THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : EEATS - Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)
Spécialité : SIGNAL IMAGE PAROLE TELECOMS
Unité de recherche : Grenoble Images Parole Signal Automatique

Algorithme de Wilson pour l'Algèbre Linéaire Randomisée

Wilsons Algorithm for Randomized Linear Algebra

Présentée par :

Yusuf Yigit PILAVCI

Direction de thèse :

Pierre-olivier AMBLARD Directeur de recherche,	Directeur de thèse
Nicolas TREMBLAY CNRS	Co-encadrant de thèse
Simon BARTHELME CNRS	Co-encadrant de thèse

Rapporteurs :

Samuel VAITER
CHARGE DE RECHERCHE, CNRS, Nice
Agnès DESOLNEUX
DIRECTEUR DE RECHERCHE, CNRS, Paris

Thèse soutenue publiquement le **15 novembre 2022**, devant le jury composé de :

Pierre-olivier AMBLARD DIRECTEUR DE RECHERCHE, CNRS, Grenoble	Directeur de thèse
Samuel VAITER CHARGE DE RECHERCHE, CNRS, Nice	Rapporteur
Agnès DESOLNEUX DIRECTEUR DE RECHERCHE, CNRS, Paris	Rapporteure
Olivier MICHEL PROFESSEUR DES UNIVERSITES, Grenoble INP	Examinateur
Alexandre GAUDILLIÈRE CHARGE DE RECHERCHE, CNRS, Aix Marseille	Examinateur
Nicolas COURTY PROFESSEUR DES UNIVERSITES, Université Bretagne Sud	Examinateur

Invités :

Nicolas Tremblay
CHARGE DE RECHERCHE, CNRS, Grenoble
Simon Barthelmé
CHARGE DE RECHERCHE, CNRS, Grenoble

Wilson's Algorithm for Randomized Linear Algebra

Yusuf Yiğit PİLAVCI

-

To those who still love poetry,
flowers and Turkish tea...

Abstract

An extensive range of problems in machine learning deals with *data structured over networks/graphs*. The examples vary from drug discovery to traffic forecasting, social network analysis to recommender systems, or epidemic analysis to natural language processing. Along with the exploding size and number of data, a big chunk of the proposed algorithms has focused on analyzing, representing and leveraging the network structures in the last decades. In many of them, the graph Laplacian is the central object. They are notable matrix representations of graphs that relate to various aspects. For example, by analyzing the Laplacian algebraically, one can measure the connectivity, count the number of spanning trees or generate a visualization of a graph. Further examples can be found in the problems of node clustering, graph sparsification, signal processing on graphs and many more. However, the algebraic analysis of Laplacian can be frustratingly time-consuming if the graph consists of a large number of nodes. Despite many numerical tools specialized for the graph Laplacian, in certain cases, the computational time remains one of the main issues.

Random spanning forests (RSFs), a random process on graphs, is a probabilistic tool for analyzing graphs with strong links with the Laplacian. In a nutshell, RSFs provide *meaningful* random sketches (spanning forests) of the graph to analyze some properties of interest. These sketches are cheap to obtain by a variant of Wilson's algorithm. Moreover, by only looking at a few of them, one can deduce significant statistical and/or algebraic information on the overall graph. The existing applications of RSFs, including graph wavelet analysis, semi-supervised learning on graphs or centrality analysis, show that RSFs provide useful information while binding diverse aspects of graphs.

In this manuscript, we present ways of leveraging the rich connections between RSFs and the graph Laplacian to speed up the algebraic analysis. In doing so, we propose randomized algorithms to approximate several quantities of interest, such as the regularized inverse of the Laplacian or effective resistances. Interestingly, these quantities are required in a wide range of applications on graphs, including graph signal filtering, hyper-parameter tuning, graph-based optimization and sparsification. We illustrate these methods on real-life networks and compare them with state-of-the-art methods.

