Graph Tikhonov Regularization and Interpolation via Random Spanning Forests

Yusuf Yigit Pilavci, Pierre-Olivier Amblard, Simon Barthelme, Nicolas Tremblay

Graph Tikhonov Regularization and Interpolation via Random Spanning Forests

Yusuf Yiğit Pilavcı, Pierre-Olivier Amblard, Simon Barthelmé, Nicolas Tremblay

***Abstract*—Novel Monte Carlo estimators are proposed to solve both the Tikhonov regularization (TR) and the interpolation problems on graphs. These estimators are based on random spanning forests (RSF), the theoretical properties of which enable to analyze the estimators' theoretical mean and variance. We also show how to perform hyperparameter tuning for these RSF-based estimators. TR is a component in many well-known algorithms, and we show how the proposed estimators can be easily adapted to avoid expensive intermediate steps in generalized semi-supervised learning, label propagation, Newton's method and iteratively reweighted least squares. In the experiments, we illustrate the proposed methods on several problems and provide observations on their run time.**

***Index Terms*—graph signal processing, random spanning forests, smoothing, interpolation, semi-supervised learning, label propagation, Newton's method, iteratively reweighted least square.**

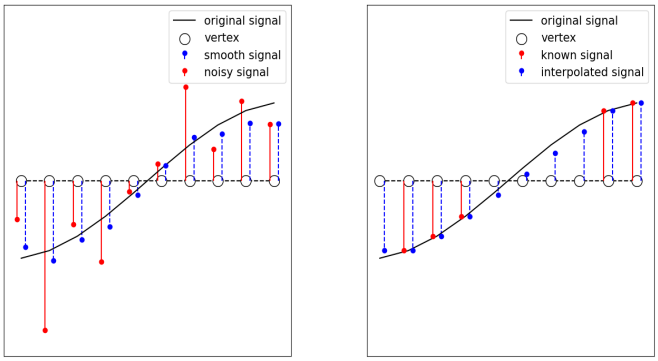


Fig. 1: Smoothing (left) and interpolation (right) in discrete 1D signal processing can be considered as a graph signal processing problem where the underlying graph is a path graph.

I. Introduction

GRAPHS are ubiquitous models of complex structures, *e.g.* social, transportation, sensors or neuronal networks. The vertices and edges, the main components of graphs, are natural representations of the elements of a network and the links between them, respectively. In many applications, these networks often come with data on the elements. For example, in a transportation network (the roads and their intersections are respectively the nodes and edges), the data can be the traffic measured at the intersections; or in a brain network, it could be the activity of each individual brain region [1]. Such type of data over vertices are called graph signals [2], [3].

Graph Tikhonov regularization and interpolation. Graph signal processing (GSP) is a field of research developed in the last decade dedicated to extending classical signal processing tools to graph signals [2], [4]. In this paper, we focus on two essential operations on graph signals, namely graph Tikhonov regularization and graph interpolation. In the former, noisy measurements of a graph signal are known over all vertices. Tikhonov regularization is a smoothing operation and recovers the underlying signal assuming that it varies slowly between neighboring vertices in the graph. Under the same assumption, interpolation, on the other hand, is the operation of recovering the signal when it is known only over a few vertices (see Fig. 1 for an illustration on the path graph). Both operations are thoroughly analyzed in [5], [6], [7], [8] and we refer the reader to Section III for formal definitions.

Hyperparameter tuning. As usual in regularization schemes, both graph TR and interpolation have hyper-parameters that need to be to tuned. Automatically tuning these parameters is not a trivial problem. In the classical framework of linear smoothers [9], there exists many methods such as Akaike and Bayesian information criteria, Marlow's Cp, leave-one-out and generalized cross validations or Stein's unbiased risk estimator. In this paper, we study possible graph adaptations for some of them to automatically tune hyper-parameters in graph TR and interpolation.

Solving graph TR and interpolation as a building block. The solutions of graph TR and interpolation appear in the intermediate steps of various graph algorithms.

Semi-supervised node classification is one problem that naturally boils down to a graph interpolation scheme. A few labels are a priori known and the purpose is to infer, via graph interpolation, the labels for all nodes given the structure of the underlying graph. One solution to this problem is given by a well-known Semi-Supervised Learning (SSL) algorithm, called label propagation [5], [36]. Label propagation is an iterative algorithm whose solution converges to the solution of the

This work was partly funded by the French National Research Agency in the framework of the "Investissements d'avenir" program (ANR-15-IDEX-02), the LabEx PERSYVAL (ANR-11-LABX-0025-01), the ANR GraVa (ANR-18-CE40-0005) the ANR GenGP (ANR-16-CE23-0008) the Grenoble Data Institute (ANR-15-IDEX-02) the MIAI@Grenoble Alpes chairs "LargeDATA at UGA." and "Pollutants" (ANR-19-P3IA-0003).

Yusuf Yiğit Pilavcı, Pierre-Olivier Amblard, Simon Barthelmé and Nicolas Tremblay are with CNRS, Univ. Grenoble Alpes, Grenoble INP, GIPSA-lab, Grenoble, France. (e-mail: firstname.lastname@gipsa-lab.fr with firstname.lastname@ = yusuf-yigit.pilavci@, pierre-olivier.amblard@, simon.barthelme@, nicolas.tremblay@)

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org., provided by the authors. The material includes proof of proposition 3 and an illustration for Section IV-C. Contact Yusuf Yiğit Pilavcı for further questions about this work.

graph interpolation problem. Along with the connections to the Dirichlet boundary problem on graphs [10], this solution can be seen as assigning labels to nodes that *smoothly* interpolate between known labels at the boundary points.

Graph TR, besides its use for graph signal denoising, also appears in SSL algorithms for node classification, such as in the work of Zhou *et al.* [11], later generalized by Avrachenkov *et al.* [12]. Moreover, graph TR can also be used in graph optimization algorithms. Two examples are Newton's method [13] and iteratively reweighted least squares (IRLS) [14]. In both methods, the computationally expensive steps can be formulated and solved as graph TR problems.

Classical approaches. Both graph TR and interpolation have closed-form solutions. However, directly computing these solutions requires the inversion of a matrix of size $n \times n$, and thus, up to $\mathcal{O}(n^3)$ elementary operations. For large graphs (*i.e.* with $n \geq 10^4$), this is prohibitive and the state-of-the-art relies on approximate methods. These approaches may be roughly separated in two groups, iterative methods (*e.g.* conjugate gradient method with preconditioning [15]) and polynomial approximations (*e.g.* Chebyshev polynomials [16]). Both classes of methods run in linear time with the number of edges m.

Random processes on graphs. A longstanding and fruitful approach to studying the properties of graphs has been to study the properties of random processes on graphs (via random walks, for instance). This paper will take such a perspective to propose novel estimators for the two problems presented.

For instance, a very well-known fact is the link between the smallest non-null eigenvalue of the graph's Laplacian matrix and the mixing time of a random walk on the graph (see, *e.g.,* [17]). Other examples include properties of electrical networks, such as potential functions or effective resistances, that can be interpreted in terms of probabilistic quantities defined for random walks such as hitting time probabilities [18].

Closer to our work, Wu *et al.* [19], [20] study probabilistic properties of a particular random process, called partially absorbed random walk; and further leverage random walks to give practical insights into algorithms dedicated to image retrieval, local clustering and semi-supervised learning.

Random spanning forests. In this paper, we will focus on random spanning forests (RSFs): random collections of disjoint trees that cover all nodes in the graph (a formal definition is in Section II). Interestingly, connections between graph TR / interpolation and RSFs have been observed by a few authors in the past, such as in [21]. Avena *et al.* [22], [23] analyze several aspects of these connections. RSFs not only have a rich theoretical background (they have connections with Markov chains, determinantal point processes, spectral graph theory), they also come with an efficient sampling algorithm [22]: a variant of Wilson's algorithm [24] based on loop-erased random walks.

Our contributions. In the conference paper [25], we have already presented some preliminary results. Differing from [25], in this work,

- We extend the theoretical analysis on the RSF-based estimators proposed in [25]:
  - Building upon known results on RSFs, we generalize the error analysis to cases where the regularization hyperparameter is not constant over the vertices.
  - We provide a precise expression for the computational cost of the estimators.
- Using certain statistics of RSFs, we provide a novel scheme to correctly tune the hyperparameters.
- We discuss how these estimators may be leveraged to approximately solve the graph interpolation problem.
- We show that the proposed estimators have versatile use in other graph-based problems and their solutions such as generalized semi-supervised learning, label propagation, Newton's method and IRLS.
- We provide illustrations for different use cases. Tikhonov regularization and Newton's method are demonstrated in image denoising applications. Graph interpolation and generalized SSL algorithms are used to solve classification tasks on citation networks.
- The runtime of the proposed methods are compared with state-of-the-art methods on certain graphs.

The Julia code to reproduce this paper's results is available on the authors' website.[1]

Organization of the paper. We start with the necessary background on graphs and RSFs in Section II. Then, we introduce the proposed methods in Section III. In Section IV, we examine several extensions to different graph-related problems. Finally, in Section V, we illustrate the methods and provide a runtime analysis and we conclude in Section VI.

II. Background on RSFs

This section contains background on graph theory, random spanning trees and forests.

A. Graph theory

A directed weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ consists of a set of vertices $\mathcal{V} = \{1, 2, ..., n\}$ and edges $\mathcal{E} = \{(i, j) \in \mathcal{V} \times \mathcal{V}\}$. The weight function $w : \mathcal{V} \times \mathcal{V} \to \mathbb{R}^+$ maps each edge in $\mathcal{E}$ to a positive weight and others to 0. A graph is called undirected if $w(i, j) = w(j, i)$ for all distinct vertices i and j. In the following, unless otherwise specified only undirected graphs are considered. Graphs are often represented using matrices, and several matrix descriptions are available.

The weighted adjacency matrix is $\mathsf{W} = [w(i, j)]_{i,j} \in \mathbb{R}^{n \times n}$. The degree matrix is the diagonal matrix $\mathsf{D} \in \mathbb{R}^{n \times n}$ with $\mathsf{D}_{i,i} = \sum_{j \in \mathcal{N}(i)} w(i, j)$ and $\mathcal{N}(i)$ is the set of nodes connected to i. The graph Laplacian matrix is defined as $\mathsf{L} = \mathsf{D} - \mathsf{W}$. It is semi-positive definite [17] and its eigenvalues and eigenvectors are denoted by $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ and $\mathsf{U} = (\mathbf{u}_1 | \mathbf{u}_2 | \ldots | \mathbf{u}_n)$, respectively. The multiplicity of eigenvalue 0 equals the number of connected components in

[1] https://gricad-gitlab.univ-grenoble-alpes.fr/pilavciy/rsf_sipn_codes.git

the graph [17]. For undirected graphs, all of these square matrices are symmetric.

Another way to represent graphs is via the edge incidence matrix $\mathsf{B} = (\mathbf{b}_1|\mathbf{b}_2|\dots|\mathbf{b}_m)^\top \in \mathbb{R}^{m\times n}$ where $\mathbf{b}_k \in \mathbb{R}^n$ is a vector associated to the k-th edge (i,j). The only nonzero entries of $\mathbf{b}_k$ are $\mathbf{b}_k(i) = \pm\sqrt{w(i,j)}$ and $\mathbf{b}_k(j) = \mp\sqrt{w(i,j)}$. In undirected graphs, the signs of the non-zero entries of $\mathbf{b}_k$ can be arbitrarily chosen as long as they are opposite. Although this matrix seems less natural than the others, it often appears in graph theory. One example is the well known identity $\mathsf{L} = \mathsf{B}^\top\mathsf{B}$.

B. Random spanning trees and Wilson's algorithm

Let us recall the definition of random spanning trees (RSTs). Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$. A subgraph of $\mathcal{G}$ is a graph whose vertex and edge sets are subsets of $\mathcal{V}$ and $\mathcal{E}$, respectively, and its edge weights are valued by w. A subgraph contains a cycle whenever there exists a pair of vertices (u, v) that are connected via (strictly) more than one path. If there exists no such pair, the subgraph is called a tree. A spanning tree $\tau = (\mathcal{V}_\tau, \mathcal{E}_\tau, w_\tau)$ is a tree whose vertex set $\mathcal{V}_\tau$ is equal to $\mathcal{V}$. A rooted spanning tree τ_r is a directed spanning tree where all edges are directed towards a node called the root. See Fig. 2 for illustrations.

Our work is related to a particular distribution on spanning trees called random spanning trees (RSTs). An RST T is a randomly generated spanning tree from the following distribution:

$$\mathbb{P}(T = \tau) \propto \prod_{(i,j)\in\tau} w(i,j). \tag{1}$$

Note that this distribution becomes uniform over all possible spanning trees whenever the given graph is unweighted *i.e.* $\forall i,j \in \mathcal{V}$, $w(i,j) \in \{0,1\}$:

$$\mathbb{P}(T = \tau) = \frac{1}{|\mathcal{T}|} \tag{2}$$

where $\mathcal{T}$ is the set of all spanning trees. In this particular case, the random tree T is also known as a uniform spanning tree (UST) in the literature.

In his celebrated work [24], Wilson proposes an algorithm, called `RandomTreeWithRoot`, that samples a random spanning tree from the set of all spanning trees rooted in node r. Wilson also shows that, in the case of undirected graphs, sampling an unrooted RST amounts to: i/ choosing uniformly a root, ii/ running `RandomTreeWithRoot`, and iii/ erasing the orientation.

C. Random spanning forests

A forest is a set of disjoint trees. When all the trees in a forest are rooted, it is called a rooted forest. A rooted spanning forest, generically denoted by ϕ, reaches all the vertices in the graph. Let ρ be the function that maps any rooted spanning forests to its set of roots. The number of roots $|\rho(\phi)|$ is between 1 and n. For $|\rho(\phi)| = 1$, ϕ corresponds to a rooted spanning tree. See Fig. 2 for illustrations.

