectral clustering* (SC, Von Luxburg 2007). The intuition behind graph-based learning is that the information expressed by the graph helps to capture the underlying structure of the problem (e.g., a manifold), thus improving the learning. For instance, LapSmo and SSL rely on the assumption that nodes that are *close* in the graph are more likely to have similar labels. Similarly, LE and SC try to find a low-dimensional representation of the nodes using the eigenvectors of the Laplacian of the graph. In general, given a graph $\mathcal{G}$ of n nodes and m edges, most of graph-based learning tasks require computing the minimum of a cost function based on the associated $n \times n$ Laplacian matrix $\mathbf{L}_\mathcal{G}$, which contains m non-zero entries. Solving *exactly* such optimization problems amounts to $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space complexity in the worst case and they become infeasible even for mildly large/dense graphs.

A complete review of the literature on large-scale graph learning is beyond the scope of this paper and we only consider methods that reduce learning space and time complexity starting from a given graph received as input.[1] We identify mainly three possible approaches. We can (1) reduce runtime replacing the pseudo-inverse operator $\mathbf{L}_\mathcal{G}^+$ with an *iterative solver*, (2) reduce time and space complexity replacing the large graph $\mathcal{G}$ with a sparser approximation $\mathcal{H}$, or (3) reduce runtime and increase memory capacity by *distributing* the computation across multiple machines.

Iterative solvers. Iterative methods can solve a number of learning problems without explicitly constructing $\mathbf{L}_\mathcal{G}^+$ (e.g., gradient descent, GD, for LapSmo, iterative averaging for SSL, and the power method for SC). In this case we only need $\mathcal{O}(m)$ time per iteration. Unfortunately, all simple iterative methods (e.g., GD) converge in a number of iterations proportional to the condition number of the Laplacian, $\kappa = \lambda_{\max}(\mathbf{L}_\mathcal{G})/\lambda_{\min}(\mathbf{L}_\mathcal{G})$, which may grow linearly with the number of nodes n, thus removing the advantage of the iterative method, whose complexity tends to $\mathcal{O}(n^3)$ in the worst case. Advanced iterative methods, such as the *preconditioned conjugate gradient,* use preconditioning to find an accurate solution in a number of iterations independent of κ. Koutis et al. (2011) gives a nearly-linear solver for Laplacians or *strongly diagonally dominant* (SDD) matri-

[1]SequeL team, INRIA Lille - Nord Europe, France [2]LCSL, IIT, Italy, and MIT, USA. [3]New Jersey Institute of Technology, USA [4]Facebook AI Research, Paris, France. Correspondence to: Daniele Calandriello <daniele.calandriello@iit.it>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

[1]Many algorithms reduce the complexity of graph learning *at construction* time but they cannot be applied to *natural* graphs (e.g., social graphs) and therefore we do not review them.

ces, that using a chain of preconditioners, converges in only $\mathcal{O}(m\log(n))$ time. As space and time costs scale with the number of edges, a natural desire is to reduce m by sparsifying and distributing the graph.

Graph sparsification. The objective of sparsification methods is to remove *redundant* edges, so that the resulting sparse sub-graph can be easily stored in memory and efficiently manipulated to compute final solutions. A simple *graph-sparsification* technique is to sample $n\overline{q}$ (with $\overline{q} > 1$) edges from $\mathcal{G}$ with probabilities proportional to the edge weights with replacement. While computationally very efficient, uniform sampling requires sampling a number of edges proportional to $\mathcal{O}(n\mu(\mathcal{G}))$ (i.e., $\overline{q} \propto \mu(\mathcal{G})$), where $\mu(\mathcal{G})$ is the *coherence* of the Laplacian matrix, and it can grow as large as n when the graph is highly structured (e.g., if there is a single edge e connecting two components of the graph we need to sample all of the edges of the graph—potentially $\mathcal{O}(n^2)$—to guarantee that we do not exclude e and generate an inappropriate $\mathcal{H}$). A more refined approach is the k-neighbors (`kN`) sparsifier (Sadhanala et al., 2016), which performs local sparsifications node-by-node by keeping all edges at nodes with degree smaller than $\overline{q}$, and samples them proportionally to their weights whenever the degree is bigger than $\overline{q}$. While in certain structured graphs, this method may perform much better than uniform (Von Luxburg et al., 2014), in the general case $\overline{q}$, still needs to scale with the coherence $\mu(\mathcal{G})$. A more effective method is to sample edges proportionally to their *effective resistance*, which intuitively measures the importance of an edge in preserving the minimum distance between two nodes. As a result, only relevant edges are kept and the sparsified graph could be reduced to $\mathcal{O}(n\,\mathrm{polylog}(n))$ edges. Nonetheless, computing effective resistances also requires the pseudo-inverse $\mathbf{L}_{\mathcal{G}}^{+}$, thus being as expensive as solving any graph-Laplacian learning problem.

Distributed computing. When the number of edges m is too large to fit the whole graph in a single machine, we are forced to distribute the edges across multiple machines. At the same time, if the sparsifier construction or the downstream inference can be parallelized, we can also reduce their runtime. Unfortunately, distributing data and computation across multiple machines can cause large communication costs. For example, simple `GD` or label propagation methods require $\mathcal{O}(\kappa)$ iterations (and communication rounds) to converge and access to non-local (e.g., neighbors in a graph) data. While preconditioned solvers reduce the number of iterations, almost none of their memory access is local, thus making difficult to have efficient distributed implementations.

Contribution. In this paper, we propose a new approach that aims at integrating the benefits of the three different methods above. Using the large memory and computational capacity of distributed computing and leveraging the sequential sparsification methods of Kelner & Levin (2013) and Calandriello et al. (2017), we show how to compute an accurate sparsifier $\mathcal{H}$ of graph $\mathcal{G}$ in $\mathcal{O}(n\log^3(n))$ time, $\mathcal{O}(n\log^2(n))$ work and $\mathcal{O}(n\log(n))$ memory, using only $\log(n)$ rounds of communication. Afterwards, learning tasks can be solved directly on $\mathbf{L}_{\mathcal{H}}$ on a single machine using near-linear time solvers, resulting in an overall $\mathcal{O}(n\log^3(n))$ runtime. Moreover, we show that the regularization used in some graph-based learning algorithms allows using even sparser graphs. In particular, we introduce the notion of *ridge* effective resistance to obtain sparsifiers that are better adapted to solve Laplacian-regularized learning tasks (e.g., LapSmo, SSL) and are smaller than standard spectral sparsifiers without compromising the performance of downstream tasks.

2. Background

We use lowercase letters a for scalars, bold lowercase letters $\mathbf{a}$ for vectors and uppercase bold letters $\mathbf{A}$ for matrices. We use $\mathbf{A} \preceq \mathbf{B}$ to denote that $\mathbf{B} - \mathbf{A}$ is positive semi-definite (PSD), $[\mathbf{A}]_{i,j}$ to indicate the (i,j)-th entry of $\mathbf{A}$, and ordered the eigenvalues as $\lambda_1(\mathbf{A}) \leq \ldots \leq \lambda_n(\mathbf{A})$.