Resumé

Un large éventail de problèmes d'apprentissage automatique traite des *données structurées sur des réseaux/graphes*. Les exemples vont de la découverte de médicaments aux prévisions de trafic, de l'analyse des réseaux sociaux aux systèmes de recommandation, ou de l'analyse des épidémies au traitement du langage naturel. Parallèlement à l'explosion des données, une grande partie des algorithmes proposés se sont concentrés sur l'analyse, la représentation et l'exploitation des structures de réseau au cours des dernières décennies. Dans beaucoup d'entre eux, les laplaciens de graphes sont les objets centraux. Ce sont des représentations matricielles notables des graphes qui se rapportent à divers aspects. Par exemple, en analysant les laplaciens de manière algébrique, on peut mesurer la connectivité, compter le nombre d'arbres spanning ou générer une visualisation d'un graphe. D'autres exemples peuvent être trouvés dans les problèmes de regroupement de nœuds, de sparsification de graphes, de traitement du signal sur les graphes et bien d'autres encore. Cependant, l'analyse algébrique du Laplacien peut prendre un temps frustrant si le graphe est constitué d'un grand nombre de nœuds. Malgré de nombreux outils numériques spécialisés pour les laplaciens, dans certains cas, le temps de calcul reste l'un des principaux problèmes.

Les Forêts Couvrante Aléatoire (FCA), un processus aléatoire sur les graphes, est un outil probabiliste pour analyser les graphes avec des liens forts avec les Laplaciens. En bref, les RSF fournissent des esquisses aléatoires du graphe pour analyser les propriétés d'intérêt. Ces esquisses sont faciles à obtenir par l'algorithme de Wilson. De plus, en ne regardant que quelques-uns d'entre elles, on peut déduire des informations statistiques et/ou algébriques significatives sur le graphe global. Les applications existantes des FCA, notamment l'analyse des ondelettes des graphes, l'apprentissage semi-supervisé sur les graphes ou l'analyse de la centralité, montrent que les FCA fournissent des informations utiles tout en liant divers aspects des graphes.

Dans ce manuscrit, nous présentons des moyens d'exploiter les riches connexions entre les FCA et les laplaciens des graphes pour accélérer l'analyse algébrique. Ce faisant, nous proposons des algorithmes aléatoires pour approcher plusieurs quantités d'intérêt, telles que l'inverse régularisé du Laplacien ou les résistances effectives. Il est intéressant de noter que ces quantités sont requises dans un large éventail d'applications sur les graphes, notamment le filtrage de signaux sur les graphes, l'ajustement d'hyperparamètres, l'optimisation basée sur les graphes et la sparsification. Nous illustrons ces méthodes sur des réseaux réels et les comparons aux méthodes de pointe.

Glossary

Acronyms

Acronym	Description
ADMM	Alternating Direction Method of Multipliers
CGD	Conjugate Gradient Descent
DPP	Determinantal Point Process
GD	Gradient Descent
GI	Graph Interpolation
GSP	Graph Signal Processing
GTR	Graph Tikhonov Regularization
LA	Linear Algebra
LASSO	Least Absolute Shrinkage and Selection Operator
LERW	Loop-erased Random Walks
LS	Least Squares
RLA	Randomized Linear Algebra
RSF	Random Spanning Forest
RST	Random Spanning Tree
SDD	Symmetric Diagonally Dominant
SOTA	State-Of-The-Art
USF	Uniform Spanning Forest
UST	Uniform Spanning Tree

Symbols

Symbol	Description
$\mathbf{x}, \mathbf{y}$	Vectors
A, B	Matrices
A_S	A submatrix of A obtained by reducing it to rows/columns indexed by S
$\mathrm{A}_{S\|T}$	A submatrix of A obtained by reducing it to rows indexed by S and columns indexed by T

Symbol	Description
$\mathsf{A}_{-S\|-T}$	A submatrix of A obtained by deleting rows indexed by S and columns indexed by T
$\mathsf{A}_{:\|-T}$	A submatrix of A obtained by taking all the rows and reducing it to the columns indexed by T
$\mathcal{G}$	A graph