Random Spanning Forests. Let $\mathcal{F}$ be the set of all rooted spanning forests. A random spanning forest (RSF) is a random variable whose outcome space is $\mathcal{F}$. Among many possible options, we focus on the following parametric distribution for RSFs. For a fixed parameter $q > 0$, Φ_q is a random variable in $\mathcal{F}$ verifying:

$$\forall \phi \in \mathcal{F}, \quad \mathbb{P}(\Phi_q = \phi) \propto q^{|\rho(\phi)|} \prod_{(i,j)\in\phi} w(i,j). \tag{3}$$

An algorithm [23] to sample from this distribution is derived from `RandomTreeWithRoot`. This algorithm:

1) extends the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ by adding a node called Γ.
2) connects each node i in $\mathcal{V}$ to Γ with an edge of weight $w(i,\Gamma) = q$.
3) runs `RandomTreeWithRoot` by setting Γ as the root to obtain a spanning tree rooted in Γ in the extended graph.
4) deletes the edges incident to Γ in the obtained tree to yield a forest in the original graph.

The result is a rooted spanning forest whose root set is formed by the nodes which were neighbors of Γ. For every distinct spanning tree rooted at Γ, a distinct spanning forest is obtained after removing the root and its incident edges. Using this one-to-one relation ensures that this algorithm indeed samples from the distribution in Eq. (3):

$$\begin{aligned}\mathbb{P}(T_\Gamma = \tau_\Gamma) = \mathbb{P}(\Phi_q = \phi) &\propto \prod_{(i,j)\in\tau_\Gamma} w(i,j) \\ &\propto \prod_{(i,\Gamma)\in\tau_\Gamma} q \prod_{\substack{(i,j)\in\tau_\Gamma \\ i,j\neq\Gamma}} w(i,j) \\ &\propto q^{|\rho(\phi)|} \prod_{(i,j)\in\phi} w(i,j).\end{aligned} \tag{4}$$

An implementation of this algorithm is detailed in Algorithm 1. In the algorithm, `rand` (line 7) returns a uniform random value between 0 and 1 and `RandomSuccessor` (line 11) returns a random node i from $\mathcal{N}(u)$ with probability $\frac{w(u,i)}{\sum_{j\in\mathcal{N}(u)} w(u,j)}$. At termination, the array Next contains all the necessary information to build the sampled spanning forest.

The expected run time of `RandomForest` is the expected number of calls of `RandomSuccessor` before termination. For `RandomTreeWithRoot`, the number of calls equals to the mean commute time, *i.e.*, the expected length of a random walk going from node i to j and back (see theorem 2 in [24]). In proposition 1 of [26], Marchal rewrites this commute time in terms of graph matrices. Adapting his result to the current setting, the expected run time of `RandomForest` can be shown to equal the trace of $\big((\mathsf{L} + q\mathsf{I})^{-1}(\mathsf{D} + q\mathsf{I})\big)$, a rough upper-bound of which is $n + 2|\mathcal{E}|/q$, which is linear with the number of edges.

Varying q over nodes. The original graph can also be extended by setting[2] $w(i,\Gamma) \leftarrow q_i \geq 0, \forall i \in \mathcal{V}$, that is, by

[2]A requirement when choosing the q_i's is that the extended graph stays connected: when q is chosen uniform, it thus has to be chosen strictly positive. When q is not chosen uniform, each q_i can be chosen positive or null as long as the extended graph is connected (for instance, all q_i's cannot be null)

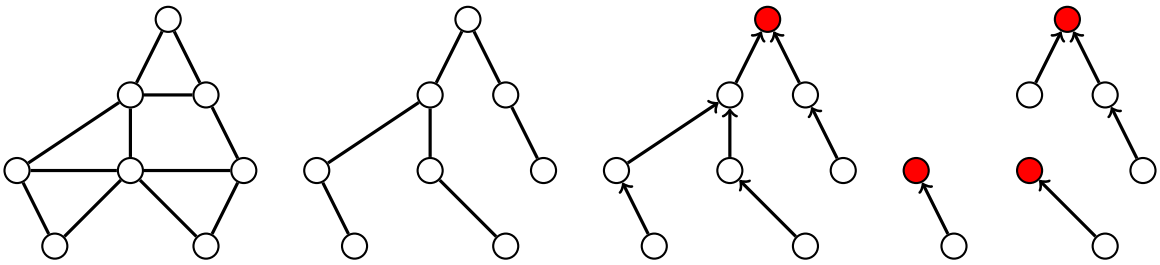


Fig. 2: From left to right, a graph $\mathcal{G}$, a spanning tree on $\mathcal{G}$, a rooted spanning tree on $\mathcal{G}$ and a rooted spanning forest on $\mathcal{G}$ (roots are colored in red)

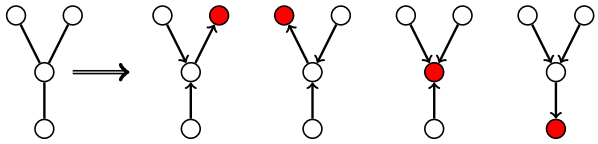


Fig. 3: All possible rooted spanning trees associated with a given undirected spanning tree. For four vertices, four different rooted trees exist.

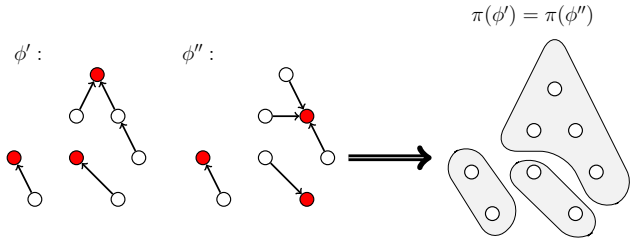


Fig. 4: Two different rooted spanning forests (on the left) with the same corresponding partition (on the right)

Algorithm 1 `RandomForest`

Inputs:
  $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$
  $q > 0$
Initialize:
  # Initially, the forest is empty
  $\forall i \in \mathcal{V}$, InForest$[i] \leftarrow$ `false`
  $\forall i \in \mathcal{V}$, Next[i] $\leftarrow -1$
  $\forall i \in \mathcal{V}$, d$[i] \leftarrow \sum_{j \in \mathcal{N}(i)} w(i,j)$ *# Degrees*
for $i \leftarrow 1$ to $|\mathcal{V}|$ **do**
  $u \leftarrow i$
  # Start a random walk to create a forest branch
  while not InForest$[u]$ **do** *# Stop if u is in the forest*
    if `rand` $\leq \frac{q}{q+\text{d}[u]}$ **then** *#If true, u becomes a root*
      InForest$[u] \leftarrow$ `true` *# Add u to the forest*
      Next$[u] \leftarrow -1$ *# Set next of u to null*
    else *# If false, continue the random walk*
      Next[u] $\leftarrow$ `RandomSuccessor`$(u, \mathcal{G})$
      $u \leftarrow$ Next$[u]$
    end if
  end while
  $u \leftarrow i$ *# Go back to the initial node*
  # Add the newly created branch to the forest
  while not InForest$[u]$ **do**
    InForest$[u] \leftarrow$ `true`
    $u \leftarrow$ Next$[u]$
  end while
end for
`return` Next

connecting the added node Γ with links of unequal weights. In this case, the distribution of sampled forests becomes:

$$\mathbb{P}(\Phi_Q = \phi) \propto \prod_{i \in \rho(\phi)} q_i \prod_{(i,j) \in \phi} w(i,j), \quad \phi \in \mathcal{F} \tag{5}$$

where $Q = \{q_1, q_2, \ldots, q_n\}$ is the collection of parameters. Algorithm 1 can easily be adapted by modifying the scalar input q to $Q = \{q_1, q_2, \ldots, q_n\}$ and $\frac{q}{q+d[u]}$ to $\frac{q_u}{q_u+d[u]}$ at line 7. In addition, the average run time in this case becomes $\text{tr}\left((\mathsf{L}+\mathsf{Q})^{-1}(\mathsf{D}+\mathsf{Q})\right)$, with $\mathsf{Q} = \text{diag}(q_1, \ldots, q_n)$.

Random partitions. A partition of $\mathcal{V}$, denoted by $\mathcal{P}$, is a set of disjoint subsets whose union equals $\mathcal{V}$. The trees of Φ_q give a random partition of $\mathcal{V}$ by splitting it into $|\rho(\Phi_q)|$ disjoint subsets. Let us enumerate the trees from 1 to $|\rho(\Phi_q)|$ and denote the vertex set of the k-th tree as $\mathcal{V}_k \subset \mathcal{V}$. Let π be a function that outputs the partition for a given spanning forest. Then, the random partition of $\mathcal{V}$ derived from Φ_q is $\pi(\Phi_q) = (\mathcal{V}_1, \ldots, \mathcal{V}_{|\rho(\Phi_q)|})$ with $|\pi(\Phi_q)|$ subsets. Note that this function is a many-to-one mapping because different spanning forests may correspond to the same partition (see Figure 4).

D. Useful properties of Φ_q

Recent studies in [22], [23] have established some theoretical properties of Φ_q that we reproduce here.

The root process. To start with, Proposition 2.2 in [23] states that $\rho(\Phi_q)$ is sampled from a determinantal point process (DPP) [27] with marginal kernel:

$$\mathsf{K} = (q\mathsf{I} + \mathsf{L})^{-1} q\mathsf{I}. \tag{6}$$

This means that the inclusion probabilities verify:

$$\forall \mathcal{S} \subset \mathcal{V}, \quad \mathbb{P}(\mathcal{S} \in \rho(\Phi_q)) = \det \mathsf{K}_{\mathcal{S}}$$

where $\mathsf{K}_{\mathcal{S}} = [\mathsf{K}_{i,j}|(i,j) \in \mathcal{S} \times \mathcal{S}]$ is the submatrix of K reduced to the rows and columns indexed by $\mathcal{S}$.

Cardinality of $\rho(\Phi_q)$. As a consequence of $\rho(\Phi_q)$ being a DPP, the first two moments of $|\rho(\Phi_q)|$ verify [27]:

$$\begin{aligned} \mathbb{E}[|\rho(\Phi_q)|] &= \mathrm{tr}(\mathsf{K}) = \sum_i \frac{q}{q+\lambda_i}, \\ \mathrm{Var}(|\rho(\Phi_q)|) &= \mathrm{tr}(\mathsf{K}-\mathsf{K}^2) = \sum_i \frac{\lambda_i q}{(q+\lambda_i)^2} \end{aligned} \tag{7}$$

where the λ_i's are the eigenvalues of L.

The root probability distribution. Given any rooted spanning forest ϕ, define the *root function* $r_\phi : \mathcal{V} \to \rho(\phi)$ which takes as input any node i and outputs the root of the tree which i belongs to. In [22], [23], the authors show that the probability, for any node pair (i,j), that i is rooted in j reads:

$$\forall i,j \in \mathcal{V} \quad \mathbb{P}(r_{\Phi_q}(i) = j) = \mathsf{K}_{ij}. \tag{8}$$

Conditioning on a partition. Let $t : \mathcal{V} \to \{1,2,\dots,|\rho(\Phi_q)|\}$ be a random mapping between any node and its tree number in Φ_q. (*e.g.*, $t(i) = k$ if $i \in \mathcal{V}_k \in \pi(\Phi_q)$). By conditioning the root probability over a fixed partition $\mathcal{P}$, one obtains (see Proposition 2.3 in [23]):

$$\forall i,j \in \mathcal{V} \quad \mathbb{P}(r_{\Phi_q}(i) = j | \pi(\Phi_q) = \mathcal{P}) = \frac{\mathbb{I}(j \in \mathcal{V}_{t(i)})}{|\mathcal{V}_{t(i)}|} \tag{9}$$

where $\mathbb{I}$ is the indicator function (*i.e.*, it outputs 1 if the input statement is true and 0 otherwise). In other words, given a fixed partition $\mathcal{P}$, the root probability within each subset $\mathcal{V}_k$ is uniform over the nodes in $\mathcal{V}_k$.

Extending to non-constant q. All of these properties are adaptable to the case of q varying over nodes (with some changes). The root process $\rho(\Phi_Q)$ is also a DPP. However, the associated marginal kernel becomes:

$$\mathsf{K} = (\mathsf{L}+\mathsf{Q})^{-1}\mathsf{Q} \quad \text{with} \quad \mathsf{Q} = \mathrm{diag}(q_1,\dots,q_n). \tag{10}$$

Notice that this kernel is not co-diagonalizable with the graph Laplacian L. Thus, the expected number of roots $\mathbb{E}[|\rho(\Phi_q)|]$ is not writable in terms of λ_i's, but it is still equal to $\mathrm{tr}(\mathsf{K})$. Similarly, the root probability $\mathbb{P}(r_{\Phi_Q}(i) = j)$ remains $\mathsf{K}_{i,j}$ whereas the conditional probability in Eq. (9) becomes:

$$\forall i,j \in \mathcal{V} \quad \mathbb{P}(r_{\Phi_Q}(i) = j | \pi(\Phi_Q) = \mathcal{P}) = \frac{q_j \mathbb{I}(j \in \mathcal{V}_{t(i)})}{\sum_{k \in \mathcal{V}_{t(i)}} q_k}. \tag{11}$$

III. RSF BASED ESTIMATORS

In this section, we present our main results. We first recall the graph Tikhonov regularization and interpolation problems. Then, we describe the RSF-based methods to solve them. We also provide some theoretical analysis of the performance of the methods. Finally, we show how to tune hyperparameters for the proposed estimators.