2.1. Graphs and graph Laplacian

We denote with $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, an undirected weighted graph with n nodes $\mathcal{V}$ and m edges $\mathcal{E}$. Each edge $e_{i,j} \in \mathcal{E}$ has a weight $a_{e_{i,j}}$ measuring the "similarity" between nodes i and j. Given graphs $\mathcal{G}$ and $\mathcal{G}'$ over the same set of nodes $\mathcal{V}$, $\mathcal{G} + \mathcal{G}'$ denotes the graph obtained by summing the weights of their edges. For graph $\mathcal{G}$, we introduce the weighted adjacency matrix $\mathbf{A}_{\mathcal{G}}$ with entries $[\mathbf{A}_{\mathcal{G}}]_{i,j} = a_{e_{i,j}}$, the total weights $A = \sum_e a_e$, and the diagonal degree matrix $\mathbf{D}_{\mathcal{G}}$ with entries $[\mathbf{D}_{\mathcal{G}}]_{i,i} \triangleq \sum_j a_{e_{i,j}}$. The Laplacian of $\mathcal{G}$ is the PSD matrix $\mathbf{L}_{\mathcal{G}} \triangleq \mathbf{D}_{\mathcal{G}} - \mathbf{A}_{\mathcal{G}}$. Furthermore, we assume that $\mathcal{G}$ is connected and thus $\mathbf{L}_{\mathcal{G}}$ has only one eigenvalue equal to 0 and $\mathrm{Ker}(\mathbf{L}_{\mathcal{G}}) = \mathbf{1}$. Let $\mathbf{L}_{\mathcal{G}}^{+}$ be the pseudoinverse of $\mathbf{L}_{\mathcal{G}}$ and $\mathbf{L}_{\mathcal{G}}^{-1/2} = (\mathbf{L}_{\mathcal{G}}^{+})^{1/2}$. For any node $i = 1, \ldots, n$, we denote with $\chi_i \in \mathbb{R}^n$, the indicator vector, so that $\mathbf{b}_e \triangleq \sqrt{a_e}(\chi_i - \chi_j)$ is the "edge" vector. If we denote with $\mathbf{B}_{\mathcal{G}}$ the $m \times n$ signed edge-vertex incidence matrix, then the Laplacian can be written as $\mathbf{L}_{\mathcal{G}} = \sum_e \mathbf{b}_e \mathbf{b}_e^{\mathsf{T}} = \mathbf{B}_{\mathcal{G}}^{\mathsf{T}} \mathbf{B}_{\mathcal{G}}$.

2.2. Learning on graphs

Given graph $\mathcal{G}$ and its Laplacian $\mathbf{L}_{\mathcal{G}}$, we denote with $\mathbf{f} \in \mathbb{R}^n$, a *labeling* of its nodes, where $[\mathbf{f}]_i$ is the value associated with the i-th node. Many graph learning algorithms assume that the optimal labeling $\mathbf{f}^\star$ is *smooth* w.r.t. the graph, i.e., the quantity $\sum_e a_e([\mathbf{f}^\star]_{e_i} - [\mathbf{f}^\star]_{e_j})^2 = \mathbf{f}^{\star\mathsf{T}} \mathbf{L}_{\mathcal{G}} \mathbf{f}^\star$ is small. In the following, we review examples from the supervised, semi-supervised and unsupervised learning with graphs.

Laplacian smoothing (LapSmo) with Gaussian noise. Given a graph $\mathcal{G}$ on n nodes, let $\mathbf{y} \triangleq \mathbf{f}^\star + \xi$ be a noisy

measurement of $\mathbf{f}^\star$ with $[\xi]_i \sim \mathcal{N}(0, \sigma^2)$. The goal of LAPSMO is to find a vector $\widehat{\mathbf{f}}$ that accurately reconstructs $\mathbf{f}^\star$ under the graph smoothness assumption by solving

$$\widehat{\mathbf{f}} \triangleq \arg\min_{\mathbf{f}\in\mathbb{R}^n} (\mathbf{f}-\mathbf{y})^\top(\mathbf{f}-\mathbf{y}) + \lambda \mathbf{f}^\top \mathbf{L}_\mathcal{G}\mathbf{f} = (\lambda \mathbf{L}_\mathcal{G} + \mathbf{I})^{-1}\mathbf{y}, \quad (1)$$

where λ is a regularization parameter.

Graph semi-supervised learning (SSL). In SSL, the input $\mathbf{f}_\ell$ is a partial observation of the labels $\mathbf{f}^\star$ for a subset $\mathcal{S} \subset [n]$ of nodes. The goal is to predict the labels $\mathbf{f}_u$ of the unrevealed nodes. The *harmonic function solution* (HFS) by Zhu et al. (2003) solves the optimization problem

$$\widehat{\mathbf{f}}_{\text{HFS}} \triangleq \arg\min_{\mathbf{f}\in\mathbb{R}^n} \tfrac{1}{\ell}(\mathbf{f}-\mathbf{y})^\top \ell_\mathcal{S}(\mathbf{f}-\mathbf{y}) + \lambda \mathbf{f}^\top \mathbf{L}_\mathcal{G}\mathbf{f} = (\lambda \ell \mathbf{L}_\mathcal{G} + \mathbf{I}_\mathcal{S})^{+}\mathbf{y}_\mathcal{S}, \quad (2)$$

where $\ell \triangleq |\mathcal{S}|$ is the number of labeled nodes received as input, $\mathbf{I}_\mathcal{S} \in \mathbb{R}^{n\times n}$ is the identity matrix with zeros at nodes not in $\mathcal{S}$, and $\mathbf{y}_\mathcal{S} \triangleq \mathbf{I}_\mathcal{S}\mathbf{y} \in \mathbb{R}^n$. Similarly, in *local transductive regression* (LTR) (Cortes et al., 2008), the optimization problem is

$$\widehat{\mathbf{f}}_{\text{LTR}} \triangleq \arg\min_{\mathbf{f}\in\mathbb{R}^n} (\mathbf{f}-\mathbf{y})^\top \mathbf{C}(\mathbf{f}-\mathbf{y}) + \mathbf{f}^\top(\mathbf{L}_\mathcal{G} + \lambda\mathbf{I})\mathbf{f} = (\mathbf{C}^{-1}(\mathbf{L}_\mathcal{G} + \lambda\mathbf{I}) + \mathbf{I})^{-1}\mathbf{y}_\mathcal{S}, \quad (3)$$

where $\mathbf{C}$ is a diagonal matrix with entries c_ℓ for nodes in $\mathcal{S}$, c_u for entries not in $\mathcal{S}$, and $c_\ell \geq c_u > 0$.

Spectral clustering (SC). Applying the Laplacian smoothness assumption, the goal of SC is to find k disjoint subset assignments such that the clusters are smooth w.r.t. the Laplacian. Let $\{\mathbf{f}_c\}_{c=1}^k$ be the cluster indicator vectors such that $[\mathbf{f}_c]_i \triangleq 1$ if node i is in the c-th cluster and $[\mathbf{f}_c]_i \triangleq 0$ otherwise. Denote with $\mathbf{F} \in \mathbb{R}^{n\times k}$, the matrix containing the assignments, and let $\mathcal{C}$ be the space of feasible clustering, such that all $\mathbf{f}_c$ are binary and each row of $\mathbf{F}$ contains only one non-zero entry. Since computing the minimum ratio-cut is NP-hard (Von Luxburg, 2007; Lee et al., 2014), even under constraints (Cucuringu et al., 2016), SC defines instead the relaxed problem

$$\widehat{\mathbf{F}} \triangleq \arg\min_{\mathbf{F}:\mathbf{F}^\top\mathbf{F}=\mathbf{I}_k, \mathbf{f}_c\perp\mathbf{1}} \operatorname{Tr}(\mathbf{F}^\top\mathbf{L}\mathbf{F}).$$

Once the relaxed solution is computed, we can use different heuristics to recover the clustering, such as thresholding or performing a k-means clustering on the $\widehat{\mathbf{F}}$ matrix.

Computational complexity. The problems above require either to compute an eigendecomposition of the Laplacian $\mathbf{L}_\mathcal{G}$ or to solve a linear system involving $\mathbf{L}_\mathcal{G}$. Computing these exactly is not feasible when the number of nodes n and edges m grows. In particular, (a) storing $\mathbf{L}_\mathcal{G}$ in memory requires $\mathcal{O}(m)$ space, and it is not feasible when m is large, (b) even if $\mathbf{L}_\mathcal{G}$ is sparse and m is small, the pseudo-inverse $\mathbf{L}_\mathcal{G}^+$ might be dense, and thus computing and storing $\mathbf{L}_\mathcal{G}^+$ exactly requires up to $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ space.