Contents

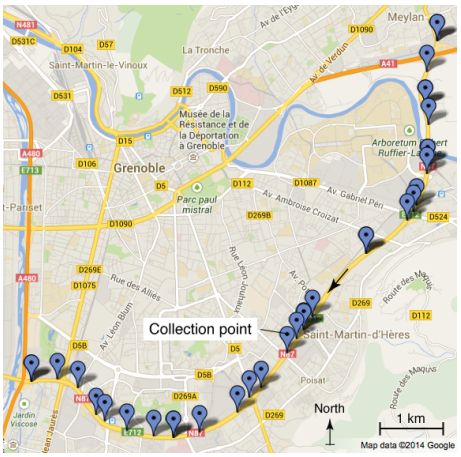
1 Introduction

> *"Graphs are ubiquitous structures."*
>
> **— Almost any paper on graph ML**

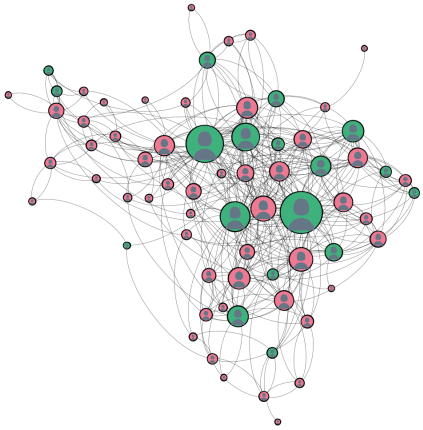
Since they first appeared for modeling the bridges of Königsberg, graphs, *i.e.*a set of vertices and edges, have been used to model and solve numerous problems in physics, chemistry, biology, computer science, social sciences and many more. They are so *ubiquitous* because they are natural models to use whenever the data represent mutual relations between individual elements. Today, processing data structured on graphs is one of the major research questions in signal processing and machine learning. Voluminous research focuses on leveraging the graph structure to accomplish advanced tasks, such as managing/analyzing Internet (a colossal computer network), understanding the human brain [FKL19], drug discovery [JWHCLWSCWH21], analyzing social networks [OR02] or large-scale traffic forecasting/planning [HLDNC17]. Further examples appear in natural language processing [NMR15], finance [MP20], food networks [DWM04], power grid networks [PA13] and many more. In many of these examples and others, graphs come with some features/signals, also called graph signals, on the vertex set. Here are some examples:

- Transportation networks [HLDN19]: A prominent example is road network. In a typical setup, each location corresponds to a node, and two nodes are connected by an edge whenever there is a road between them. The signals over such networks are usually some traffic features, such as the traffic speed or volume.

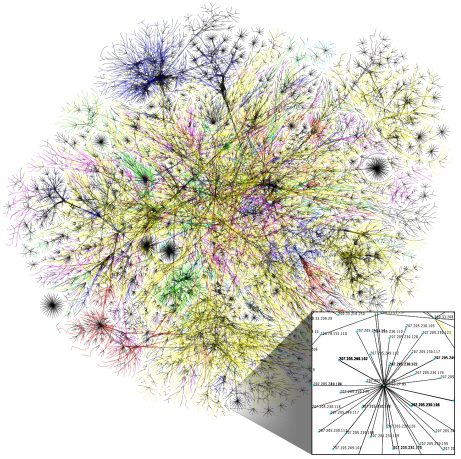
- Social networks [VMS21]: In many social studies, along with the individual information over the subjects, the structure of their interactions is also of central importance. Social media networks such as Facebook and Twitter are probably the most-known examples. In those cases, every user is considered a node and two nodes are connected if there is an interaction between the corresponding users, such as friends on Facebook or followers on Twitter. Given this graph/network structure, any feature of a user, such as their likes/dislikes, can be analyzed as a graph signal.