Graph Tikhonov regularization. For a given graph $\mathcal{G} = (\mathcal{V},\mathcal{E},w)$ and measurements $\mathbf{y} = (y_1,y_2,\dots,y_n)^\top$ on the $|\mathcal{V}| = n$ vertices, the Tikhonov regularization of $\mathbf{y}$ reads:

$$\hat{\mathbf{x}} = \underset{\mathbf{z}\in\mathbb{R}^n}{\mathrm{argmin}}\, \mu||\mathbf{y}-\mathbf{z}||^2 + \mathbf{z}^\top \mathsf{L}\mathbf{z} \tag{12}$$

where $\mathsf{L} \in \mathbb{R}^{n\times n}$ is the graph Laplacian of $\mathcal{G}$. The solution of this minimization problem is:

$$\hat{\mathbf{x}} = \mathsf{K}\mathbf{y} \text{ with } \mathsf{K} = (\mathsf{L}+\mu\mathsf{I})^{-1}\mu\mathsf{I}. \tag{13}$$

Interestingly, the matrix in this solution also appears in Eq. (6) as the marginal kernel of the root process. This correspondence plays a significant role for the proposed methods by connecting RSFs to the Tikhonov regularization problem.

In some important cases, instead of $(\mathsf{L}+\mu\mathsf{I})^{-1}\mu\mathsf{I}\,\mathbf{y}$, the generalized solution $(\mathsf{L}+\mathsf{Q})^{-1}\mathsf{Q}\,\mathbf{y}$ is required where Q is an entry-wise non-negative diagonal matrix. For example, if we write the Tikhonov regularization of Eq. (12) with another graph Laplacian such as the random walk Laplacian $\mathsf{L}_{rw} = \mathsf{D}^{-1}\mathsf{L}$, then the solution reads $\hat{\mathbf{x}} = (\mathsf{L}+\mathsf{Q})^{-1}\mathsf{Q}\,\mathbf{y}$ where $\mathsf{Q} = \mu\mathsf{D}$. Another example occurs when the noise variance is known to be non-constant over vertices, *i.e.* heteroscedastic noise. The measurements may be less reliable at some vertices compared to others, meaning that there are different noise variances $\sigma_1,\dots,\sigma_n$. This implies that $q_1\dots,q_n$ should be set proportional to $\frac{1}{\sigma_1},\dots,\frac{1}{\sigma_n}$ in the estimation of $\hat{\mathbf{x}}$. This again corresponds to the generalized formulation.

Graph interpolation. Given a connected graph $\mathcal{G} = (\mathcal{V},\mathcal{E},w)$, a parameter $\mu \geq 0$, and $\ell \subset \mathcal{V}$ a set of nodes where a signal $\mathbf{x}$ is known, one way of defining the interpolated signal is [8]:

$$\begin{aligned} \hat{\mathbf{x}} = \underset{z\in\mathbb{R}^n}{\mathrm{argmin}}\, &\mathbf{z}^\top(\mathsf{L}+\mu\mathsf{I})\mathbf{z} \\ \text{subject to } &\forall i \in \ell,\ z_i = x_i. \end{aligned} \tag{14}$$

Define $u = \mathcal{V}\backslash\ell$ the set of nodes for which $\mathbf{x}$ is not known and write L in block form:

$$\mathsf{L} = \begin{bmatrix} \mathsf{L}_{\ell|\ell} & \mathsf{L}_{\ell|u} \\ \mathsf{L}_{u|\ell} & \mathsf{L}_{u|u} \end{bmatrix}$$

where $\mathsf{L}_{\ell|u}$ is the Laplacian reduced to its rows and columns indexed by ℓ and u, respectively. The solution of Eq. (14) reads:

$$\hat{\mathbf{x}} = \begin{cases} x_i & \text{if } i \in \ell \\ \left(-(\mathsf{L}_{u|u}+\mu\mathsf{I})^{-1}\,\mathsf{L}_{u|l}\,\mathbf{x}_\ell\right)_i & \text{otherwise} \end{cases} \tag{15}$$

where $\mathbf{x}_\ell \in \mathbb{R}^{|\ell|}$ is the signal $\mathbf{x}$ reduced to its entries in ℓ. This solution can almost always[3] be rewritten as:

$$\hat{\mathbf{x}}_u = \mathsf{K}\mathbf{y} \quad \text{with} \quad \begin{cases} \mathsf{K} = (\mathsf{L}_{\mathcal{G}\backslash\ell}+\mathsf{Q})^{-1}\mathsf{Q} \\ \mathbf{y} = -\mathsf{Q}^{-1}\mathsf{L}_{u|\ell}\mathbf{x}_\ell \end{cases} \tag{16}$$

where $\mathsf{L}_{\mathcal{G}\backslash\ell}$ is the Laplacian of the reduced graph obtained by removing the vertices (and the incident edges) in ℓ, $\mathsf{Q} \in \mathbb{R}^{|u|\times|u|}$ is a diagonal matrix with

[3] Q, as defined in the paragraph following Eq. (16), needs to be invertible for $\mathbf{y}$ to be well-defined. This is always the case if $\mu > 0$. When $\mu = 0$, it may not be the case. We will see in Section IV-A what can be done in this scenario.

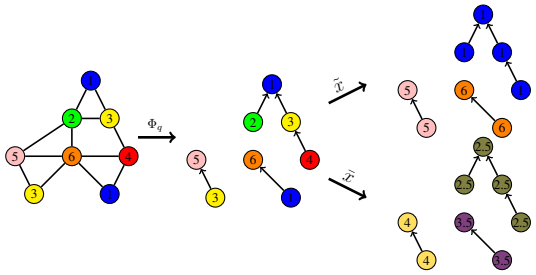


Fig. 5: An illustration for the estimators where q is constant over all nodes. In the left, the graph signal is interpreted by both colors and numbers. In the middle, a realization of Φ_q, a forest, is illustrated. On this forest, the estimators $\tilde{\mathbf{x}}$ and $\bar{\mathbf{x}}$ are illustrated in top-right and bottom right, respectively.

$\mathsf{Q}_{i,i} = \mu + \sum_{j\in\ell} w(i,j)$. Similarly to graph TR, the RSF-based estimator for interpolation proposed in this paper draws upon the connection between Eqs. (16) and (10).

Parameter selection. The solution to graph TR Eq. (13) tends to the constant vector (equal to the average of $\mathbf{y}$) as $\mu \to 0$, and to $\mathbf{y}$ for $\mu \to \infty$, where it suffers from underfitting and overfitting, respectively. In the interpolation problem, as $\mu \to \infty$, no prior information gets propagated through the other vertices, and $\hat{\mathbf{x}}_u$ tends to the zero vector. The case of $\mu = 0$ corresponds to solving the Dirichlet problem [10] which does not necessarily give the closest inference to the original signal $\mathbf{x}$. Due to these reasons, μ needs to be set at a value that gives the best approximation to the original signal. In both problems, choosing μ is a classical hyperparameter selection problem which can be approached in several ways for the proposed estimators.

In the following, we first present the proposed estimators for approximating $\hat{\mathbf{x}}$ for a fixed value of μ in Section III-A. Then, we outline in Section III-B several methods that select an appropriate μ automatically. Combining the hyperparameter selection and the RSF based estimators forms an RSF-based framework to approximate the solutions of graph Tikhonov regularization and graph interpolation.

A. RSF based estimation of $\hat{\mathbf{x}} = \mathsf{K}\mathbf{y}$

We propose two novel Monte Carlo estimators to approximate $\hat{\mathbf{x}} = \mathsf{K}\mathbf{y}$ with $\mathsf{K} = (\mathsf{L}+\mathsf{Q})^{-1}\mathsf{Q}$. These estimators leverage the probability distribution of the root process on RSFs presented in Eq. (8).

The first estimator, denoted by $\tilde{\mathbf{x}}$, is defined as follows:

$$\forall i \in \mathcal{V} \quad \tilde{x}(i) = y(r_{\Phi_Q}(i)). \tag{17}$$

In practice, a realization of Φ_Q is considered. Then, in each tree, the measurement of the root is propagated through the nodes of the tree. (See top-right in Fig. 5).

Proposition 1. *$\tilde{\mathbf{x}}$ is an unbiased estimator of $\hat{\mathbf{x}}$:*

$$\mathbb{E}\left[\tilde{\mathbf{x}}\right] = \hat{\mathbf{x}}.$$

Moreover, the weighted expected error of $\tilde{\mathbf{x}}$ is:

$$\mathbb{E}\big(||\hat{\mathbf{x}} - \tilde{\mathbf{x}}||^2_{\mathsf{Q}}\big) = \sum_{i\in\mathcal{V}} q_i \operatorname{Var}(\tilde{x}(i)) = \mathbf{y}^\top(\mathsf{Q} - \mathsf{K}^\top\mathsf{Q}\mathsf{K})\mathbf{y}$$

where $||\mathbf{x}||^2_{\mathsf{Q}} = \mathbf{x}^\top\mathsf{Q}\mathbf{x}$.

Proof. For every node i, $\tilde{x}(i)$ is an unbiased estimator of $\hat{x}(i)$ thanks to the following:

$$\begin{aligned}\mathbb{E}\left[\tilde{x}(i)\right] = \mathbb{E}\left[y(r_{\Phi_Q}(i))\right] &= \sum_j \mathbb{P}(r_{\Phi_Q}(i) = j)y(j) \\ &= \sum_j \mathsf{K}_{ij}y(j) = \boldsymbol{\delta}_i^\top\mathsf{K}\mathbf{y} = \hat{x}(i)\end{aligned}$$

where δ_i is the Kronecker delta (*i.e.* $\delta_i(i) = 1$ and 0 otherwise). This result is prominently due to the root probability of RSFs given in Eq. (8). Also, the variance of $\tilde{x}(i)$ reads:

$$\operatorname{Var}(\tilde{x}(i)) = \mathbb{E}\left[\tilde{x}(i)^2\right] - \mathbb{E}\left[\tilde{x}(i)\right]^2 = \boldsymbol{\delta}_i^\top\mathsf{K}\mathbf{y}^{(2)} - (\boldsymbol{\delta}_i^\top\mathsf{K}\mathbf{y})^2.$$

where $y^{(2)}(k) = y(k)^2$, $\forall k \in \mathcal{V}$. Then, the weighted sum reads:

$$\sum_{i\in\mathcal{V}} q_i \operatorname{Var}(\tilde{x}(i)) = \mathbf{1}^\top\mathsf{Q}\mathsf{K}\mathbf{y}^{(2)} - \mathbf{y}^\top\mathsf{K}^\top\mathsf{Q}\mathsf{K}\mathbf{y}$$

where $\mathbf{1}$ denotes the all-ones vector. Note that $\mathbf{1}^\top\mathsf{Q}\mathsf{K} = \mathbf{1}^\top\mathsf{K}^\top\mathsf{Q}$. Moreover, $\mathbf{1}$ is a left eigenvector of $\mathsf{K}^\top$ with corresponding eigenvalue 1. Then, the first term becomes $\mathbf{1}^\top\mathsf{K}^\top\mathsf{Q}\mathbf{y}^{(2)} = \mathbf{1}^\top\mathsf{Q}\mathbf{y}^{(2)} = \mathbf{y}^\top\mathsf{Q}\mathbf{y}$, and, one obtains:

$$\sum_{i\in\mathcal{V}} q_i \operatorname{Var}(\tilde{x}(i)) = \mathbf{y}^\top(\mathsf{Q} - \mathsf{K}^\top\mathsf{Q}\mathsf{K})\mathbf{y}. \tag{18}$$

□

The second estimator, denoted by $\bar{\mathbf{x}}$, is the expectation of $\tilde{\mathbf{x}}$ conditioned on the partition induced by Φ_Q:

$$\bar{x}(i) = \mathbb{E}[\tilde{x}(i)|\pi(\Phi_Q) = \mathcal{P}] = \frac{\sum\limits_{j\in\mathcal{V}_{t(i)}} y(j)q_j}{\sum\limits_{j\in\mathcal{V}_{t(i)}} q_j}. \tag{19}$$

Due to the law of iterated expectations, this estimator is also unbiased, moreover it has a reduced variance compared to $\tilde{x}(i)$ due to the law of total variance:

$$\operatorname{Var}(\tilde{x}(i)) = \mathbb{E}[\operatorname{Var}(\tilde{x}(i)|\pi(\Phi_Q) = \mathcal{P})] + \operatorname{Var}(\bar{x}(i))$$

which implies $\operatorname{Var}(\tilde{x}(i)) \geq \operatorname{Var}(\bar{x}(i))$. This idea of improving an estimator is often called Rao-Blackwellization [28], [29].

In practice, we again take a realization of Φ_Q and consider the corresponding partition $\pi(\Phi_Q)$. Then, we compute the weighted average of the measurements in each subset of $\pi(\Phi_Q)$. Then, we finally propagate these averages in each subset (see Fig. 5).