3. Distributed Spectral Sparsification

In this section, we describe a new, sequential, distributed, and efficient algorithm for graph sparsification that can be used as a preprocessing step to solve a large variety of downstream learning tasks, without significantly affecting their performance. We point out that while distributing data-agnostic sparsifiers (e.g. uniform sampling) is straightforward, distributing the computation of sparsifiers based on effective resistances requires a careful merging procedure to guarantee satisfactory memory vs. accuracy tradeoff, which is what we provide in this section.

3.1. (ε, γ)-spectral sparsifiers

We start with the introduction of the notion of (ε, γ)-sparsifier that is adapted for the learning tasks that use sparsified graph Laplacian.

Definition 1. *A (ε, γ)-spectral sparsifier of $\mathcal{G}$ is a reweighted sub-graph $\mathcal{H} \subseteq \mathcal{G}$ whose Laplacian $\mathbf{L}_\mathcal{H}$ satisfies*

$$(1-\varepsilon)\mathbf{L}_\mathcal{G} - \varepsilon\gamma\mathbf{I} \preceq \mathbf{L}_\mathcal{H} \preceq (1+\varepsilon)\mathbf{L}_\mathcal{G} + \varepsilon\gamma\mathbf{I}. \quad (4)$$

For $\gamma = 0$, this definition reduces to the standard notion of ε-spectral sparsifier (Spielman & Teng, 2011). The main difference is that an (ε, γ)-spectral sparsifier allows for an extra *additive* error of order $\varepsilon\gamma$. This change is directly motivated by the fact that the sparsifier $\mathcal{H}$ may be used in learning tasks whose solution may not be sensitive to small (additive) errors. As a result, (ε, γ)-spectral sparsifiers are able to further reduce the size of $\mathcal{H}$ w.r.t. $(\varepsilon, 0)$-sparsifiers, without significantly affecting the final learning performance. Formally, an ε-sparsifier preserves all the quadratic forms up to a small multiplicative (constant) error, and thus can be used to provide an accurate approximation to many important quantities such as graph cuts or eigenvalues. In fact, for all $i \in [n]$, an ε-sparsifier guarantees that $(1-\varepsilon)\lambda_i(\mathbf{L}_\mathcal{G}) \leq \lambda_i(\mathbf{L}_\mathcal{H}) \leq (1+\varepsilon)\lambda_i(\mathbf{L}_\mathcal{G})$. Nonetheless, in many learning tasks (e.g., LTR) the noise level in the signal $\mathbf{f}$ requires regularizing the solution so that the Laplacian $\mathbf{L}_\mathcal{G}$ itself is eventually replaced by $\mathbf{L}_\mathcal{G} + \lambda\mathbf{I}$ (e.g., Eq. 1). This corresponds to *soft-thresholding* the eigenvalues of the Laplacian, so that eigenvalues below λ are partially ignored. If λ is properly tuned w.r.t. the noise, the regularization increases stability and improves the learning performance. Therefore, constructing a sparsifier that accurately reconstructs *all* eigenvalues of $\mathbf{L}_\mathcal{G}$ may be wasteful, as it may require keeping most of the edges. As a result, in tasks where $\mathbf{L}_\mathcal{G}$ is regularized, it is better to use (ε, γ)-sparsifiers, as their additive error $\gamma\mathbf{I}$ is homogeneous with the regu-

larization $\lambda\mathbf{I}$ and their smaller size allows scaling to up.[2] We now extend the results of Spielman & Srivastava (2011) for the construction of ε-spectral sparsifiers to the general case of (ε,γ)-sparsifiers. We redefine the edge effective resistance to account for the regularization.

Definition 2. *The γ-effective resistance of an edge e in graph $\mathcal{G}$ is defined as*

$$r_e(\gamma) \triangleq \mathbf{b}_e^\mathsf{T}\left(\mathbf{L}_\mathcal{G}+\gamma\mathbf{I}\right)^{-1}\mathbf{b}_e. \tag{5}$$

The "effective dimension" of the graph is the total sum of the γ-effective resistances, $d_{\mathit{eff}}(\gamma) \triangleq \sum_e r_e(\gamma)$.

We can now construct a sparsifier $\mathcal{H}$ by sampling $\overline{q}$ times each edge with a probability proportional to its γ-effective resistance. More formally, the resulting (random) graph contains $q_e \sim \mathcal{B}(r_e(\lambda);\overline{q})$ copies of each edge, where $\mathcal{B}$ is the Binomial distribution, and its associated Laplacian is $\mathbf{L}_\mathcal{H} = \sum_{e\in\mathcal{H}} q_e/(\overline{q}r_e(\gamma))\mathbf{b}_e\mathbf{b}_e^\mathsf{T}$, which is an unbiased estimator of $\mathbf{L}_\mathcal{G}$. We can then apply existing results from sketching of PSD matrices (Alaoui & Mahoney, 2015) to prove that $\mathcal{H}$ is a valid (ε,γ)-sparsifier.

Proposition 1 (Cohen et al. 2017)**.** *Let $\varepsilon>0$ and $\gamma\geq 0$ be the accuracy parameters and $0\leq\delta\leq 1$ the probability of error. Let $\mathcal{H}$ be the graph obtained by sampling edges in $\mathcal{G}$ with a probability proportional to their γ-effective resistances. If $\overline{q}\geq 4\log(4n/\delta)/\varepsilon^2$, then w.p. $1-\delta$, $\mathcal{H}$ is an (ε,γ)-sparsifier with $\mathcal{O}(d_{\mathit{eff}}(\gamma)\overline{q})$ edges.*

We first notice that this result reduces to the one of Spielman & Srivastava (2011) for $\gamma=0$. In fact, $d_{\mathrm{eff}}(0)=n-1$ for all graphs, thus matching the space requirement $\overline{q}$ for ε-sparsifiers. Nonetheless, as γ increases, the size of $\mathcal{H}$ reduces significantly. Using $\mathbf{L}_\mathcal{G}=\mathbf{B}_\mathcal{G}^\mathsf{T}\mathbf{B}_\mathcal{G}$, the effective dimension $d_{\mathrm{eff}}(\gamma)$ can be conveniently rewritten as

$$d_{\mathrm{eff}}(\gamma)=\mathrm{Tr}\left(\mathbf{B}_\mathcal{G}^\mathsf{T}\mathbf{B}_\mathcal{G}(\mathbf{B}_\mathcal{G}^\mathsf{T}\mathbf{B}_\mathcal{G}+\gamma\mathbf{I})^{-1}\right)=\sum_{i=2}^{n}\frac{\lambda_i(\mathbf{L}_\mathcal{G})}{\lambda_i(\mathbf{L}_\mathcal{G})+\gamma},$$

thus showing that $d_{\mathrm{eff}}(\gamma)$ is the "soft" rank of the Laplacian, where γ significantly reduces the contribution of small eigenvalues to the total sum. While in the worst case $d_{\mathrm{eff}}(\gamma)$ can be as large as $n-1$, for a variety of graphs with rapidly decaying spectrum (Jamakovic & Mieghem, 2006; Samukhin et al., 2008; Zhan et al., 2010; Akoglu et al., 2015), $d_{\mathrm{eff}}(\gamma)$ may be significantly smaller than $n-1$, thus reducing the number of edges $\overline{q}$ required to obtain an (ε,γ)-sparsifier.

3.2. The algorithm

As pointed out in the introduction, the main limitation of effective-resistance-based sparsification is that the computation of r_e requires inverting the Laplacian matrix, thus resulting in a computational cost that already matches the cost of the learning tasks themselves. Moreover, large graphs cannot be stored in memory, and multiple passes over the graph would result in a disk access overhead larger than the computational cost. In order to avoid these problems, we adapt our previous work (Calandriello et al., 2017) in online sparsification and randomized linear algebra (see a thorough discussion and comparison at the end of the section) to obtain the distributed sequential resparsification (DiSRe) algorithm (Alg. 1).[3]

Algorithm 1 The DiSRe algorithm.