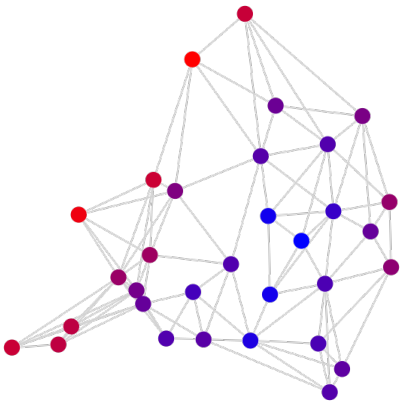
(a) Traffic data collected by a network of sensors in Grenoble [WMOKBB]



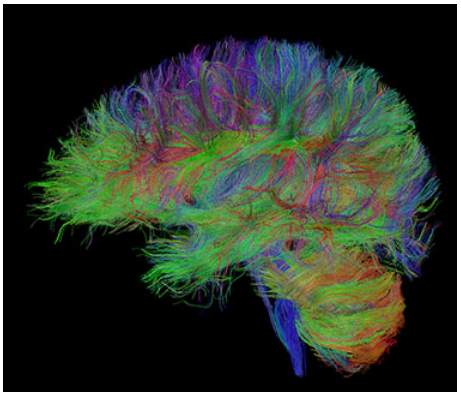
(b) Twitter friendship network of the authors of the tweets that the user @PilavciYigit liked until 29/06/2022.



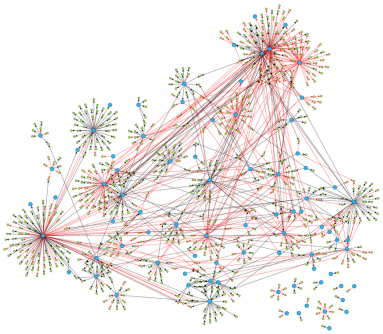
(c) A partial map of Internet



(d) Temperature sensor network over Bretagne/France [PV17]. The red colour depicts higher average temperatures.



(e) Human Brain Connectivity [HBMBRV17]



(f) Gene regulatory network [MKGS14]

Fig. 1.1.: *Real-life examples of graphs*

- Temperature networks [PV17]: A temperature network is a network of temperature sensors over a region in which each sensor is a node. Here the edges can be established according to different criteria. The common practice is to build the nearest neighbour graph from the proximity of the sensors. The signal, in this case, is naturally the temperature measurements collected from the sensors.

- Neurological networks [HBMBRV17]: Given numerous unanswered questions, the analysis of the brain is indeed an exciting and arduous task. The graph-based study, in fact, gives a natural way to analyze brain signals. In this type of analysis, we often consider the different regions of the brain as the nodes and the edges are established according to their structural or functional properties such as correlation in their activities. Analyzing neurological signals over such a network can decipher interesting information about the brain.

- Other biological networks [GDJSRLHVRT+21; CPFSC21; YB20]: The brain is not only the biological subject of graph-signal analysis. In the vast literature, we find many studies related to drug discovery, analysis of protein structures or gene interactions.

Given the large volume and diversity of such datasets, developing appropriate tools for processing and analyzing them has become quite central. In fact, research in machine learning and signal processing for graphs and graph signals has been emerging in the last decades [OFKMV18; RBTGV19]. In turn, these tools have been used for solving real-life problems varying from fake-news detection in social media [MFEMB19] to spread analysis for COVID-19 [PNV21], from decoding brain signals [OTLFPG22] to weather forecasting [Kei22], and so on. In most of these tools and methods, the matrix representations of graphs, especially the graph Laplacian, are of significant importance.