Proposition 2. *$\bar{\mathbf{x}}$ is an unbiased estimator of $\hat{\mathbf{x}}$:*

$$\mathbb{E}[\bar{\mathbf{x}}] = \hat{\mathbf{x}}$$

Moreover, the weighted expected error reads:

$$\mathbb{E}\left(||\hat{\mathbf{x}} - \bar{\mathbf{x}}||^2_{\mathsf{Q}}\right) = \sum_{i\in\mathcal{V}} q_i \operatorname{Var}(\bar{x}(i)) = \mathbf{y}^\top(\mathsf{Q}\mathsf{K} - \mathsf{K}^\top\mathsf{Q}\mathsf{K})\mathbf{y}.$$

Proof. Let $\mathsf{S} = \left[\frac{q_j\mathbb{I}(j\in\mathcal{V}_{t(i)})}{\sum_{k\in\mathcal{V}_{t(i)}} q_k}\right]_{i,j}$ be a symmetric random matrix associated to the random partition $\pi(\Phi_Q)$. A simple matrix product shows that $\mathsf{S}^\top\mathsf{Q}\mathsf{S} = \mathsf{Q}\mathsf{S}$ by definition of S. Moreover,

$\bar{x}(i) = \boldsymbol{\delta}_i^\top \mathsf{S}\mathbf{y}$ and the expectation of $\mathsf{S}_{i,j}$ over all possible partitions derived from Φ_Q is:

$$\begin{aligned}\mathbb{E}\left[\mathsf{S}_{i,j}\right] &= \sum_{\mathcal{P}\in\pi(\mathcal{F})} \frac{q_j \mathbb{I}(j \in \mathcal{V}_{t(i)})}{\sum\limits_{k\in\mathcal{V}_{t(i)}} q_k} \mathbb{P}(\pi(\Phi_Q) = \mathcal{P}) \\ &= \sum_{\mathcal{P}\in\pi(\mathcal{F})} \mathbb{P}(r_{\Phi_Q}(i) = j | \pi(\Phi_Q) = \mathcal{P}) \mathbb{P}(\pi(\Phi_Q) = \mathcal{P}) \\ &= \mathbb{P}(r_{\Phi_Q}(i) = j) = \mathsf{K}_{i,j}.\end{aligned} \tag{20}$$

Similarly, the expectation of $\bar{x}(i)$ reads:

$$\mathbb{E}[\bar{x}(i)] = \mathbb{E}[\boldsymbol{\delta}_i^\top \mathsf{S}\mathbf{y}] = \boldsymbol{\delta}_i^\top \mathbb{E}[\mathsf{S}]\mathbf{y} = \boldsymbol{\delta}_i^\top \mathsf{K}\mathbf{y} = \hat{x}(i). \tag{21}$$

Thus, $\bar{\mathbf{x}}$ is unbiased. The expected error is also computed in a similar way:

$$\begin{aligned}\mathbb{E}\left(||\hat{\mathbf{x}} - \bar{\mathbf{x}}||_{\mathsf{Q}}^2\right) &= \sum_{i\in\mathcal{V}} q_i \operatorname{Var}(\bar{x}(i)) \\ &= \sum_{i\in\mathcal{V}} q_i \left(\mathbb{E}[(\boldsymbol{\delta}_i^\top \mathsf{S}\mathbf{y})^2] - \mathbb{E}[(\boldsymbol{\delta}_i^\top \mathsf{S}\mathbf{y})]^2\right) \\ &= \sum_{i\in\mathcal{V}} \mathbf{y}^\top \mathbb{E}\left[q_i \mathsf{S}^\top \boldsymbol{\delta}_i \boldsymbol{\delta}_i^\top \mathsf{S}\right] \mathbf{y} - q_i \mathbf{y}^\top (\boldsymbol{\delta}_i^\top \mathbb{E}[\mathsf{S}])^2 \mathbf{y} \\ &= \mathbf{y}^\top \mathbb{E}\left[\mathsf{S}^\top \mathsf{Q}\mathsf{S}\right] \mathbf{y} - \mathbf{y}^\top (\mathbb{E}[\mathsf{S}]^\top \mathsf{Q} \mathbb{E}[\mathsf{S}])\mathbf{y} \\ &= \mathbf{y}^\top (\mathbb{E}\left[\mathsf{S}^\top \mathsf{Q}\mathsf{S}\right] - \mathsf{K}^\top \mathsf{Q}\mathsf{K})\mathbf{y}.\end{aligned} \tag{22}$$

Finally, rewriting $\mathsf{S}^\top \mathsf{Q}\mathsf{S} = \mathsf{Q}\mathsf{S}$, one has:

$$\mathbb{E}\left(||\hat{\mathbf{x}} - \bar{\mathbf{x}}||_{\mathsf{Q}}^2\right) = \mathbf{y}^\top (\mathbb{E}\left[\mathsf{Q}\mathsf{S}\right] - \mathsf{K}^\top \mathsf{Q}\mathsf{K})\mathbf{y} = \mathbf{y}^\top (\mathsf{Q}\mathsf{K} - \mathsf{K}^\top \mathsf{Q}\mathsf{K})\mathbf{y}.$$

□

Sample Mean. The sample mean of an unbiased Monte Carlo estimator over different realizations has a reduced variance, and so, gives a better estimator. Thus, in the rest, we use the sample means $\frac{1}{N}\sum_{k=1}^N \tilde{\mathbf{x}}_{\Phi_Q^{(k)}}$ and $\frac{1}{N}\sum_{k=1}^N \bar{\mathbf{x}}_{\Phi_Q^{(k)}}$ over N forest realizations as the outputs of the RSF based methods.

A remark. Reducing these results to the constant q case ($\mathsf{Q} = q\mathsf{I}$), one recovers the preliminary results presented in [25].

B. Parameter selection for the RSF estimators

The proposed estimators are efficient tools to approximate $\hat{\mathbf{x}}$ in both graph TR and interpolation problems for a fixed value of μ. However, as usual in these problems, a difficult question is the tuning of the hyper-parameter: the choice of μ that yields the best performance. For linear smoothers such as the one we have at hand ($\hat{\mathbf{x}} = \mathsf{K}\mathbf{y}$), many methods such as Akaike information criterion (AIC), Bayesian information criterion (BIC), Marlow's Cp, leave-one-out cross validation (LOOCV), generalized cross validation (GCV) or Stein's unbiased risk estimator (SURE) are readily available[4] for this tuning step (for more details and motivations, we refer the reader to [9]).

[4]Note that the theoretical framework used to justify these various criteria does not directly apply to graphs, because large-n asymptotics are not immediately well-defined in graphs. One may appeal to random graph assumptions, as in [30], but a detailed argument is beyond the scope of this paper.

All of these methods need to compute a quantity called the effective number of parameters or the degree of freedom [9], which equals $\operatorname{tr}(\mathsf{K})$ for linear smoothers of the form $\mathsf{K}\mathbf{y}$. Computing exactly this trace requires the matrix inversion we wish to avoid from the start. A classical estimator of this quantity is Girard's estimator [31] (also known as Hutchinson's estimator [32]). We showed in [33] that RSFs can also be used to efficiently estimate $\operatorname{tr}(\mathsf{K})$. In this section, we build upon these preliminary results to show how the SURE and LOOCV methods can be adapted to the proposed estimators in order to select a good value of μ. Other methods are adaptable in a similar fashion.

Stein's Unbiased Risk Estimator. Given independent noisy measurements $\mathbf{y} = \mathbf{x} + \boldsymbol{\epsilon} \in \mathbb{R}^n$ with a Gaussian noise $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, let $\theta(\mathbf{y})$ be an estimate for the unknown quantity $\mathbf{x}$. $\texttt{SURE}(\mathbf{y}, \theta(\mathbf{y}))$ provides an unbiased estimate of the expected error $\mathbb{E}_{\boldsymbol{\epsilon}}[||\theta(\mathbf{y}) - \mathbf{x}||_2^2]$. For the linear smoother $\theta(\mathbf{y}) = \mathsf{K}\mathbf{y}$ with $\mathsf{K} = (\mathsf{Q} + \mathsf{L})^{-1}\mathsf{Q}$, the generic formula of SURE in [34] can be adapted as:

$$\texttt{SURE}(\mathbf{y}, \theta(\mathbf{y})) = -n\sigma^2 + ||\mathbf{y} - \theta(\mathbf{y})||_2^2 + 2\sigma^2 \operatorname{tr}(\mathsf{K}) \tag{23}$$

where the degree of freedom term is replaced with $\operatorname{tr}(\mathsf{K})$. The theory behind relies on Stein's lemma on multivariate Gaussians [35]. Note that this method requires prior knowledge on the noise variance σ^2 and it outputs an unbiased estimation of the error. Then, this error needs to be evaluated for different values of μ and select the value yielding the smallest error.

SURE for RSF estimators. Similar to $\hat{\mathbf{x}}$, the RSF estimators benefit from optimising the value of μ. For this purpose, SURE can be used. In the following derivations, we present the adapted SURE formula for $\tilde{\mathbf{x}}$ and $\bar{\mathbf{x}}$. Moreover, these derivations show that numerically computing this formula is trivial after sampling N spanning forests.

Consider two random matrices $\tilde{\mathsf{S}} = \left[\mathbb{I}(r_{\Phi_Q}(i) = j)\right]_{i,j}$ and $\bar{\mathsf{S}} = \left[\frac{q_j \mathbb{I}(j\in\mathcal{V}_{t(i)})}{\sum\limits_{k\in\mathcal{V}_{t(i)}} q_k}\right]_{i,j}$ (previously defined as S in the proof of Prop. 2). With these definitions, notice that $\tilde{\mathbf{x}} = \tilde{\mathsf{S}}\mathbf{y}$ and $\bar{\mathbf{x}} = \bar{\mathsf{S}}\mathbf{y}$. Moreover, the proposed estimators can be written in the form of linear smoothers:

$$\begin{aligned}\tilde{\theta}(\mathbf{y}) &= \frac{1}{N}\sum_{k=1}^N \tilde{\mathbf{x}}_{\Phi_Q^{(k)}} = \frac{1}{N}\sum_{k=1}^N \tilde{\mathsf{S}}^{(k)}\mathbf{y}, \\ \bar{\theta}(\mathbf{y}) &= \frac{1}{N}\sum_{k=1}^N \bar{\mathbf{x}}_{\Phi_Q^{(k)}} = \frac{1}{N}\sum_{k=1}^N \bar{\mathsf{S}}^{(k)}\mathbf{y}\end{aligned} \tag{24}$$

where superscript (k) denotes k-th realization of $\tilde{\mathsf{S}}$ or $\bar{\mathsf{S}}$.

Then, one can also evaluate the formula in Eq. (23) for $\tilde{\theta}(\mathbf{y})$ and $\bar{\theta}(\mathbf{y})$. For instance, we get for $\tilde{\theta}(\mathbf{y})$:

$$\texttt{SURE}(\mathbf{y}, \tilde{\theta}(\mathbf{y})) = -n\sigma^2 + ||\mathbf{y} - \tilde{\theta}(\mathbf{y})||_2^2 + 2\sigma^2 \operatorname{tr}\left(\frac{1}{N}\sum_{k=1}^N \tilde{\mathsf{S}}^{(k)}\right). \tag{25}$$

The residual error is trivial to compute after sampling N spanning forests. Moreover, this is also the case for the degree of freedom term. A closer look at the trace shows:

$$\mathrm{tr}\left(\frac{1}{N}\sum_{k=1}^{N}\tilde{\mathsf{S}}^{(k)}\right) = \mathrm{tr}\left(\frac{1}{N}\sum_{k=1}^{N}\bar{\mathsf{S}}^{(k)}\right) = \frac{1}{N}\sum_{k=1}^{N}|\rho(\Phi_Q^{(k)})|.$$

This result yields that the trace term can be replaced with the average number of roots in the computation of $\texttt{SURE}(\mathbf{y},\tilde{\theta}(\mathbf{y}))$ or $\texttt{SURE}(\mathbf{y},\bar{\theta}(\mathbf{y}))$. Thus, the SURE scores of both estimators are trivial to numerically compute after sampling N spanning forests.

Note that neither $\texttt{SURE}(\mathbf{y},\tilde{\theta}(\mathbf{y}))$ nor $\texttt{SURE}(\mathbf{y},\bar{\theta}(\mathbf{y}))$ is an unbiased estimator for $\texttt{SURE}(\mathbf{y},\theta(\mathbf{y}))$. Moreover, the estimation errors read:

$$\begin{aligned}\mathbb{E}\left[\texttt{SURE}(\mathbf{y},\tilde{\theta}(\mathbf{y}))\right] - \texttt{SURE}(\mathbf{y},\theta(\mathbf{y})) &= \sum_{i\in\mathcal{V}}\mathrm{Var}(\tilde{\theta}(\mathbf{y})_i) \geq 0,\\ \mathbb{E}\left[\texttt{SURE}(\mathbf{y},\bar{\theta}(\mathbf{y}))\right] - \texttt{SURE}(\mathbf{y},\theta(\mathbf{y})) &= \sum_{i\in\mathcal{V}}\mathrm{Var}(\bar{\theta}(\mathbf{y})_i) \geq 0.\end{aligned} \tag{26}$$

Thus, $\texttt{SURE}(\mathbf{y},\tilde{\theta}(\mathbf{y}))$ and $\texttt{SURE}(\mathbf{y},\bar{\theta}(\mathbf{y}))$ are (with high probability) upper-bounds for $\texttt{SURE}(\mathbf{y},\theta(\mathbf{y}))$. For large graphs, in which computing $\texttt{SURE}(\mathbf{y},\theta(\mathbf{y}))$ is prohibitive, these upper bounds might also be useful since they can be obtained cheaply.

Leave-One-Out cross validation. LOOCV is a simple and popular method for hyperparameter selection. Unlike SURE, it does not require for the noise variance to be known. Keeping the same notation as above, LOOCV is defined as: (See Chapter 5.5 in [9]):

$$\texttt{LOOCV}(\mathbf{y}_\ell,\theta(\mathbf{y}_\ell)) = \frac{1}{|\ell|}\sum_{i\in\ell}(\theta^{-i}(\mathbf{y}_\ell)_i - y_i)^2$$

where $\theta^{-i}(\mathbf{y}_\ell)$ is the estimation without using the i-th measurement.

This method leaves y_i out at the estimation stage, and calculates the corresponding prediction error. The overall score is the average error over the vertices in ℓ. Note that this method needs to compute $\theta^{-i}(\mathbf{y}_\ell)$ for each individual i which in general costs n times the cost of the original estimator on the full data. Fortunately, this formula simplifies to the following for linear estimators in the form of $\theta(\mathbf{y}) = \mathsf{K}\mathbf{y}$ [9]:

$$\texttt{LOOCV}(\mathbf{y}_\ell,\theta(\mathbf{y})) = \frac{1}{|\ell|}\sum_{i\in\ell}\left(\frac{\theta(\mathbf{y}_\ell)_i - y_i}{1-\mathsf{K}_{i,i}}\right)^2 \tag{27}$$

which avoids re-computation.