Input: $\mathcal{G}$
Output: $\mathcal{H}_\mathcal{G}$

```
Partition G into k sub-graphs:
    H_{1,ℓ} ← G_ℓ ← {(e_{i,j}, q_e = 1, p̃_{1,e} = 1)}
Initialize set S_1 = {H_{1,ℓ}}_{ℓ=1}^k
for h = 1, ..., k − 1 do
    Pick two sparsifiers H_{h,i'}, H_{h,i'} from S_h
    H̄ ← Merge-Resparsify(H_{h,i}, H_{h,i'})
    Place H̄ back into S_{h+1}
end for
Return H_G, the last sparsifier in S_k
```

Algorithm 2 Merge-Resparsify

Require: (ε,γ)-sparsifiers $\mathcal{H}_{h,i},\mathcal{H}_{h,i'}$ of graphs $\mathcal{G}_{h,i},\mathcal{G}_{h,i'}$
Ensure: $\overline{\mathcal{H}}$, an (ε,γ) sparsifier of $\mathcal{G}_{h,i}+\mathcal{G}_{h,i'}$

Initialize $\overline{\mathcal{H}}=\mathcal{H}_{h,i}+\mathcal{H}_{h,i'}$
For all $e\in\overline{\mathcal{H}}$, use a fast SDD solver to compute

$$\widetilde{r}_{h+1,e}(\gamma)\leftarrow(1-\varepsilon)\mathbf{b}_e^\mathsf{T}(\mathbf{L}_{\overline{\mathcal{H}}}+(1+\varepsilon)\gamma\mathbf{I})^{-1}\mathbf{b}_e$$

Set probabilities $\widetilde{p}_{h+1,e}\leftarrow\min\{\widetilde{r}_{h+1,e}(\gamma),\widetilde{p}_{h,e}\}$
Sample $q_{h+1,e}$ from $\mathcal{B}(\widetilde{p}_{h+1,e}/\widetilde{p}_{h,e},q_{h,e})$
Return $\overline{\mathcal{H}}\leftarrow\{(e_{i,j},q_{h+1,e},\widetilde{p}_{h+1,e})\}$ for all $q_{h+1,e}>0$

The structure. We represent a sparsifier $\mathcal{H}$ as a collection of weighted edges $\mathcal{H}\triangleq\{(e_{i,j},q_e,\widetilde{p}_e)\}$, and the Laplacian can be reconstructed as $\mathbf{L}_\mathcal{H}\triangleq\sum_{e\in\mathcal{H}}1/\widetilde{p}_e(q_e/\overline{q})\mathbf{b}_e\mathbf{b}_e^\mathsf{T}$. Intuitively, each edge e has an associated weight based on its probability $\widetilde{p}_e$, and a number of included copies q_e. Keeping multiple copies of each edge helps the random $\mathbf{L}_\mathcal{H}$ to concentrate towards $\mathbf{L}_\mathcal{G}$, where the maximum number of copies $\overline{q}$ for an edge trades-off success probability and the size of $\mathcal{H}$. We assume we have k machines. DiSRe begins by partitioning the graph $\mathcal{G}$ into k sub-graphs $\mathcal{G}_\ell$ on n vertices and $m_\ell\geq n$ edges, such that $\mathcal{G}=\{\mathcal{G}_\ell\}_{i=\ell}^k$ In other words, it splits the matrix $\mathbf{B}_\mathcal{G}$ into submatrices $\mathbf{B}_{\mathcal{G}_i}$ by arbitrarily selecting a subset of rows. The sub-graphs are small enough that they can be stored in memory,[4] and they are

[2] Whenever no regularization is required in the learning task (i.e., HFS, SC), we set $\gamma=0$ and consider "standard" ε-sparsifiers.

[3] Whenever the original graph contains $m\leq\widetilde{\mathcal{O}}(d_{\mathrm{eff}}(\gamma))$edges, there is no need to run DiSRe as the (ε,γ)-sparsifiers would not reduce the size of the graph.

[4] Whenever this is not possible (i.e., m/k is too large to be

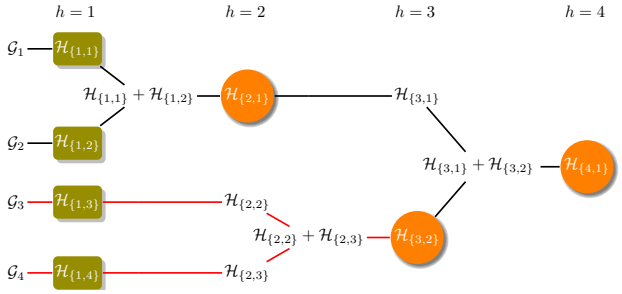

Figure 1. Merge tree for Alg. 1.

also obviously sparsifiers of themselves, therefore we can define an initial set of sparsifiers $\mathcal{S}_1 \triangleq \{\mathcal{H}_{1,\ell}\}_{\ell=1}^{k}$, with $\mathcal{H}_{1,\ell} \triangleq \{(e_{i,j}, q_{1,e} = \overline{q}, \widetilde{p}_{1,e} = 1)\}_{e\in\mathcal{G}_l}$. With this definition, $\mathcal{H}_{1,\ell}$ contains edges $e_{i,j}$ with unit weight $\widetilde{p}_{1,e} = 1$ and $\mathcal{H}_{1,\ell} = \mathcal{G}_\ell$. Starting from these initial sparsifiers, DiSRe proceeds through a sequence of *merge* and *sparsify* operations where two sparsifiers are first combined and then sparsified again to keep having manageable-size graphs at each step. While DiSRe can run on any arbitrary sequence of merges, we consider the most (computationally) effective scheme, where sparsifiers are merged two-by-two in parallel, thus inducing a *balanced* full binary merge tree (see Fig. 2). For notational convenience, we consider that at each iteration h, the inner loop of Alg. 1 only merges two arbitrary sparsifiers from the pool of available sub-graphs $\mathcal{S}_h$ and merges them into a new sparsifier. In practice, multiple merge-and-sparsify operations can be executed in a parallel and asynchronous way. The size of $\mathcal{S}_h$, number of sparsifiers present at layer h, is $|\mathcal{S}_h| = k - h + 1$. Therefore, a node in the tree corresponding to a sparsifier is uniquely identified by two indices $\{h, \ell\}$ where h is the height of the layer and $\ell \leq |\mathcal{S}_h|$ is the index of the node in the layer. We also define the graph $\mathcal{G}_{\{h,\ell\}}$ as the union of all sub-graphs $\mathcal{G}_{\ell'}$ that are reachable from node $\{h, \ell\}$ as leaves (descendants of $\{h, \ell\}$). For example, in Fig. 2, sparsifier $\mathcal{H}_{3,1}$ in node $\{3, 1\}$ approximates the graph $\mathcal{G}_{\{3,2\}} = \mathcal{G}_3 + \mathcal{G}_4$, where we highlight in red the descendant tree.

The resparsification. In Alg. 2 we detail how two arbitrary sparsifiers are combined to obtain a temporary graph $\overline{\mathcal{H}}$. While the merge operation simply combines $\mathcal{H}_{h,i}$ and $\mathcal{H}_{h,i'}$ by summing their weights, the resparsification aims at generating a valid sparsifier from the "original" sub-graph $(\mathcal{G}_{h,i} + \mathcal{G}_{h,i'})$, as if it was directly sparsified at the beginning. We first compute estimates $\widetilde{r}(\gamma)$ of the γ-effective resistance by using fast solvers to invert the strongly diagonal dominant $L_{\overline{\mathcal{H}}} + \gamma\mathbf{I}$ matrix. Instead of sampling edges in $\overline{\mathcal{H}}$ directly proportionally to $\widetilde{r}(\gamma)$ (more precisely $\widetilde{p}_{h+1,e}$), we perform a "resampling" scheme where an edge e is preserved with a "reweighted" probability $\widetilde{p}_{h+1,e}/\widetilde{p}_{h,e}$. Intuitively, the overall sequence of resampling guarantees that at each step $h + 1$, an edge $e \in (\mathcal{G}_{h,i} + \mathcal{G}_{h,i'})$ has the "correct" probability $\widetilde{p}_{h+1,e}$ of being included in the sparsifier.