A Path Graph

Its Laplacian L

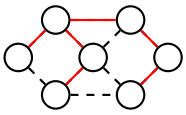
$$\begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

Graph Laplacian. Many analysis tools extract information from the matrix representations of the graphs, particularly from the graph Laplacian. The graph Laplacian is a symmetric square matrix in the size of number of the vertices. It can be considered as the discrete Laplacian operator on graphs when applied to a function over the vertices:

$$(\mathsf{L}f)_i = \sum_{j \in \mathcal{N}(i)} (f(i) - f(j)),$$

where $\mathcal{N}(i)$ denotes the nodes that share an edge with node i (formal definition can be found in Section 2.1.1). The graph Laplacian is a simple but useful matrix representation of graphs. The algebraic analysis relates them with many useful properties of graphs. For example, many tasks in connectivity analysis of the

graph can be accomplished by observing the eigendecomposition of the graph Laplacian. Simply by looking at the multiplicity of zero eigenvalues, one can count the number of connected components *i.e.* disconnected sub-parts of the graph. Moreover, the second smallest eigenvalue, also called spectral gap, gives a measure of the connectivity of the graph, which is mainly used in applications of stability and robustness of dynamic networks. Similarly, the corresponding eigenvector, also known as the Fiedler vector, yields the sparsest cut partition of the graph *i.e.* the partition that leaves the least number of edges inter-parts. Spectral graph drawing is another problem that uses the eigendecomposition of graph Laplacian. In this problem, one seeks a drawing (a mapping from vertices to Euclidean coordinates) of a graph that minimizes the distance between the vertices that are close to each other in the graph. It turns out that certain eigenvectors of graph Laplacian is the analytic solutions to this problem. Closer to the main themes of this thesis, we find an interesting link between the eigenvalues of L to the enumeration of a certain type of sub-graphs. In particular, the celebrated Kirchoff's matrix-tree theorem states that the product of the non-zero eigenvalues of L equals the number of all spanning trees, *i.e.* connected sub-graphs that contains no cycles, on a graph.



A spanning tree. Its edges are given in red, and the dashed lines indicate the edges of the graph.

Apart from these examples, the graph Laplacian has an important role in analyzing electrical resistor networks. In these networks, we model each resistor by an edge where the edge weight depicts the conductance of the resistor, and each node corresponds to its connection points. These models are extensively employed in circuit theory in order to calculate the currents flowing through the resistors/edges or the voltages induced at the nodes when a fixed potential difference (voltage) is applied between two (or more) nodes. Surprisingly, these calculations boil down to solving linear systems involving the graph Laplacian (See Chapter 4).

In the last decades, even more, examples have emerged in signal processing and machine learning involving graphs.

Laplacian in graph signal processing. Graph Signal Processing (GSP) is the sub-field of signal processing that deals with the signals defined over the vertices of the graph. In the last decade, many of the classical signal processing tools have been adapted in order to deal with such signals. Examples include filtering [SVF11; SNFOV13], sampling [TAB17; PTGV18], translation operation [SNFOV13], wavelet analysis [HVG11; NO12; ACGM20] or uncertainty principle [TBD16]. In many of these adaptations, the *graph Fourier transform* plays a significant role. By analogy, it allows to represent the graph signals in the graph frequency domain. In turn, one can define graph translation and filtering schemes by using an analogous of the celebrated convolution theorem [OBS01]. However, due to the irregularity of the

domain of the signal, the definition of the Fourier transform is not straight-forward. One popular definition in GSP builds an analogy on the fact that the eigenfunctions of the Laplacian operator are the basis functions in the Fourier transform. Developing on this, the authors in [SNFOV13] suggest using the eigenvectors of the graph Laplacian as the Fourier basis and the eigenvalues as the graph frequencies. In this way, higher eigenvalues correspond to a higher frequency component in the Fourier analysis. By projecting the signal on these basis, one can calculate the graph frequency response of the signal to each component.