LOOCV for RSF estimators. Similar to SURE, this score can be adapted for the RSF based estimators. For example, in case of $\tilde{\theta}(\mathbf{x}_\ell)$, it becomes:

$$\texttt{LOOCV}(\mathbf{y}_\ell,\tilde{\theta}(\mathbf{y}_\ell)) = \frac{1}{|\ell|}\sum_{i\in\ell}\left(\frac{\tilde{\theta}(\mathbf{y}_\ell)_i - y_i}{1-\frac{1}{N}\sum_{k=1}^{N}\tilde{\mathsf{S}}_{i,i}^{(k)}}\right)^2 \tag{28}$$

and for $\bar{\theta}$, it can be derived in the same way. Notice that every element in this expression is numerically available after sampling N spanning forests. Thus, this score can be easily computed for both estimators.

IV. RSF estimators for other graph problems

In this section, we explore a few graph problems in which the RSF based estimators presented can replace expensive exact computations.

A. Node Classification in semi-supervised learning

Consider a dataset consisting of elements one wishes to classify. In the semi-supervised learning context, the class label of a few elements are supposed to be known *a priori*, along with a graph structure encoding some measure of affinity between the different elements: the larger the weight of the edge connecting two elements, the closer they are according to some application-dependent metric, the more likely these two elements belong to the same class. The goal is then to infer all the labels given this prior information.

Among many options to solve this problem, label propagation [5], [36] and generalized SSL framework [12] are two well-known baseline approaches. In this section, we deploy $\tilde{\mathbf{x}}$ and $\bar{\mathbf{x}}$ to approximate the solutions given by these approaches.

Problem definition. Let us denote the labeled vertices by $\ell \subset \mathcal{V}$ (typically $|\ell| \ll |\mathcal{V}|$) and the unlabeled ones by $u = \mathcal{V}\backslash\ell$. Assume C distinct label classes and define the following encoding of the prior knowledge for the c-th class:

$$\forall i \in \mathcal{V}, \mathbf{y}_c(i) = \begin{cases}1 \text{ if } i \text{ is known to belong to the } c\text{-th class}\\ 0 \text{ otherwise.}\end{cases} \tag{29}$$

The matrix $\mathsf{Y} = [\mathbf{y}_1|\dots|\mathbf{y}_C] \in \mathbb{R}^{n\times C}$ thus encodes the prior knowledge. Many approaches to SSL formulate the problem as follows. First, for each class c, compute the so-called "classification function", defined as:

$$\mathbf{f}_c = \underset{\mathbf{z}_c\in\mathbb{R}^n}{\mathrm{argmin}}\,\mu\sum_{i\in\mathcal{V}}q_i'(y_c(i)-z_c(i))^2 + \mathbf{z}_c^\top\mathsf{L}\mathbf{z}_c \tag{30}$$

where μ and q_i''s are regularization parameters: μ sets the global regularization level, and each q_i' acts entry-wise (when q_i' is high, the corresponding entry in $\mathbf{f}_c$ is close to the measurement $\mathbf{y}_c(i)$). Eq. (30) has the following explicit solution:

$$\mathbf{f}_c = (\mathsf{L}+\mathsf{Q})^{-1}\mathsf{Q}\,\mathbf{y}_c \tag{31}$$

where $\mathsf{Q} = \mathrm{diag}(q_1,\dots,q_n)$ and $q_i = \mu q_i'$. Thus, each classification function $\mathbf{f}_c$ can be viewed as a smoothed version of the prior knowledge encoded in $\mathbf{y}_c$. As such, if $\mathbf{f}_c(i)$ is large, it implies that labels of class c are relatively dense around node i. The last step in these SSL algorithms is to assign each node i to the class $\mathrm{argmax}_c\,\mathbf{f}_c(i)$.

Label propagation and generalized SSL framework are two algorithms that adapt this solution in different ways. In particular, by using different set of q_i''s in Eq. (30), label propagation corresponds to solving graph interpolation and the generalized SSL framework may be understood as solving a graph TR. Thus, the RSF estimators can be

used to approximate the solution for both algorithms. In the following, we discuss the corresponding parameter settings for these algorithms along with their RSF versions.

Label Propagation. The label propagation algorithm [5], [36] solves the Dirichlet problem for each class c, that is:

$$\begin{aligned} \forall i \in \mathcal{V}, \quad & \mathsf{L}\mathbf{f}_c(i) = 0 \\ \text{s. t. } \forall i \in \ell, \quad & \mathbf{f}_c(i) = \mathbf{y}_c(i) \end{aligned} \tag{32}$$

which is equivalent to Eq. (14) for $\mu = 0$. Defining the classification matrix $\mathsf{F} = [\mathbf{f}_1 | \dots | \mathbf{f}_C] \in \mathbb{R}^{n \times C}$, one thus has:

$$\mathsf{F}_{i,c} = \begin{cases} \mathsf{Y}_{i,c}, & \text{if } i \in \ell \\ (-(\mathsf{L}_{u|u})^{-1} \mathsf{L}_{u|\ell} \mathsf{Y}_{\ell|:})_{i,c}, & \text{otherwise} \end{cases} \tag{33}$$

where $\mathsf{Y}_{\ell|:}$ is the matrix Y restricted to rows in ℓ. Note in passing that F corresponds to a special set of functions for graphs called *harmonic functions*. Besides being the solution of Dirichlet boundary problem, they have interesting connections with electrical networks and random walks [36].

Zhu [36] provide a simple algorithm to compute F without computing the inverse matrix. Starting from an arbitrary initial $\mathsf{F}^{(0)}$, at each iteration k, the algorithm updates $\mathsf{F}^{(k)} \leftarrow \mathsf{D}^{-1}\mathsf{W}\mathsf{F}^{(k-1)}$. The iteration is completed by setting the known labels $\mathsf{F}^{(k)}_{\ell|:}$ to $\mathsf{Y}_{\ell|:}$. They prove that the output of this iteration converges to F as $k \to \infty$ (see Section 2.3 in [36]).

Here, we provide an RSF-based estimator to approximate F. Two scenarii are possible. The first (unlikely) scenario is when any node in u is connected to at least one node in ℓ. In this case, one can rewrite Eq. (33) as:

$$\mathsf{F} = -\mathsf{K}\, \mathsf{Q}^{-1} \mathsf{L}_{u|\ell} \mathsf{Y} \quad \text{with } \mathsf{K} = (\mathsf{L}_{\mathcal{G}\setminus\ell} + \mathsf{Q})^{-1}\mathsf{Q} \tag{34}$$

where $\mathsf{L}_{\mathcal{G}\setminus\ell}$ is the Laplacian of the reduced graph obtained by removing the vertices (and the incident edges) in ℓ, $\mathsf{Q} \in \mathbb{R}^{|u|\times|u|}$ is a diagonal matrix with $\mathsf{Q}_{i,i} = \sum_{j\in\ell} w(i,j)$. The condition of this first scenario ensures that Q is indeed invertible; and RSFs on the reduced graph $\mathcal{G} \setminus \ell$ can thus estimate the columns of F.

However, when there exists at least one node in u that is not connected to ℓ, Q is no longer invertible and another approach is needed. In this second scenario, the parameters are defined over all vertices and set to $q_i = \alpha > 0$ for $i \in \ell$ and $q_i = 0$ for $i \in u$. The following proposition guarantees that as $\alpha \to \infty$, the RSF estimator $\tilde{\mathbf{x}}$ with this setting approximates the solution given in Eq (33).

Proposition 3. *Given the parameters $q_i = \alpha > 0$ for $i \in \ell$ and $q_i = 0$ for $i \in u$, as well as the input vector $\mathbf{y}_c \in \mathbb{R}^n$ for the RSF estimator $\tilde{\mathbf{x}}$, the following is verified:*

$$\lim_{\alpha\to\infty} \mathbb{E}\left[\tilde{\mathbf{x}}\right] = \mathbf{f}_c.$$

Proof. See the supplementary material for the detailed proof. □

From the random forest point-of-view, setting q_i to infinity for all nodes i in ℓ, and to 0 otherwise, implies that all possible realizations of Φ_Q have exactly the same root set: ℓ. Thus, when using the estimator $\tilde{\mathbf{x}}$, the measurements in ℓ are not altered and are simply propagated to other vertices via the sampled random trees. In addition, the estimator $\bar{\mathbf{x}}$ boils down to $\tilde{\mathbf{x}}$ in this very specific case.

Generalized SSL framework. The generalized SSL framework proposed in [12] defines the classification function as follows:

$$\mathbf{f}_c = \frac{\mu}{\mu+2}\left(\mathsf{I} - \frac{2}{\mu+2}\mathsf{D}^{-\eta}\mathsf{W}\mathsf{D}^{\eta-1}\right)^{-1}\mathbf{y}_c$$

where $\mu > 0$ is the regularization parameter and η controls the normalization of the graph Laplacian. This formula can also be written as:

$$\mathbf{f}_c = \mathsf{D}^{1-\eta}\mathsf{K}\mathsf{D}^{\eta-1}\mathbf{y}_c \text{ with } \mathsf{K} = (\mathsf{L}+\mathsf{Q})^{-1}\mathsf{Q}$$

where $\mathsf{Q} = \frac{\mu}{2}\mathsf{D}$. Notice that the cumbersome part in this formula is to compute $\mathsf{K}\mathsf{D}^{\eta-1}\mathbf{y}_c$ and it can be approximated by the proposed estimators on the input vector $\mathbf{y} = \mathsf{D}^{\eta-1}\mathbf{y}_c$. Then, $\mathbf{f}_c$ is obtained by left-multiplying the result by $\mathsf{D}^{1-\eta}$.

Both solutions, label propagation and the generalized SSL, can be considered as two different versions of a more generic optimization problem. Label propagation puts a very high confidence on the prior. $\tilde{\mathbf{x}}$, for example, only propagates the measurements of the labeled vertices (from the second scenario's perspective). Whereas, in generalized SSL, lower confidence over the prior information is assumed, and thus, the propagation of other measurements, which are all set to 0 in this encoding, is authorized. The success of both methods depends on the correctness of these assumptions on the data. Section V provides empirical comparisons on benchmark datasets.

B. Non-quadratic convex functions and Newton's method

Consider the following generalized optimization problem:

$$\hat{\mathbf{x}} = \underset{z\in\mathbb{R}^n}{\operatorname{argmin}}\, \mu f(\mathbf{z}) + \frac{1}{2}\mathbf{z}^\top \mathsf{L}\mathbf{z} \tag{35}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a generic twice-differentiable function (*e.g.* previously $f(\mathbf{z}) = ||\mathbf{y}-\mathbf{z}||_2^2$). A common scenario is when f is a log-likelihood, i.e. $f(\mathbf{z}) = \sum_{i=1}^n \log p(y_i|z_i)$. This is used when the assumption that the observations are Gaussian (given the signal) is inappropriate, for instance when the observations are discrete. In such cases f is not a quadratic function and there is typically no closed-form solution for 35.

In these cases, iterative approaches are often deployed. One popular approach among them is Newton's method. Let $L(\mathbf{z})$ denote the loss function, Newton's method follows the following iteration scheme:

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \alpha(\mathsf{H}L(\mathbf{z}_k))^{-1}\nabla L(\mathbf{z}_k) \text{ with } \alpha \in [0,1], \tag{36}$$

H and ∇ are the Hessian and gradient operators, respectively and $\mathbf{z}_k$ denotes the estimation at iteration k. Note that, by definition of the Hessian operator, this method requires twice-differentiability for the loss function.

Given this scheme, the methods proposed here may become useful for approximating the inversion at each iteration. We illustrate this usage in the following setup.

Assume an independent Poisson distribution for each likelihood at i:

$$P(y_i|\lambda = z_i) = \frac{z_i^{y_i}\exp(-z_i)}{y_i!}$$

where λ is the distribution parameter. This assumption is often made in image processing applications to eliminate shot noise [37]. Also, consider the slightly modified loss function:

$$L'(\mathbf{t}) = -\mu\sum_{i=1}^{n}\log P(y_i|\lambda = \exp(t_i)) + \frac{1}{2}\mathbf{t}^\top \mathsf{L}\mathbf{t} \qquad (37)$$

where $\exp(t_i) = z_i$. The gradient $\nabla L'(\mathbf{t})$ reads:

$$\nabla L'(\mathbf{t}) = \mu\exp(\mathbf{t}) - \mu\mathbf{y} + \mathsf{L}\mathbf{t}$$

where $\exp$ operates entry-wise on vectors. Then, the Hessian matrix becomes:

$$\mathsf{H}L'(\mathbf{t}) = \mu\,\mathrm{diag}(\exp(\mathbf{t})) + \mathsf{L}.$$

With these two ingredients, the iterative scheme becomes:

$$\mathbf{t}_{k+1} = \mathbf{t}_k - \alpha\left[\mu\,\mathrm{diag}(\exp(\mathbf{t}_k)) + \mathsf{L}\right]^{-1}\left(\mu\exp(\mathbf{t}_k) - \mu\mathbf{y} + \mathsf{L}\mathbf{t}_k\right)$$

The update term, which requires an inverse operation, can be approximated by our RSF estimators. This approximation is achieved by setting $\mathsf{Q} = \mu\,\mathrm{diag}(\exp(\mathbf{t}_k))$ and the graph measurements $\mathbf{y}'$ to $\mu^{-1}\,\mathrm{diag}(\exp(-\mathbf{t}_k))(\mu\exp(\mathbf{t}_k) - \mu\mathbf{y} + \mathsf{L}\mathbf{t}_k)$. This particular case yields the following computation:

$$\begin{aligned}\mathsf{K}\mathbf{y}' &= (\mathsf{Q}+\mathsf{L})^{-1}\mathsf{Q}\mathbf{y}' \\ &= \left[\mu\,\mathrm{diag}(\exp(\mathbf{t}_k)) + \mathsf{L}\right]^{-1}(\mu\exp(\mathbf{t}_k) - \mu\mathbf{y} + \mathsf{L}\mathbf{t}_k)\end{aligned}$$

which equals to the update in Newton's method. Thus, the RSF estimators can be easily used to compute each update step with a cheap cost.