Performance. We now study the performance of DiSRe and its complexity. *Time* complexity refers to the amount of time necessary to compute the final solution and *work* complexity refers to the total amount of operations carried out by *all* machines to compute the final solution.

Theorem 1. *Let $\varepsilon > 0$ be the accuracy, $0 \leq \delta \leq 1$ the probability of error, and $\rho \triangleq (1 + 3\varepsilon)/(1 - \varepsilon)$. Given an arbitrary graph $\mathcal{G}$ and an arbitrary merge tree structure, if DiSRe is run with parameter $\overline{q} \triangleq 26\rho \log(3n/\delta)/\varepsilon^2$, then each sub-graphs $\mathcal{H}_{\{h,\ell\}}$ is an (ε, γ)-sparsifier of $\mathcal{G}_{\{h,\ell\}}$ with at most $3\overline{q}d_{\mathit{eff}}(\gamma)$ edges with probability $1 - \delta$. Whenever the merge tree is balanced and k is big enough such that $m/k \leq 3\overline{q}d_{\mathit{eff}}(\gamma)$,[5] then merge operations can be run in parallel across the machines with an overall time complexity of $\mathcal{O}(d_{\mathit{eff}}(\gamma)\log^3(n))$, a total work $\mathcal{O}(m\log^3(n))$, and $\mathcal{O}(\log(n))$ rounds of communication.*

Discussion. Kelner & Levin (2013) proposed a sequential algorithm for graph sparsification that closely emulates the batch sampling of Spielman & Srivastava (2011) in a semi-streaming setting and incrementally constructs an ε-sparsifier. However, their proof had a flaw since they treated dependent variables as independent (Calandriello et al., 2016). Kyng et al. (2016) resolved the issues in the proof of Kelner & Levin (2013) and showed that a slightly modified algorithm can construct a sparsifier with $\mathcal{O}(n\log(n)/\varepsilon^2)$ edges in $\mathcal{O}(m\log^2(n)/\varepsilon^2)$ time, matching the space complexity of batch sampling. The method proposed by Kyng et al. (2016) can be further improved by parallelizing its computation over multiple machines. Using the parallel sparsification algorithm of Koutis & Xu (2016), the time complexity can be reduced up to $\widetilde{\mathcal{O}}(\log^6(n))$. Nonetheless, since these methods require *random access to the edges*, they cannot be easily distributed (it would have $\mathcal{O}(m\,\mathrm{polylog}(n))$ communication cost) and scaled to graphs that cannot be stored on a single machine. Furthermore, the algorithm of Kyng et al. (2016) accurately reconstructs the whole spectrum of the Laplacian, which leads to sparsifiers whose number of edges scales linearly with n. On the other hand, in regularized learning tasks, the presence of multiplicative and additive spectral error allows creating smaller sparsifiers whose size scales with $d_{\mathrm{eff}}(\gamma)$. Notice that, for γ large enough, this possibly means sparsifiers with less than $n - 1$ edges, necessarily leading to disconnected graphs. Finally, note that merging two traditional ε-sparsifiers gives an ε-sparsifier, merging two (γ, ε)-sparsifiers produces a less accurate $(2\gamma, \varepsilon)$-sparsifier. Therefore simple merge-and-reduce strategies (Feldman et al.,

stored on a single machine), we can simply apply the same merging scheme of DiSRe by loading *small enough* chunks of the graph and sparsifying them sequentially.

[5] This implies that there are enough machines so that the leaves in the merge tree already have relatively sparse sub-graphs.

2013), which address every resparsification as independent, would either cumulate errors or require multiple passes over the data. Similarly to Kyng et al. (2016), `DiSRe`'s sequential `Merge-Resparsify` solves this problem (Appendix B).

Mixed additive-multiplicative reconstruction is studied more extensively in randomized matrix algebra (Drineas & Mahoney, 2017). Cohen et al. (2016) developed an efficient method to spectrally sparsify generic matrices up to $(1 \pm \varepsilon)$ multiplicative and γ-additive errors using an incremental sampling method based on *ridge leverage scores* (i.e., the analog of γ-effective resistances for matrices). If applied to graph Laplacians, their method adds edges incrementally and returns an (ε, γ)-sparsified graph with $\mathcal{O}(d_{\text{eff}}(\gamma)\log^2(n))$ edges in $\mathcal{O}(m\log(n))$ time. Nonetheless, Cohen et al. (2016) provided only ε-sparsifiers, suggesting to set γ as small as possible, and did not explore the advantages possible in machine learning. Moreover, no existing (ε, γ)-sparsifier construction method can leverage both distribution and fast solvers. Cohen et al. (2016) can only add edges (but not remove them as `DiSRe`), preventing repeated merge-and-resparsify. Other streaming RLS sampling methods, such as by Cohen et al. (2017), use dense intermediate sketches, such as *frequent directions* (Ghashami et al., 2016), that are not Laplacians of a subgraph and cannot be easily paired with near-linear solvers for Laplacians.

4. Downstream Guarantees

We now show how the spectral reconstruction guarantees provided by (ε, γ)-sparsifiers translate into guarantees on the quality of the approximate solutions computed using $\mathcal{H}$ instead of $\mathcal{G}$. We first introduce a result for ε-sparsifiers in SSL and then show how for regularized problems, (ε, γ)-sparsification can further improve computational performance without loss in accuracy in LapSmo.

4.1. Generalization bounds for SSL

Given the closed form solutions of `HFS` (Eq. 2) and `LTR` (Eq. 3), we simply replace $\mathbf{L}_{\mathcal{G}}$ with $\mathbf{L}_{\mathcal{H}}$ and then run a nearly-linear time solver to obtain approximate solutions $\widetilde{\mathbf{f}}_{\text{HFS}}$ and $\widetilde{\mathbf{f}}_{\text{LTR}}$. We compare approximate solutions to their exact counterparts in the context of algorithmic stability.

Definition 3. *Let $\mathcal{L}$ be a transductive learning algorithm. We denote by $\mathbf{f}$ and $\mathbf{f}'$ the solutions obtained by running $\mathcal{L}$ on datasets $\mathcal{V} \triangleq (\mathcal{S}, \mathcal{T})$ and $\mathcal{V} \triangleq (\mathcal{S}', \mathcal{T}')$ respectively. $\mathcal{L}$ is uniformly β-stable w.r.t. the squared loss if there exists $\beta \geq 0$ such that for any two partitions $(\mathcal{S}, \mathcal{T})$ and $(\mathcal{S}', \mathcal{T}')$ that differ by exactly one training (and test) point and for all $i \in [n]$, we have $|([\mathbf{f}]_i - [\mathbf{y}]_i)^2 - ([\mathbf{f}']_i - [\mathbf{y}]_i)^2| \leq \beta$.*