Laplacian in machine learning. An early use of the graph Laplacian occurs in the semi-supervised learning on graphs [Zhu05; AMGS12]. In this problem, we are given a few labels over the vertices and the goal is to infer the labels of the others by using the given labels and the underlying graph. A baseline solution given by [Zhu05] suggest using the graph Laplacian to formulate this problem. Their main assumption is that the labeling function is fixed at a subset $V \subset \mathcal{V}$ and smooth over the rest $U = \mathcal{V} \setminus V$, *i.e.*it does not vary too much through the edges. For K classes of labels, this formulation seeks a classification function $F : \mathcal{V} \times \{1, \dots, K\} \to \mathbb{R}$ such that for each node i, $\operatorname{argmax}_k F(i,k)$ yields the label of node i. Let us denote the known labels by $\mathsf{Y} \in \mathbb{R}^{|V| \times K}$, then, [Zhu05] suggest solving the following constrained problem:

$$\begin{aligned} F^\star = &\operatorname*{argmin}_F \sum_{i \in U} \sum_{j \sim i} (F(i) - F(j))^2 \\ &\text{s. t. } \forall i \in \mathcal{V}, k \in \{1, \dots, K\}, F(i,k) = \mathsf{Y}_{i,k}, \end{aligned}$$

In fact, the minimized term can be written in terms of the graph Laplacian as $\mathsf{F}\mathsf{L}_U\mathsf{F}$ where $\mathsf{F} = [F(i,k)]_{i \in U, k \in \{1,\dots,K\}}$. Moreover, the closed form solution involves the following in the graph Laplacian:

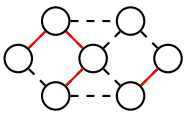
$$F^\star = (\mathsf{L}_U)^{-1} \mathsf{L}_{U|V} \mathsf{Y}.$$

Later on, this formulation is considered in an unconstrained setup and generalized in [AMGS12]. In these forms as well, the solution takes the form of the matrix inverse of the graph Laplacian. More recent examples in machine learning that extensively use graph Laplacian include spectral clustering [Von07; TPGV16], graph embedding [Xu21], sparsification [SS11] and deep learning on graphs [WPCLZP20]. In all of these applications and many more, the analysis of L is of utmost importance.

Laplacian-based Numerical Algebra. As listed above, the number of applications involving the graph Laplacian is myriad. In many of these applications, the solution

requires computing an (regularized or pseudo) inverse or eigendecomposition of the graph Laplacian. However, computing these quantities directly does not scale well with the size of the graph and even becomes impractical for very large graphs. In turn, many researchers have focused on developing numerical algebra tools specified for Laplacian. The most prominent ones are closely related to spectral graph theory [Spi12] and solving Laplacian linear systems [Vis+13]. These studies give a rich collection of algebraic tools that avoid the expensive direct computation and, instead, approximate the required solution within certain error ranges. In doing so, they leverage some common properties of the graph Laplacian, such as its algebraic properties, its strong links with graph theory, or the sparsity of real-life graphs. Compared to these works, we take in this thesis an alternative path by using randomization in order to develop Laplacian-specific algebraic tools.

Randomized Linear Algebra (RLA). In a nutshell, RLA is a set of tools that avoid the direct computation of an algebraic operation via randomization. The main advantage of these tools is that they can approximate very expensive operations such as matrix inversion, and singular value decomposition, very cheaply by only taking random samples from the input matrix. More details and examples can be found in [DM16]. Closer to the themes of this thesis, [DM21] show that determinantal point processes (DPP), *i.e.* a random point process with many tractable properties, can be easily adapted as RLA algorithms in order to solve linear systems involving a large set of input matrices covering the graph Laplacian. Inspired by these examples and many more, we focus on a particular determinantal point process that allows us to develop randomized tools for Laplacian-based numerical algebra.



A spanning forest with three trees. The edges of the forest are in red, and the graph is depicted by dashed lines.

Random Spanning Forests (RSF). A forest in a graph is a subgraph which contains no cycle in it. It is called spanning if it contains all the vertices of the graph. Finally, the random spanning forests are the random processes where we choose spanning forest at random on a given graph. One can come up with infinitely many options for the distribution of such a process. In this thesis, we focus on the distribution introduced in [ACGM18]. In this distribution, the probability of having a fixed spanning forest is mainly regulated by the weights of its edges and the number of its connected components (trees). We restrict ourselves to this distribution due to two main reasons:

- The RSFs sampled from this distribution have rich theoretical connections with the graph Laplacian via certain DPPs,
- There exists an efficient algorithm to sample from this distribution, called Wilson's algorithm.