In the classical Newton's method *i.e.* $\alpha = 1$, convergence of the result is not guaranteed. It might diverge or get stuck in a loop depending on the closeness of the initial point $\mathbf{t}_0$ to the solution. Guessing a good initial point is not an easy task and may require expensive computations in high dimensions. Instead, modifying α is a more applicable option to ensure convergence. Thus, Newton's method is often combined with an additional step at each iteration in which α is reset accordingly. Line search algorithms [13] are simple and well-understood methods for this purpose. At each iteration, if needed, they damp the applied update by shrinking α. These methods provide convergence, however, they may require more iteration steps w.r.t. the pure Newton's method with a good initial point. In our case, the updates are stochastic and exact convergence cannot be expected.

C. l_1-Regularization and iteratively reweighted least squares

As with the data fidelity term, many alternatives for the regularization term are also available. Among them, l_1-regularization [38] is often deployed to obtain sparser solutions. In [39], Sharpnack *et al.* adapts l_1-regularization for graphs as follows:

$$\hat{\mathbf{x}} = \underset{\mathbf{z}\in\mathbb{R}^n}{\mathrm{argmin}}\,\mu||\mathbf{y}-\mathbf{z}||_2^2 + ||\mathsf{B}\mathbf{z}||_1 \qquad (38)$$

where B is the edge incidence matrix and $||\mathsf{B}\mathbf{z}||_1 = \sum_{(i,j)\in\mathcal{E}} w(i,j)^{1/2}|z_i - z_j|$. In contrast to the l_2 regularization, a closed form solution for the l_1 case is not available and iterative optimization schemes are again used to find the solution. The iterative reweighted least square [14] (IRLS) method is one such option and we show here how our RSF based estimators can help compute its numerical bottleneck.

Let $\mathsf{M} = \mathrm{diag}(\mathrm{abs}(\mathsf{B}\mathbf{z}))^{-1}$ where abs is the entry-wise absolute value operator. Note that

$$||\mathsf{B}\mathbf{z}||_1 = \mathrm{abs}(\mathbf{z}^\top\mathsf{B}^\top)\mathbf{1} = \mathbf{z}^\top\mathsf{B}^\top\mathsf{M}\mathsf{B}\mathbf{z}$$

where $\mathbf{1}\in\mathbb{R}^m$ is the all-ones vector. The problem of Eq. (38) can thus be re-written as:

$$\hat{\mathbf{x}} = \underset{\mathbf{z}\in\mathbb{R}^n}{\mathrm{argmin}}\,\mu||\mathbf{y}-\mathbf{z}||_2^2 + \mathbf{z}^\top\mathsf{B}^\top\mathsf{M}\mathsf{B}\mathbf{z}, \qquad (39)$$

and the IRLS iterative loop reads:

$$\mathbf{z}_{k+1} = (2\mu\mathsf{I} + \mathsf{B}^\top\mathsf{M}_k\mathsf{B})^{-1}\mu\,\mathbf{y} \text{ with } \mathsf{M}_k = \mathrm{diag}(\mathrm{abs}(\mathsf{B}\mathbf{z}_k))^{-1}.$$

A more detailed derivation and the convergence analysis of this scheme can be found in [14]. Notice that, by definition, $\mathsf{B}^\top\mathsf{M}_k\mathsf{B}$ equals to a reweighted graph Laplacian L_k. Then, computing the update at each iteration step immediately reduces to solving a graph Tikhonov regularization. Thus, $\tilde{\mathbf{x}}$ or $\bar{\mathbf{x}}$ can be used for estimating the update. See the supplementary materials for an illustration.

V. Experiments

We provide in this section several illustrations and run time analysis of the proposed methods. First of all, in Section V-A, the RSF based methods are run on two image denoising setups. In these, we consider

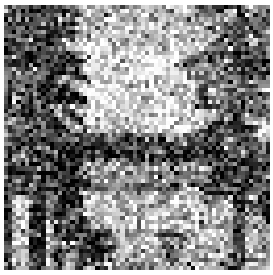
- An image corrupted by an additive Gaussian noise, for which we show its RSF-based Tikhonov regularization parameterized by SURE.
- An image corrupted by a Poisson noise, for which we show its version denoised by the RSF-based Newton's method coupled with line search.

In both setups, the underlying graph is assumed to be a 2-dimensional (2D) grid graph. We stress that these illustrations' main purpose is not to compete with the state-of-the-art image denoising methods but to provide visual examples for the proposed methods. Denoising performance can be improved in many ways, but it is not our goal here.

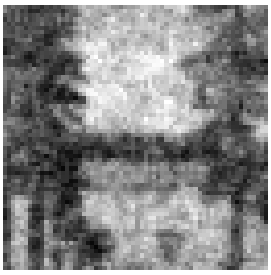
Secondly, in Section V-B, the SSL node classification problem is considered on three benchmark datasets. We examine the classification performances of Tikhonov regularization, label propagation and the RSF versions of these algorithms. In these experiments, the parameter selection for the Tikhonov regularization is done by leave-one-out cross validation separately for each method.

Finally, in Section V-C, we compare RSF estimators with the state-of-the-art methods, *i.e.* Chebyshev polynomials and conjugate gradient method with and without preconditioning, in various graphs. Given a denoising setup, error vs. runtime plots are shown for all these algorithms.

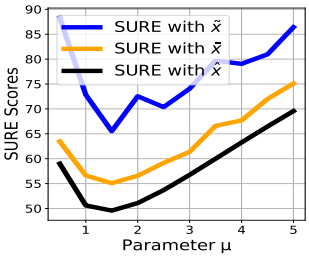
(a) Original Image $\mathbf{x}$



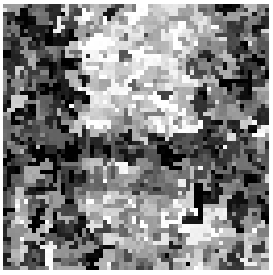
(b) Noisy Measurments $\mathbf{y}$, PSNR = 14.0



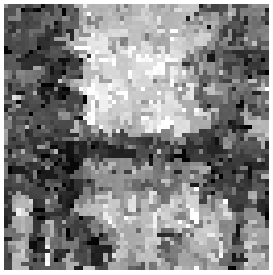
(c) Exact solution $\hat{\mathbf{x}}$, PSNR = 19.2



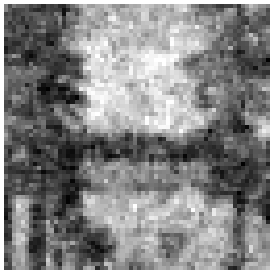
(d) SURE scores vs μ



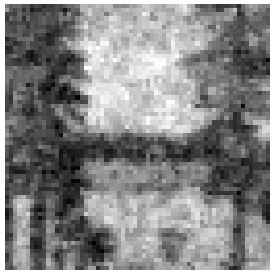
(e) $\tilde{\mathbf{x}}$ with $N = 1$, PSNR=12.2



(f) $\bar{\mathbf{x}}$ with $N = 1$, PSNR=16.1



(g) $\tilde{\mathbf{x}}$ with $N = 20$, PSNR = 18.3



(h) $\bar{\mathbf{x}}$ with $N = 20$, PSNR = 18.9

Fig. 6: An image denoising experiment with additive Gaussian noise. a) the original image. b) a noisy version $\mathbf{y} = \mathbf{x} + \boldsymbol{\epsilon}$ *with* $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma)$ *with* $\sigma = 0.2$*. c) the exact solution* $\hat{\mathbf{x}} = \mathsf{K}\mathbf{y}$*. Figure d) summarizes the SURE scores of* $\hat{\mathbf{x}}$*,* $\tilde{\mathbf{x}}$ *and* $\bar{\mathbf{x}}$ *for different* μ*'s where* $\mu \in \{0.5, 1.0, 1.5, \ldots 5.0\}$*. For each estimator, the value of* μ *that minimizes the SURE score is selected. For the computation of SURE scores, the noise variance is assumed to be known. e-f) the two RSF estimates* $\tilde{\mathbf{x}}$ *and* $\bar{\mathbf{x}}$ *based on only one sampled forest. g-h) same as e-f) but averaged over* $N = 20$ *sampled forests. Given the initial noisy image with PSNR*$= 14.0$*, the exact solution produces a denoised image with PSNR*$= 19.2$ *where the RSF estimates* $\tilde{\mathbf{x}}$ *and* $\bar{\mathbf{x}}$ *closely follows with PSNR*$=18.3$ *and* 18.9*, respectively.*

A. Image denoising

A 2D grid graph is a natural underlying structure for images: every pixel corresponds to a node and each pixel is connected to its four direct neighbors with equal weights. Other structures could be used (such as an 8-neighbour version of the grid graph) and performances will depend on the chosen structure. However, for the purpose of illustration, we will only consider the simplest 4-neighbour grid graph.

In the first setup, noisy (with additive Gaussian noise) measurements $\mathbf{y}$ of the original image $\mathbf{x}$ are given:

$$\mathbf{y} = \mathbf{x} + \boldsymbol{\epsilon} \text{ with } \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathsf{I}).$$

To recover $\mathbf{x}$, Tikhonov regularization is applied. Fig. 6 compares the exact result $\hat{\mathbf{x}} = \mathsf{K}\mathbf{y} = (\mathsf{L} + \mu\mathsf{I})^{-1}\mu\mathbf{y}$, to its forest-based approximations $\tilde{\mathbf{x}}$ and $\bar{\mathbf{x}}$. The SURE method to estimate the best value of μ is also illustrated for all three estimations of the original image $\mathbf{x}$ (and we remark in passing that they are consistent). The results confirm that $\bar{\mathbf{x}}$ produces a better estimate for $\hat{\mathbf{x}}$ than $\tilde{\mathbf{x}}$. Also, in Fig. 6d, the scores computed for the RSF estimators are observed as upper bounds of the scores for $\hat{\mathbf{x}}$, as expected from the results in Eq (26). As observed in this illustration, and as expected by our theoretical analysis, $\bar{\mathbf{x}}$ always performs better than $\tilde{\mathbf{x}}$. In the following experiments and in order to avoid overloading the figures, we will omit the results obtained with $\tilde{\mathbf{x}}$.

In the second setup, each pixel value is assumed to be sampled from a Poisson distribution whose mean is the true value of the pixel:

$$\mathbf{y} \sim \text{Poisson}(\mathbf{x}).$$

To reconstruct $\mathbf{x}$, Newton's method is applied, as explained in Section IV-B. The line search algorithm is used for picking the value of α at each iteration to ensure convergence. Both qualitative and quantitative results in Fig. 7 show that both $\hat{\mathbf{x}}$ and $\bar{\mathbf{x}}$ converge to the same solution. Fig. 7c shows a similar decrease of the loss function for both $\hat{\mathbf{x}}$ and $\bar{\mathbf{x}}$, even though convergence is slightly faster for $\hat{\mathbf{x}}$.

B. Node classification

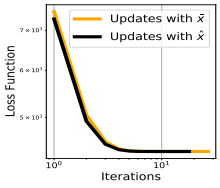
In this illustration, we run our methods to solve the node classification problem discussed in Section IV-A. For the generalized SSL framework, η is set to 0, and μ is set by RSF based cross-validation.

The experiments are done on three standard benchmark datasets, namely Cora , Citeseer and Pubmed [5]. In the first two, the underlying graphs are disconnected, thus we use the largest connected components. Also, in all datasets, the orientations of the edges are omitted to operate on undirected graphs. The general statistics of these datasets after the preprocessing are summarized in Table I. Note that in these three datasets, the class of each node is known. This will enable us to test the different SSL frameworks (a small arbitrary fraction of nodes will serve as pre-labeled nodes, and one tests whether or not this is sufficient to infer the class of all nodes). More precisely,

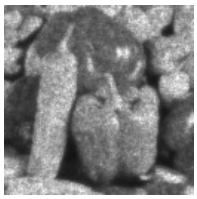
[5]These datasets can be found in https://linqs.soe.ucsc.edu/data .

(a) Original image $\mathbf{x}$

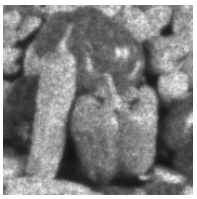
(b) Noisy image $\mathbf{y}$, PSNR=17.1



(c) Loss function through iterations



(d) Denoised image with $\hat{\mathbf{x}}$, PSNR=22.8



(e) Denoised image with $\bar{\mathbf{x}}$ with $N = 40$, PSNR=22.8

Fig. 7: An image denoising experiment with Poisson noise. a) original image $\mathbf{x}$. b) a noisy version $\mathbf{y}\sim$Poisson$(\mathbf{x})$. Newton's method is deployed to recover $\mathbf{x}$ by minimizing the loss function in 37. For two update options, namely $\hat{\mathbf{x}}$ and $\bar{\mathbf{x}}$, Newton's method yields the results shown on the bottom line. Figure c) shows the loss function through the iterations for the two update options.

TABLE I: SSL dataset statistics after preprocessing

Dataset	#Nodes	#Edges	#Classes
Citeseer	2110	3668	6
Cora	2485	5069	7
Pubmed	19717	44324	3

we use the following procedure:

- m vertices are selected at random per class as the labeled nodes,
- the parameter μ is set by LOOCV separately for $\hat{\mathbf{x}}$ and $\bar{\mathbf{x}}$,
- the classification functions $\mathbf{f}_c$ for each class c are computed by the generalized SSL framework, label propagation and their RSF versions averaged over N repetitions,
- for each vertex i, we assign $\text{argmax}_c \, \mathsf{F}_{i,c}$ as its class and calculate the classification accuracy as the ratio of correctly predicted labels to the total number of predictions.

In Fig. 8, the classification accuracy is reported as m and N vary. The results are averaged over 50 realizations of the m labeled vertices for all datasets.