The stability of LTR was proven by Cortes et al. (2008). On the other hand, the singularity of the Laplacian may lead to unstable behavior in HFS due to the $(\gamma\ell\mathbf{L}_{\mathcal{G}} + \mathbf{I}_{\mathcal{S}})^+$ pseudo-inverse, with drastically different results for small perturbations of the dataset. For this reason, we take the `Stable-HFS` algorithm by Belkin et al. (2004), where an additional regularization term is introduced to restrict the space of admissible solutions to the space $\mathcal{F} \triangleq \{\mathbf{f} : \langle \mathbf{f}, \mathbf{1} \rangle = 0\}$ of solutions orthogonal to the null space of $\mathbf{L}_{\mathcal{G}}$ (i.e., centered functions). As shown by Belkin et al. (2004), to satisfy the constraint, it is sufficient to set an additional regularization parameter μ to $\mu \triangleq ((\gamma\ell\mathbf{L}_{\mathcal{G}} + \mathbf{I}_{\mathcal{S}})^+\mathbf{y}_{\mathcal{S}})^\top\mathbf{1}/((\gamma\ell\mathbf{L}_{\mathcal{G}} + \mathbf{I}_{\mathcal{S}})^+\mathbf{1})^\top\mathbf{1}$, and compute the solution $\widehat{\mathbf{f}}_{\text{STA}}$ as $\widehat{\mathbf{f}}_{\text{STA}} \triangleq (\gamma\ell\mathbf{L}_{\mathcal{G}} + \mathbf{I}_{\mathcal{S}})^+(\mathbf{y}_{\mathcal{S}} - \mu\mathbf{1})$. While `Stable-HFS` is more stable and thus more suited for theoretical analysis, its space and time requirement remains $\mathcal{O}(m)$ and cannot be applied to graphs with a large number of edges. Therefore, we again replace $\widehat{\mathbf{f}}_{\text{STA}}$ with an approximate solution $\widetilde{\mathbf{f}}_{\text{STA}}$ computed using $\mathbf{L}_{\mathcal{H}}$. Define $\widehat{R}(\mathbf{f}) \triangleq \frac{1}{\ell}\sum_{i=1}^{\ell}(\mathbf{f}(x_i) - \mathbf{y}(x_i))^2$ as the empirical error and $R(\mathbf{f}) \triangleq \frac{1}{u}\sum_{i=1}^{u}(\mathbf{f}(x_i) - \mathbf{y}(x_i))^2$ as the generalization.

Theorem 2. *Let $\mathcal{G}$ be a fixed (connected) graph with eigenvalues $0 = \lambda_1(\mathcal{G}) < \lambda_2(\mathcal{G}) \leq \ldots \leq \lambda_n(\mathcal{G})$, and $\mathcal{H}$ an ε-sparsifier of $\mathcal{G}$. Let $\mathbf{y} \in \mathbb{R}^n$ be the labels of the nodes in $\mathcal{G}$ with $|\mathbf{y}(x)| \leq c$ and $\mathcal{F}$ be the set of centered functions such that $|\mathbf{f}(x) - \mathbf{y}(x)| \leq 2c$. Let $\mathcal{S} \subset \mathcal{V}$ be a random subset of labeled nodes, if the labels $\mathbf{y}_{\mathcal{S}}$ are centered, then w.p. at least $1 - \delta$ (w.r.t. the random generation of the sparsifier $\mathcal{H}$ and the random subset of labeled points $\mathcal{S}$) the resulting `Stable-HFS` solution satisfies*

$$R(\widetilde{\mathbf{f}}) \leq \widehat{R}(\widehat{\mathbf{f}}) + \beta + \left(2\beta + \frac{4c^2(\ell+u)}{\ell u}\right)\sqrt{\frac{\pi(\ell,u)\ln\frac{1}{\delta}}{2}} + \frac{1}{1-\varepsilon}\left(\frac{2(1+\varepsilon)\varepsilon\ell\gamma\lambda_2(\mathcal{G})c}{((1-\varepsilon)\ell\gamma\lambda_2(\mathcal{G})-1)^2}\right)^2, \tag{6}$$

where $\widetilde{\mathbf{f}}$ and $\widehat{\mathbf{f}}$ are computed on $\mathcal{H}$ and $\mathcal{G}$,

$$\pi(\ell,u) \triangleq \frac{\ell u}{\ell+u-0.5}\frac{2\max\{\ell,u\}}{2\max\{\ell,u\}-1} \quad \text{and}$$

$$\beta \leq \frac{3c\sqrt{\ell}}{((1-\varepsilon)\ell\gamma\lambda_2(\mathcal{G})-1)^2} + \frac{4c}{(1-\varepsilon)\ell\gamma\lambda_2(\mathcal{G})-1}.$$

Thm. 2 (full proof in Appendix A) shows how approximating $\mathcal{G}$ with $\mathcal{H}$ impacts the generalization error as the number of labeled samples ℓ increases. If we set $\varepsilon = 0$, we recover the bound of Cortes et al. (2008), which depends only on $\widehat{R}(\widehat{\mathbf{f}})$ and β. When $\varepsilon > 0$, we see from Eq. 6 that the two terms already present in the exact case are either unchanged ($\widehat{R}(\widehat{\mathbf{f}})$) or increase only by a constant factor β. Because of the approximation, a new error term (the last one in Eq. 6) is added to the bound, but we can see that it is negligible compared to β. In fact, it converges to zero as $\mathcal{O}(\varepsilon^2/\ell^2(1-\varepsilon)^4)$ as ℓ grows and it is dominated by β for any constant value

of ε. This means that increasing ε corresponds to a constant increase in the bound, regardless of the size of the problem. Consequently, ε can be freely chosen to trade off accuracy and space complexity (Thm. 1) depending on the problem constraints. Finally, because the eigenvalues present in the bound are the ones of the original graph, any additional knowledge on the spectral properties of the input graph can be easily included in the analysis. Therefore, it is straightforward to provide stronger guarantees for `Sparse-HFS` when combined with assumptions on the graph generating model. Finally, we remark the level of generality of this result that holds for the integration between `HFS` and any ε-accurate spectral sparsification method. We postpone computational considerations to the following subsection.

4.2. Generalization bounds for LapSmo

Starting from the closed form solution of LapSmo (Eq. 1) we can replace the $\mathbf{L}_\mathcal{G}$ matrix with a sparsified Laplacian $\mathbf{L}_\mathcal{H}$ and using a fast linear solver, compute an approximate solution $\widetilde{\mathbf{f}} = (\lambda\mathbf{L}_\mathcal{H} + \mathbf{I})^{-1}\mathbf{y}$ in $\mathcal{O}(n\log^2(n))$ time and $\mathcal{O}(n\log(n))$ space. Finally, we can decompose the error as $\|\mathbf{f}^\star - \widetilde{\mathbf{f}}\|_2^2 \leq \|\mathbf{f}^\star - \widehat{\mathbf{f}}\|_2^2 + \|\widehat{\mathbf{f}} - \widetilde{\mathbf{f}}\|_2^2$. The first term can be bounded using classical results from empirical process theory (Bühlmann & Van De Geer, 2011). We bound the second term in the following theorem.

Theorem 3. *For an arbitrary graph $\mathcal{G}$ and its (ε,γ)-sparsifier, let $\widehat{\mathbf{f}}$ be the* LapSmo *solution computed using $\mathbf{L}_\mathcal{G}$ and $\widetilde{\mathbf{f}}$ the solution computed using $\mathbf{L}_\mathcal{H}$. Then,*

$$\|\widetilde{\mathbf{f}} - \widehat{\mathbf{f}}\|_2^2 \leq \frac{\varepsilon^2}{1-\varepsilon}\,(0.25 + \lambda\gamma)\left(\lambda\widehat{\mathbf{f}}^\mathsf{T}\mathbf{L}_\mathcal{G}\widehat{\mathbf{f}} + \lambda\gamma\|\widehat{\mathbf{f}}\|_2^2\right),$$

where λ is the regularization of LapSmo.

For ε-sparsifiers, Sadhanala et al. (2016) derive a similar bound $\|\widetilde{\mathbf{f}} - \widehat{\mathbf{f}}\|_2^2 \leq \mathcal{O}(\lambda\widehat{\mathbf{f}}^\mathsf{T}\mathbf{L}_\mathcal{G}\widehat{\mathbf{f}})$. Setting $\gamma = 0$, we recover their bound up to constants. When $\gamma > 0$ instead, additional error terms emerge due to the introduced bias. In particular, the term $\lambda\gamma\|\widehat{\mathbf{f}}\|_2^2$ depends on the norm of the exact solution $\widehat{\mathbf{f}}$, which in turn depends on the value of λ. Nonetheless, when $\|\mathbf{f}^\star\|_2^2$ is small, as is the case in our experiments, setting $\gamma = 1/\lambda$ makes this term a constant, which is reflected by the good empirical performance. Computationally, for both `Stable-HFS` and LapSmo, passing from computing a solution on the full graph to computing a solution on the sparsifier reduces the number of edges, which makes the memory and runtime plummet. Moreover, carefully distributing the sparsification process across multiple machines allows computing a final solution in a time *independent* from the number of edges, since the preprocessing sparsification step takes only $\mathcal{O}(n\log^3(n))$ time, and the solution step only $\mathcal{O}(n\log^2(n))$. Up to logarithmic terms, this results in an overall $\widetilde{\mathcal{O}}(n)$ near-linear runtime, without any assumptions on the input graph. For graphs with a particularly favorable spectrum and problems with enough regularization, this is only $\widetilde{\mathcal{O}}(d_{\text{eff}}(\gamma))$, resulting in a potentially sub-linear runtime. This result, only possible due to a particular structure of learning problems, opens up unexplored possibilities that would not be possible for general graph problems.

4.3. Bounds for other problems

Many other problems can be well approximated using (ε,γ)-sparsifiers. For example, the cost of a SC solution evaluated on $\mathbf{L}_\mathcal{H}$ is very close to the cost evaluated on $\mathbf{L}_\mathcal{G}$.

Proposition 2. *For any rank k orthogonal projection $\mathbf{F}^\mathsf{T}\mathbf{F}$, if $\mathcal{H}$ is an (ε,γ)-sparsifier of $\mathcal{G}$, we have*

$$\operatorname{Tr}(\mathbf{F}^\mathsf{T}\mathbf{L}_\mathcal{H}\mathbf{F}) \leq (1+\varepsilon)\operatorname{Tr}(\mathbf{F}^\mathsf{T}\mathbf{L}_\mathcal{G}\mathbf{F}) + \varepsilon\gamma k.$$

Therefore, a clustering that well separates the sparsifier will also separate well the true graph. Similarly, we can obtain strong approximation guarantees for a variety of other Laplacian-based algorithms. Regularized problems such as `LTR` (Cortes et al., 2008), Laplacian-regularized least squares, and Laplacian SVM (Belkin et al., 2005) are of particular interest since the additive γ error is absorbed by the regularization and it is possible to provide strong generalization guarantees.

5. Experiments

We empirically validate our theoretical findings by testing how (ε,γ)-sparsifiers improves computational complexity without sacrificing final accuracy.

Dataset. We run experiments on the Amazon co-purchase graph (Sadhanala et al., 2016). This graph fits our setting: It cannot be generated from vectorial data and is only artificially sparse, since the crawler that created it had no access to the true private co-purchase network held by Amazon. To compensate, Gleich & Mahoney (2015) use a densification procedure that given the graph adjacency matrix $\mathbf{A}_\mathcal{G}$, computes all k-step neighbors $\mathbf{A}_{\mathcal{G},k} \triangleq \sum_{s=1}^{k}\mathbf{A}_\mathcal{G}^s$. We make the graph unweighted for numerical stability. The final graph has $n = 334,863$ nodes and $m = 98,465,352$ edges, with an average degree of 294. We followed an approach similar to Sadhanala et al. (2016) and introduce a hand-designed smooth signal as a target. We then perform 2000 iterations of the power method to compute an approximation of the smallest eigenvector $\mathbf{v}_{\min}$, which is used as a smooth function over the graph.

Baselines. For all setups, we compute an "exact" solution (up to convergence error) using a fast linear solver. Computing this `EXACT` baseline requires $\mathcal{O}(m\log(n))$ time and $\mathcal{O}(m)$ space and achieves the best performance. Afterwards, we compare three different sparsification procedures to eval-

Alg.	*Parameters*	$\lvert\mathcal{E}\rvert$ ($x10^6$)	*Err.* SSL($\ell=346$)	*Err.* SSL($\ell=672$)	*Err.* $D(\widetilde{\mathbf{f}})(\sigma=10^{-3})$	*Err.* $D(\widetilde{\mathbf{f}})$ $(\sigma=10^{-2})$
EXACT		98.5	0.312 ± 0.022	0.286 ± 0.010	0.067 ± 0.0004	0.756 ± 0.006
kN	$k=60$	15.7	0.329 ± 0.0143	0.311 ± 0.027	0.172 ± 0.0004	0.822 ± 0.002
kN	$k=90$	21.2	0.334 ± 0.024	0.311 ± 0.024	0.125 ± 0.0002	0.811 ± 0.003
DiSRe	$\gamma=0, \overline{q}=100$	15	0.314 ± 0.0165	0.296 ± 0.015	0.068 ± 0.0003	0.758 ± 0.005
DiSRe	$\gamma=0, \overline{q}=150$	22.8	0.314 ± 0.0158	0.310 ± 0.024	0.068 ± 0.0004	0.756 ± 0.005
DiSRe	$\gamma=10^3, \overline{q}=100$	7.3	–	–	0.072 ± 0.0003	0.789 ± 0.005
DiSRe	$\gamma=10^2, \overline{q}=100$	11.8	–	–	0.068 ± 0.0002	0.772 ± 0.004
DiSRe	$\gamma=10, \overline{q}=100$	14.4	–	–	0.068 ± 0.0004	0.760 ± 0.004

Table 1. Results for the SSL and the smoothing problems.

uate if they can accelerate computation while preserving accuracy. We run DiSRe with different values of γ depending on the setting. For empirically strong heuristics, we attempted to uniformly subsample the edges, but at the sparsity level achieved by the other methods, the uniformly sampled sparsifier is disconnected and highly inaccurate. Instead, we compare to the state-of-the-art k-neighbors (kN) heuristic by Sadhanala et al. (2016), which is just as fast as uniform sampling and more accurate in practice.

Experimental procedure. We repeat each experiment 10 times with different sparsifiers and report the average performance of $\widetilde{\mathbf{f}}$ on the specific task and its standard deviation. More details on experiments are given in the Appendix C.

5.1. Laplacian smoothing with Gaussian noise

We set $\mathbf{f}^\star = \mathbf{v}_{\min}$ and test different levels of noise, $\log_{10}(\sigma) \in \{-3, -2, -1, 0\}$. After constructing the sparsifier $\mathcal{H}$, we compute an approximate solution $\widetilde{\mathbf{f}}$ using LapSmo (Eq. 2) with $\lambda \in \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10\}$. We measure the performance by the squared error $D(\widetilde{\mathbf{f}}) = \|\mathbf{f}^\star - \widetilde{\mathbf{f}}\|_2^2$. As $\|\mathbf{f}^\star\|_2^2 = \|\mathbf{v}_{\min}\|_2^2 = 1$, good values of $D(\widetilde{\mathbf{f}})$ should be below 1.

Accuracy. In the interest of space, in Tab. 1, we report results for $\sigma = \{0.001, 0.01\}$ and the best regularization λ for each method. We first notice that all sparsifiers are considerably smaller than the original graph, keeping only a small fraction of its edges. The smallest sparsifiers are obtained by DiSRe when γ is large. The comparison with DiSRe with $\gamma = 0$ (i.e., ε-sparsifier) confirms that the additive error translates into an extra compression of the resulting sparsifier. This also impacts the accuracy which degrades as γ increases. Nonetheless, we notice that while ε-sparsifiers perfectly match the accuracy of the exact method, even for large γ (and thus much smaller graph), DiSRe still outperforms kN, which has a significantly worse accuracy. Finally, we note that for $\gamma = 0$, the impact of $\overline{q}$ is as expected: Increasing $\overline{q}$ increases the size of the sparsifier and slightly improves the performance.