For the first two datasets, Cora and Citeseer, $\bar{\mathbf{x}}$ has a comparable performance with the exact solution. However, in Pubmed, $\bar{\mathbf{x}}$ fails to perform as good as the $\hat{\mathbf{x}}$ for gSSL due to larger approximation errors in both the parameter selection and the estimation steps.

The empirical results yield that the proposed methods need much less forest realizations to reach the exact solution of LP rather than the generalized SSL. However, sampling a forest for LP often takes more time if n is large and m is relatively small. For example, in the Pubmed graph, for $m = 20$, sampling a single forest for LP (resp. the generalized SSL) takes 6.3×10^{-2} (resp. 1.4×10^{-3}) seconds averaged over 100 repetitions in a single threaded run time of a laptop.

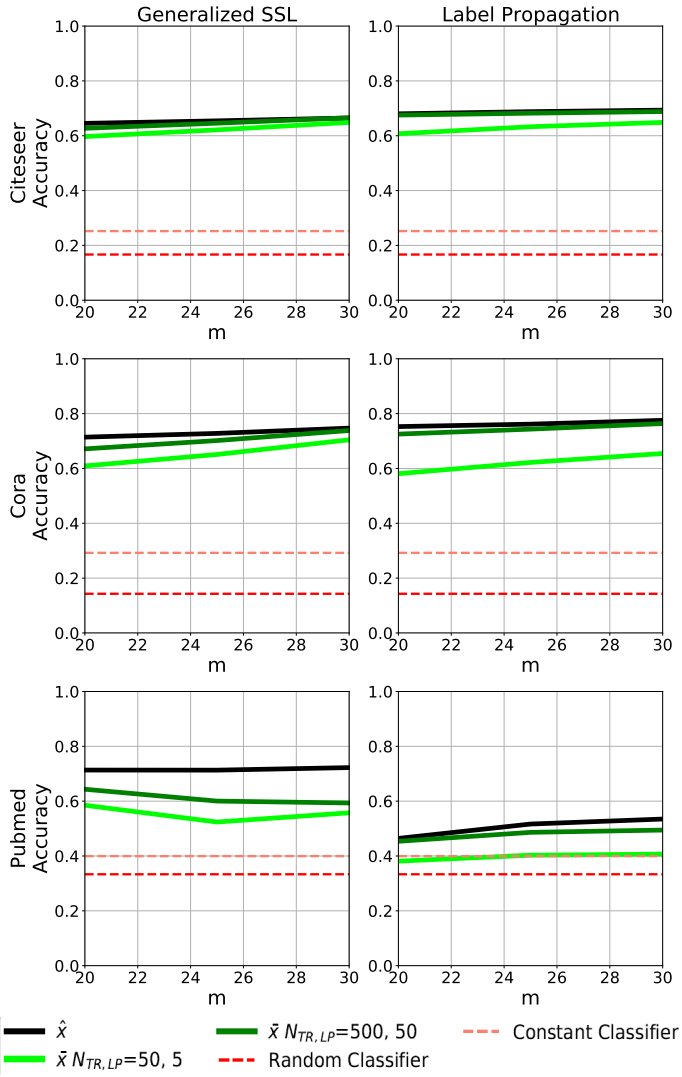


Fig. 8: The classification accuracy of the generalized SSL, LP and their RSF variants are presented on the datasets Citeseer, Cora and Pubmed. The RSF methods for the generalized SSL are illustrated for $N = 50,\ 500$ forest realizations, whereas, these numbers for LP are $N = 5,\ 50$. In these plots, the random classifier denotes the accuracy of inferring classes at random and the constant classifier is the accuracy of assigning the most occurring class to all unlabeled vertices. The results for Citeseer, Cora and Pubmed datasets are averaged over respectively 50 different set of labeled vertices.

Note that these figures strongly depend on m and the given network. Thus, one needs to examine this trade-off with the given dataset to adjust the total run time.

C. Run-time analysis

In this section, we present an empirical comparison of the proposed estimators with classical approaches, namely Chebyshev polynomial approximation [16], and the conjugate gradient (CG) method (with and without preconditioning) [15]. We run these algorithms on various graphs and graph signals and report curves of performance versus run-time.

1) Experimental setup: In these experiments, we compute the result of the following graph smoothing operation via several methods:

$$\hat{\mathbf{x}} = \mathsf{K}\mathbf{y} \text{ with } \mathsf{K} = (\mathsf{Q}+\mathsf{L})^{-1}\mathsf{Q} \tag{40}$$

where Q is set to $q\mathsf{I}$ for the sake of simplicity in parameter tuning. Given a graph, the vector $\mathbf{y}$ is assumed to contain noisy measurements of a k-*bandlimited* signal $\mathbf{x}$ [40], [41]. The term "bandlimited" means that the signal belongs to a particular subspace of harmonic functions and is defined more precisely below. The parameter q is set to the value maximizing the denoising performance. In the following, these steps are further detailed.

Graph signal generation. Consider $\mathsf{L} = \mathsf{U}^\top\Lambda\mathsf{U}$, the eigendecomposition of L with the eigenvectors $\mathsf{U} = (\mathbf{u}_1|\mathbf{u}_2|\dots|\mathbf{u}_n)$ and eigenvalues $\lambda_1 = 0 \leq \lambda_2 \leq \dots \leq \lambda_n$ forming $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$. In GSP, $\mathsf{U} = (\mathbf{u}_1|\mathbf{u}_2|\dots|\mathbf{u}_n)$ is considered to be a Fourier basis for graph signals, and the λ_i's are interpreted as generalized graph frequencies. For more on graph Fourier bases, we refer the reader to [2], [42]. In our experiments, we consider the denoising of a noisy version $\mathbf{y}$ of a bandlimited signal $\mathbf{x}$ defined as:

- $\mathbf{x}$ is a k-bandlimited signal, *i.e.*:

$$\mathbf{x} = \sum_{i=1}^{k} \alpha_i \mathbf{u}_i$$

  where the α_i's are the graph Fourier coefficients of $\mathbf{x}$. In other words, $\mathbf{x}$ is a low frequency graph signal, and is thus smooth on the given graph.
- The noise is Gaussian: $\mathbf{y} = \mathbf{x} + \boldsymbol{\epsilon}$ with $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$.

In the simulations, we randomly generate different realizations of $\mathbf{y}$ as follows:

1) Sample $k = 5$ Fourier coefficients α_i's from a Gaussian distribution $\mathcal{N}(0, 1)$ and obtain a realization of $\mathbf{x}$.
2) Normalize it to unit norm *i.e.* $||\mathbf{x}||_2 = 1$.
3) Draw n iid values from $\mathcal{N}(0, \sigma^2)$ to create the noise vector $\boldsymbol{\epsilon}$. To obtain a given signal-to-noise ratio (the SNR is fixed to 2), the variance of the noise is set to $\sigma^2 = (n \times \mathrm{SNR})^{-1} = 1/2n$.
4) Pass the noisy signal $\mathbf{y} = \mathbf{x} + \boldsymbol{\epsilon}$ to the algorithms.

Performance metrics. Let $\mathbf{x}^*$ be the generic output of the tested algorithms. In the experiments, we measure their performance with respect to two metrics. The first one is the approximation error $||\hat{\mathbf{x}} - \mathbf{x}^*||_2$ which evaluates the quality of the approximation to $\hat{\mathbf{x}} = \mathsf{K}\mathbf{y}$ achieved by the algorithms. The second is the reconstruction error $||\mathbf{x} - \mathbf{x}^*||_2$ which measures the denoising performance of the output.

Parameter tuning. The parameter q is always set to the value yielding the best denoising performance (that is, to the value of q minimizing $||\mathbf{x} - \hat{\mathbf{x}}||_2$).

Graphs. We experimented on many graphs with varying structures and density. Table II summarizes all graphs used in the experiments.

Experimental procedure. For a given graph, we follow this simple procedure:

1) Generate $\mathbf{y}$ as described earlier.
2) Find the value of q that minimizes $||\mathbf{x} - \hat{\mathbf{x}}||$ by a grid search.
3) Run the algorithms and measure the run-time, approximation error and reconstruction error.

This procedure is repeated for different runs of the algorithms and realizations of $\mathbf{y}$, and the results are averaged. All simulations are done in a single thread of a laptop. Fig. 9 summarizes the results.

2) Results: The algorithms compared are: the RSF based estimators, Chebyshev polynomial approximation, conjugate gradient method with and without preconditioning[6]. The preconditioner for CG is the Algebraic Multigrid (AMG)[7] method [46]. All of these algorithms admit an iteration parameter determining their performance: the larger this iteration parameter, the better the approximation but the longer the computation time. These parameters are:

1) The number of forests for $\bar{\mathbf{x}}$
2) The polynomial's degree in Chebyshev approximation
3) The number of iterations for CG and preconditioned CG (PCG).

For each algorithm and each graph, this iteration parameter is swept through 17 logarithmically spaced values between 1 and 100: $\{1, 2, \dots, 62, 78, 100\}$, corresponding to the 17 plotted markers forming each curve. In each curve, the marker at the top left-hand corner thus corresponds to 1 iteration, whereas the marker at the bottom right-hand corner corresponds to 100 iterations. The samples for error and time measurements are collected from separate runs. The error results are averaged over 20 realizations of $\mathbf{y}$. The time measurements are averaged over 100 runs for each realization of $\mathbf{y}$. For the graphs generated from random graph models, such as Erdos-Renyi or Barabasi-Albert, only a single realization of the graph is used in these results.

PCG and Chebyshev polynomial approximation are two algorithms that require some preprocessing. In PCG, this preprocessing is simply the calculation of the preconditioner. In the polynomial approximation method, one needs to provide the spectrum interval $[0, b]$ over which the approximation should be computed (the interval starts at 0 since the smallest eigenvalue of L is necessarily null). For the polynomial approximation to work, b should be an upper bound of λ_n, L's largest eigenvalue. A popular option[8] in the GSP literature,

[6]Julia implementation in https://julialinearalgebra.github.io/IterativeSolvers.jl/dev/

[7]Julia implementation in https://github.com/JuliaLinearAlgebra/AlgebraicMultigrid.jl. The package's default parameters are used.

[8]Another option is to set b to the trivial upper bound equal to $2d_{\max}$ where $d_{\max}$ is the largest degree (this upper bound is easily obtained via Gershgorin's circle theorem). The preprocessing cost is in this case null but this upper bound has the drawback of being very crude for some graphs: a good polynomial approximation constrained on such a large interval will require an unnecessarily large polynomial order –inducing a longer computation time.

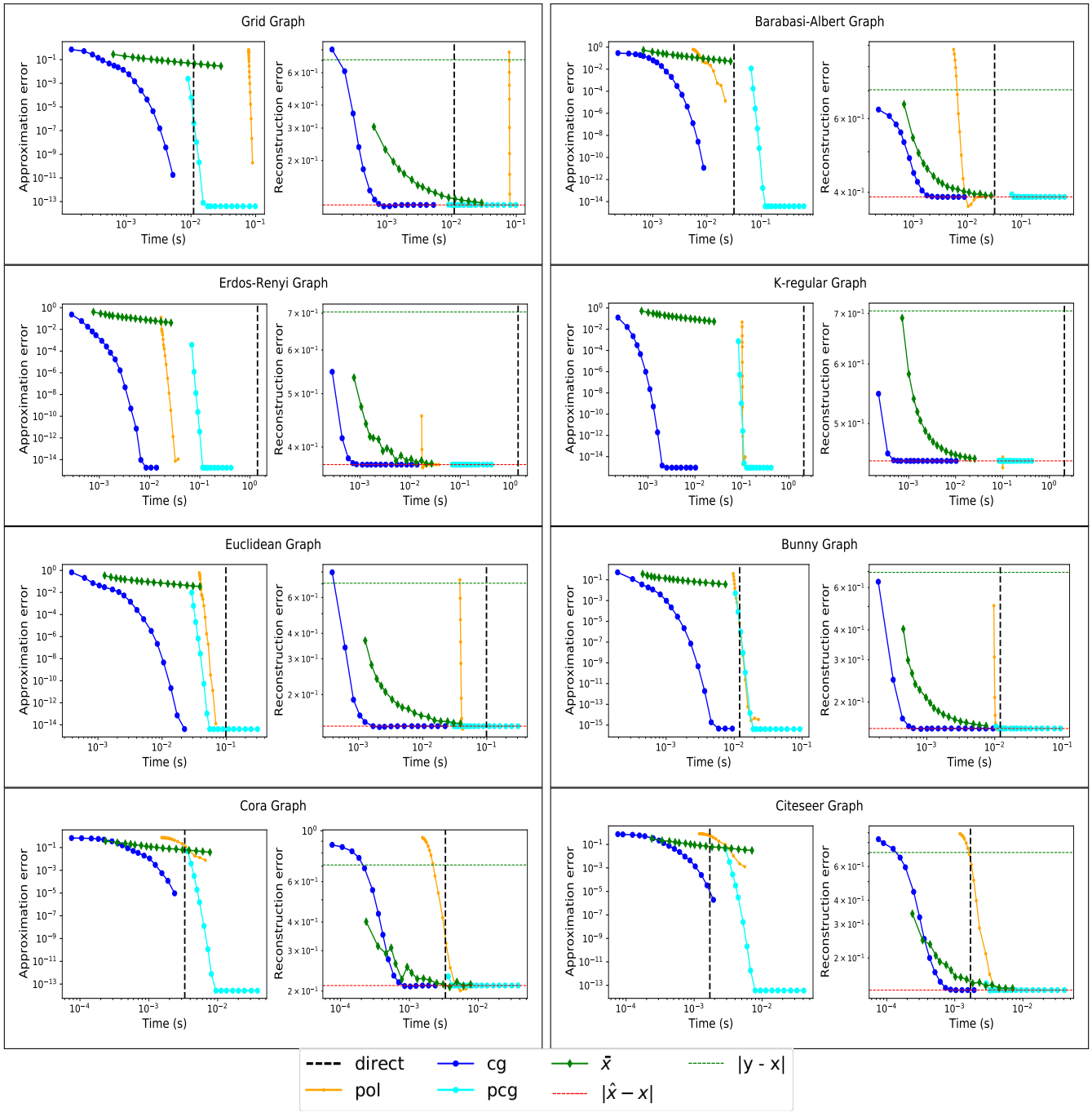


Fig. 9: Approximation and Reconstruction error vs run-time for CG (dark blue), PCG (cyan), Chebyshev polynomial approximations (orange) and the RSF estimator $\bar{\mathbf{x}}$ (dark green) over various graphs. The vertical dashed line corresponds to the time to calculate the exact solution via the backslash operation of Julia (Sparse decomposition). The horizontal dashed red line indicate the reconstruction error of the exact solution $||\mathbf{x} - \hat{\mathbf{x}}||_2$. Finally, the horizontal green dashed line is the reconstruction error of the noisy signal $\mathbf{y}$. In all graphs and methods, we observe that the best denoising performance is reached many iterations before reaching the best approximation error.