Computational complexity. All algorithms require 90s to load the graph from disk. The preprocessing phase of kN takes slightly less than 1min, while DiSRe's takes 12min on 4 machines. For the solving step, EXACT is unsurprisingly the slowest, requiring 12min to compute an $\widehat{\mathbf{f}}$ solution. Both kN and (ε, γ)-sparsifiers require 1–2min, depending on the number of edges preserved. Overall, preprocessing the graph with DiSRe before computing a solution does not introduce any overhead compared to EXACT (both take roughly 12min). We notice that while kN is overall faster, the time for DiSRe could be easily reduced by increasing the number of parallel processes when computing effective resistances or with a better network topology allowing point-to-point communication. Moreover, once we have access to an accurate ε-sparsifier, it is easier to solve problem repeatedly, e.g., to cross-validate regularization. For example, computing a solution for 4 different values of λ (see the appendix) is crucial for good performance and requires 48min for EXACT and only 20min for DiSRe. Finally, memory usage is reduced by a factor of 3 as EXACT requires over 30GB of memory to execute while DiSRe never exceeds 10GB. We expect these advantages to only grow larger as we scale to larger graphs.

5.2. SSL with harmonic function solution

We also test DiSRe on a SSL problem. The labels are generated taking the sign of $\mathbf{f}^\star = \mathbf{v}_{\min}$ and $\ell \in \{20, 346, 672, 1000\}$ labels are revealed. The labeled nodes are chosen at random so that 0 and 1 labels are balanced in the dataset. We run Stable-HFS with $\lambda \in \{10^{-6}, 10^{-4}, 10^{-2}, 1\}$. In Tab. 1, we report results for $\ell = \{346, 672\}$ and the best λ for each method. We run DiSRe with $\gamma = 0$ as Stable-HFS does not have any regularization and ε-sparsifiers are preferable. The average size of the sparsifiers is the same as before as they are agnostic to the learning task. Similar to the smoothing case, DiSRe achieves a performance that closely approximates the exact solution, despite the significant compression of the original graph. Furthermore, the effectiveness of the ε-sparsifier returned by DiSRe is confirmed by its comparison with kN, whose error is significantly worse. Finally, we notice that the computational analysis in the previous section holds for SSL as well. In fact, although the learning task is different, we use the same SSD solver to compute the HFS and thus the running time are comparable in the two tasks.

Acknowledgements

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and several universities as well as other organizations (see https://www.grid5000.fr). The research presented was also supported by European CHIST-ERA project DELTA, French Ministry of Higher Education and Research, Nord-Pas-de-Calais Regional Council, Inria and Otto-von-Guericke-Universität Magdeburg associated-team north-european project Allocate, and French National Research Agency projects ExTra-Learn (n.ANR-14-CE24-0010-01) and BoB (n.ANR-16-CE23-0003). I. Koutis is supported by NSF CAREER award CCF-1814563.

References

Akoglu, L., Tong, H., and Koutra, D. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery*, 29(3):626–688, 2015.

Alaoui, A. E. and Mahoney, M. W. Fast randomized kernel methods with statistical guarantees. In *Neural Information Processing Systems*, 2015.

Belkin, M. and Niyogi, P. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Neural Information Processing Systems*, 2001.

Belkin, M., Matveeva, I., and Niyogi, P. Regularization and Semi-Supervised Learning on Large Graphs. In *Conference on Learning Theory*, 2004.

Belkin, M., Niyogi, P., and Sindhwani, V. On manifold regularization. In *International conference on Artificial Intelligence and Statistics*, 2005.

Bühlmann, P. and Van De Geer, S. *Statistics for high-dimensional data: methods, theory and applications*. Springer Science & Business Media, 2011.

Calandriello, D., Lazaric, A., and Valko, M. Analysis of kelner and levin graph sparsification algorithm for a streaming setting. *arXiv:1609.03769*, 2016.

Calandriello, D., Lazaric, A., and Valko, M. Distributed sequential sampling for kernel matrix approximation. In *International conference on Artificial Intelligence and Statistics*, 2017.

Chapelle, O., Schlkopf, B., and Zien, A. *Semi-supervised learning*. The MIT Press, 2010.

Cohen, M. B., Musco, C., and Pachocki, J. W. Online row sampling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2016.

Cohen, M. B., Musco, C., and Musco, C. Input sparsity time low-rank approximation via ridge leverage score sampling. In *Symposium on Discrete Algorithms*, 2017.

Cortes, C., Mohri, M., Pechyony, D., and Rastogi, A. Stability of transductive regression algorithms. In *International Conference on Machine Learning*, 2008.

Cucuringu, M., Koutis, I., Chawla, S., Miller, G., and Peng, R. Simple and scalable constrained clustering: a generalized spectral method. In *International Conference on Artificial Intelligence and Statistics*, 2016.

Drineas, P. and Mahoney, M. W. Lectures on randomized numerical linear algebra. *arXiv:1712.08880*, abs/1712.08880, 2017.

Feldman, D., Schmidt, M., and Sohler, C. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. In *Symposium on Discrete Algorithms*, 2013.

Ghashami, M., Liberty, E., Phillips, J. M., and Woodruff, D. P. Frequent directions: Simple and deterministic matrix sketching. *SIAM Journal on Computing*, 45(5):1762–1792, 2016.

Gleich, D. F. and Mahoney, M. W. Using local spectral methods to robustify graph-based learning algorithms. In *Knowledge Discovery and Data Mining*, pp. 359–368, 2015.

Jamakovic, A. and Mieghem, P. V. The Laplacian spectrum of complex networks. In *European Conference on Complex Systems*, 2006.

Kelner, J. A. and Levin, A. Spectral sparsification in the semi-streaming setting. *Theory of Computing Systems*, 53(2):243–262, 2013.

Koutis, I. and Xu, S. C. Simple parallel and distributed algorithms for spectral graph sparsification. *Transactions on Parallel Computing*, 3(2):14, 2016.

Koutis, I., Miller, G. L., and Peng, R. A nearly-m log n time solver for SDD linear systems. In *Symposium on Foundations of Computer Science*, pp. 590–598, 2011.

Kyng, R. and Sachdeva, S. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *Foundations of Computer Science*, 2016.

Kyng, R., Pachocki, J., Peng, R., and Sachdeva, S. A framework for analyzing resparsification algorithms. In *Symposium on Theory of Computing*, 2016.

Lee, J. R., Gharan, S. O., and Trevisan, L. Multiway spectral partitioning and higher-order cheeger inequalities. *Journal of the ACM*, 61(6):37:1–37:30, 2014.

Levy, H. *Stochastic dominance: Investment decision making under uncertainty*. Springer, 2015.

Sadhanala, V., Wang, Y.-X., and Tibshirani, R. Graph sparsification approaches for laplacian smoothing. In *International conference on Artificial Intelligence and Statistics*, 2016.

Samukhin, A. N., Dorogovtsev, S. N., and Mendes, J. F. F. Laplacian spectra of, and random walks on, complex networks: Are scale-free architectures really important? *Physical Review E*, 77(3):036115, 2008.

Spielman, D. A. and Srivastava, N. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6), 2011.

Spielman, D. A. and Teng, S.-H. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

Tropp, J. A. Freedman's inequality for matrix martingales. *Electronic Communications in Probability*, 16:262–270, 2011.

Tropp, J. A. An introduction to matrix concentration inequalities. *Foundations and Trends® in Machine Learning*, 8 (1-2):1–230, 2015.

Von Luxburg, U. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

Von Luxburg, U., Radl, A., and Hein, M. Hitting and commute times in large random neighborhood graphs. *Journal of Machine Learning Research*, 15(1):1751–1798, 2014.

Zhan, C., Chen, G., and Yeung, L. F. On the distributions of Laplacian eigenvalues versus node degrees in complex networks. *Physica A: Statistical Mechanics and its Applications*, 389(8):1779–1788, 2010.

Zhu, X., Ghahramani, Z., and Lafferty, J. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In *International Conference on Machine Learning*, 2003.