TABLE II: Graphs considered in the run-time experiments

Graph	n	m	description
Grid	10000	20000	100x100 grid with periodic boundary conditions.
Barabasi-Albert	10000	19996	Randomly generated by the Barabasi-Albert model [43].
Erdos-Renyi	10000	49877	A random Erdos-Renyi graph with average degree around 10.
K-regular	10000	50000	A random graph where each node has exactly 10 neighbors.
Euclidean	10000	112613	(K=20)-nearest neighbor graph of n randomly drawn points in $\mathbb{R}^3$. Whenever a node i is a K-nearest neighbor of a node j, both edges (i,j) and (j,i) are added to symmetrize the graph.
Bunny	2503	65490	($\epsilon = 0.2$)-Nearest neighbor graph of Stanford's bunny point cloud dataset [44]. Taken from the PyGSP toolbox [45]. Whenever a point is close to another one in a distance less than ϵ, an edge between them is added.
Citeseer	2110	3668	-
Cora	2485	5069	-

and the option we choose in this paper, is straightforward: compute λ_n and set b to λ_n. This ensures that the polynomial order required for a good approximation on $[0, b]$ is the lowest possible, thus minimizing the number of matrix-vector computations. However, the preprocessing step of computing λ_n, even via efficient Krylov-based routines, is not free and is in fact a significant part of the cost on the graph examples considered here. In Fig. 9, we report the total time (preprocessing and algorithm run) for these algorithms. Both algorithms have very steep convergence but their preprocessing costs make them stand out compared to the other methods: a more favorable context for them would be cases where one needs to filter *many* graph signals, thus making their overhead cost worthwhile.

Regarding the performance of $\bar{\mathbf{x}}$, we observe that it follows the usual Monte Carlo convergence regime (*i.e.*, $\mathcal{O}(N^{-\frac{1}{2}})$, thus linear on log-scale) as expected. Due to this inherent weakness, RSF methods cannot compete with the state-of-the-art if the goal is to obtain high precision in the approximation of $\hat{\mathbf{x}}$. However, as the results also show, it is often not necessary to have very high precision in approximation to get a good reconstruction performance (no point computing with high accuracy an inaccurate quantity). See [47] for a detailed discussion of this point in the context of Stochastic Gradient Descent. When looking at reconstruction errors instead of approximation errors, RSF-based estimators are comparable to state-of-the-art methods (results depend on the graph at hand).

Note that the current implementation of the RSF methods fail to benefit from compiler and hardware optimizations as much as the other deterministic methods do. The deterministic methods use the matrix L only in the vector-matrix products. In sparse implementations, these operations are well-optimized in terms of memory-access to the entries of L. On the other hand, the current implementation of RSF methods does not have this advantage. To sample a forest, the entries of L are accessed randomly, which incurs additional cost as random parts of the matrix need to be pulled into cache. Yet, there are several avenues open for speeding up our method so that it reaches state-of-the-art performance.

Importantly, recent research in computer science has shown that Wilson's algorithm is not asymptotically optimal for UST sampling. There are recent methods that are nearly linear in the number of edges, at the cost of a small approximation error, for instance, [48], [49]. However, at this stage, the asymptotically-optimal algorithms are very complex and, in some cases, close to unimplementable. Wilson's algorithm, on the other hand, is very easy to implement.

In terms of practical, short term ideas for speeding up our methods, there are quite a few options. One is to exploit a multilevel representation. A second is to use the coupled random forests algorithm of Avena & Gaudillière [23], which generates forests for different values of q in one go and so can produce a regularisation path for the Tikhonov estimator. A third is to use random forests as preconditioners to CG. We plan to explore these possibilities in future work.

VI. Conclusion

The Monte Carlo estimators can be used as building blocks in various problems, and are amenable to theoretical analysis. As we have shown, the proposed methods are adaptable to various graph-based optimization algorithms including the generalized SSL framework, label propagation, Newton's method and IRLS. Moreover, via a trick known at least since Gremban [50, ch. 7] , these RSF techniques can be easily extended to a larger class of matrices than Laplacians, namely symmetric diagonally dominant matrices, as explained for instance in [51]. In future work, we will continue to explore the links between RSFs and graph-related algebra to develop efficient estimators of graph characteristics such as effective resistances or $\mathrm{tr}(\mathsf{L}^\dagger)$ where $\mathsf{L}^\dagger$ is the Moore-Penrose inverse of L.

Acknowledgments

We thank the anonymous reviewers for their comments and suggestions to improve this manuscript.

References

[1] W. Huang, T. A. Bolton, J. D. Medaglia, D. S. Bassett, A. Ribeiro, and D. Van De Ville, "A Graph Signal Processing Perspective on Functional Brain Imaging," *Proc. IEEE*, vol. 106, no. 5, pp. 868–885, may 2018.

[2] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE signal processing magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[3] A. Sandryhaila and J. M. Moura, "Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 80–90, 2014.

[4] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.

[5] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," 2002.

[6] A. J. Smola and R. Kondor, "Kernels and regularization on graphs," in *Learning theory and kernel machines*. Springer, 2003, pp. 144–158.

[7] M. Belkin, I. Matveeva, and P. Niyogi, "Regularization and semi-supervised learning on large graphs," in *International Conference on Computational Learning Theory*. Springer, 2004, pp. 624–638.

[8] I. Pesenson, "Variational Splines and Paley-Wiener Spaces on Combinatorial Graphs," *Constr. Approx.*, vol. 29, no. 1, pp. 1–21, nov 2009.

[9] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, "The elements of statistical learning: data mining, inference and prediction," *Math. Intell.*, vol. 27, no. 2, pp. 83–85, 2005.

[10] L. Grady and E. Schwartz, "Anisotropic interpolation on graphs : The combinatorial dirichlet problem Anisotropic Interpolation on Graphs : The Combinatorial Dirichlet Problem," *Bost. Univ.*, 2003.

[11] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Adv. Neural Inf. Process. Syst.*, 2004, pp. 321–328.

[12] K. Avrachenkov, A. Mishenin, P. Gonçalves, and M. Sokol, "Generalized optimization framework for graph-based semi-supervised learning," in *Proc. 2012 SIAM Int. Conf. Data Min.* SIAM, 2012, pp. 966–974.

[13] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[14] C. S. Burrus, J. A. Barreto, and I. W. Selesnick, "Iterative Reweighted Least-Squares Design of FIR Filters," *IEEE Trans. Signal Process.*, vol. 42, no. 11, pp. 2926–2936, 1994.

[15] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[16] D. I. Shuman, P. Vandergheynst, and P. Frossard, "Chebyshev polynomial approximation for distributed signal processing," in *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*. IEEE, 2011, pp. 1–8.

[17] F. R. K. Chung and F. C. Graham, *Spectral graph theory*. American Mathematical Soc., 1997, no. 92.

[18] G. Grimmett, *Probability on graphs: Random processes on graphs and lattices: Second edition*. Cambridge University Press, 2018, vol. 8.

[19] X. M. Wu, Z. Li, A. M. C. So, J. Wright, and S. F. Chang, "Learning with partially absorbing random walks," in *Adv. Neural Inf. Process. Syst.*, vol. 4, 2012, pp. 3077–3085.

[20] X. Wu, "Learning on Graphs with Partially Absorbing Random Walks: Theory and Practice," Ph.D. dissertation, Columbia University, 2016.

[21] K. Avrachenkov, P. Chebotarev, and A. Mishenin, "Semi-supervised learning with regularized Laplacian," *Optim. Methods Softw.*, vol. 32, no. 2, pp. 222–236, 2017.

[22] L. Avena and A. Gaudillière, "Random spanning forests, markov matrix spectra and well distributed points," *arXiv preprint arXiv:1310.1723*, 2013.

[23] L. Avena and A. Gaudillière, "Two Applications of Random Spanning Forests," *Journal of Theoretical Probability*, Jul. 2017. [Online]. Available: http://link.springer.com/10.1007/s10959-017-0771-3

[24] D. B. Wilson, "Generating random spanning trees more quickly than the cover time," in *Proc. Annu. ACM Symp. Theory Comput.*, vol. Part F1294. Association for Computing Machinery, jul 1996, pp. 296–303.

[25] Y. Y. Pilavci, P. O. Amblard, S. Barthelmé, and N. Tremblay, "Smoothing graph signals via random spanning forests," in *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, vol. 2020-May. IEEE Inc., may 2020, pp. 5630–5634.

[26] P. Marchal, "Loop-erased random walks, spanning trees and hamiltonian cycles," *Elect. Comm. in Probab*, vol. 5, pp. 39–50, 2000.

[27] A. Kulesza and B. Taskar, "Determinantal point processes for machine learning," *Found. Trends Mach. Learn.*, vol. 5, no. 2-3, pp. 123–286, jul 2012.

[28] D. Blackwell, "Conditional Expectation and Unbiased Sequential Estimation," *Ann. Math. Stat.*, vol. 18, no. 1, pp. 105–110, mar 1947.

[29] C. R. Rao, "Information and the accuracy attainable in the estimation of statistical parameters," in *Breakthroughs in statistics*. Springer, 1992, pp. 235–247.

[30] N. Keriven, A. Bietti, and S. Vaiter, "Convergence and stability of graph convolutional networks on large random graphs," *arXiv preprint arXiv:2006.01868*, 2020.

[31] D. Girard, "Un algorithme simple et rapide pour la validation croisée généralisée sur des problèmes de grande taille," Tech. Rep., 1987.

[32] M. F. Hutchinson, "A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines," *Commun. Stat. - Simul. Comput.*, vol. 19, no. 2, pp. 433–450, 1990.

[33] S. Barthelme, N. Tremblay, A. Gaudilliere, L. Avena, and P.-O. Amblard, "Estimating the inverse trace using random forests on graphs," in *XXVIIème Colloq. GRETSI (GRETSI 2019)*, Lille, France, aug 2019.

[34] R. Tibshirani and L. Wasserman, "Stein's unbiased risk estimate," *Course notes from "Statistical Mach. Learn.*, pp. 1–12, 2015.

[35] C. M. Stein, "Estimation of the Mean of a Multivariate Normal Distribution," *Ann. Stat.*, vol. 9, no. 6, pp. 1135–1151, nov 1981.

[36] X. Zhu, "Semi-supervised learning with graphs," Ph.D. dissertation, Carnegie Mellon University, language technologies institute, school of computer science, 2005.

[37] M. Lebrun, M. Colom, A. Buades, and J. M. Morel, "Secrets of image denoising cuisine," *Acta Numer.*, vol. 21, pp. 475–576, may 2012.

[38] R. J. Tibshirani, J. Taylor *et al.*, "The solution path of the generalized lasso," *The Annals of Statistics*, vol. 39, no. 3, pp. 1335–1371, 2011.

[39] J. Sharpnack, A. Singh, and A. Rinaldo, "Sparsistency of the edge lasso over graphs," in *Artificial Intelligence and Statistics*, 2012, pp. 1028–1036.

[40] G. Puy, N. Tremblay, R. Gribonval, and P. Vandergheynst, "Random sampling of bandlimited signals on graphs," *Applied and Computational Harmonic Analysis*, pp. –, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1063520316300215

[41] P. di Lorenzo, S. Barbarossa, and P. Banelli, "Sampling and recovery of graph signals," in *Cooperative and Graph Signal Processing*. Elsevier, 2018, pp. 261–282.

[42] N. Tremblay, P. Gonçalves, and P. Borgnat, "Design of graph filters and filterbanks," in *Cooperative and Graph Signal Processing*. Elsevier, 2018, pp. 299–324.

[43] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[44] G. Turk and M. Levoy, "Zippered polygon meshes from range images," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 1994, pp. 311–318.

[45] M. Defferrard, L. Martin, R. Pena, and N. Perraudin, "Pygsp: Graph signal processing in python," *URL https://github. com/epfl-lts2/pygsp*, 2017.

[46] J. Xu and L. Zikatanov, "Algebraic multigrid methods," *Acta Numerica*, vol. 26, pp. 591–721, 2017.

[47] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[48] N. Anari, K. Liu, S. Oveis Gharan, and C. Vinzant, "Log-concave polynomials iv: Exchange properties, tight mixing times, and faster sampling of spanning trees," *CoRR*, vol. abs/2004.07220, 2020.

[49] A. Schild, "An almost-linear time algorithm for uniform random spanning tree generation," in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, 2018, pp. 214–227.

[50] K. D. Gremban, "Combinatorial preconditioners for sparse, symmetric, diagonally dominant linear systems," Ph.D. dissertation, Carnegie Mellon University, 1996.

[51] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu, "A simple, combinatorial algorithm for solving SDD systems in nearly-linear time," in *Proc. forty-fifth Annu. ACM Symp. Theory Comput.*, 2013, pp. 911–920.