Thanks to these facts, we know that the samples of RSFs can be obtained cheaply, and their algebraic properties closely relate to the algebraic properties of the original graph. The main goal of this thesis is to leverage these facts in order to develop RLA algorithms for Laplacian-based numerical algebra.

The main contributions of this thesis can be listed as follows:

- We shed light on the rich connections of RSFs with determinantal point processes and the graph Laplacian; We go through the elegant theory and useful properties of RSFs as DPPs. Throughout our tour, while for some of the properties, we reproduce the existing proofs, for some others, we give mostly algebraic proofs that are more accessible for the researchers from different domains (See Theorems 2.4.4, 2.4.5).

- In turn, we develop and analyze RLA algorithms based on RSFs for approximating the solutions of a wide variety of problems involving the graph Laplacian. These problems are:

  - **Tikhonov regularization and interpolation for graph signals**: The problem of denoising, *i.e.* removing noisy parts of a signal, and completing missing parts of a signal are long-standing problems in signal processing. We first give a unifying optimization framework for these problems for graph signals *i.e.* signals defined over vertices. Then, we give novel randomized algorithms that approximate the solution for this framework.

  - **Estimating the trace of regularized inverse of symmetric diagonally dominant (SDD) matrices**: Trace (sum of diagonal entries of a matrix) is a central operation in linear algebra. However, computing the trace of a matrix is a challenging problem whenever the input matrix is not directly accessible *e.g.* multiplication of large matrices or inverse of a large matrix. We give novel algorithms based on RSFs for estimating the regularized inverse of SDD matrices (a class of matrices that contain the graph Laplacian). In turn, our algorithms are at least comparable, and they usually outperform state-of-the-art methods.

  - **Estimating effective resistances in electrical networks**: Effective resistance is a metric that takes its root from electrical resistor networks[1] and they are used for measuring the similarity of a pair of nodes. They play a central role in many graph-related applications such as graph sparsification, clustering or graph learning. However, the computation of this

[1] Graph representation of an electrical resistor networks takes every edge as a resistor with a unit conductance. The conductance equals the weight of the edge for weighted graphs.

useful quantity does not scale well with increasing sizes of graphs as one needs to invert a graph Laplacian. To avoid this expensive computation, we give RSF-based algorithms to estimate effective resistances. The proposed methods are comparable and even better than the state-of-the-art algorithms when the number of desired effective resistances is small.

  - **Estimating various graph filters:** This aspect of our work makes interesting links between RSFs and graph filtering *i.e.* spectral signal filtering for graph signals [TGB18]. Graph filtering is an essential tool for dealing with these type of signals and have been used in various applications. Yet again, the associated computations often require diagonalizing L, which is expensive. In this work, we give a set of graph filters that can be approximated via RSF-based estimators by avoiding the diagonalization of L. These filters either cover or can mimic by optimizing a few hyperparameters some of the frequently used filters, such as the ideal low-pass filter.

- We provide a bias-variance analysis of the proposed algorithms for stating their theoretical performance.

- We illustrate these algorithms in real-life applications and datasets while comparing these algorithms with the existing algorithms.

The main structure of the thesis is as follows; we start by giving necessary technical machinery in Chapter 2 on graph theory, randomized linear algebra and random spanning forests. Then, by Chapter 3, we introduce the RSF-based methods for solving graph Tikhonov regularization. We compare these methods with existing methods and show their uses in other graph-related problems while illustrating real-life applications. In Chapter 4, we extend the range of RSFs-based methods into a new set of problems, including estimating the trace of regularized inverse of SDD matrices, effective resistances and certain graph filters. Finally, we finish in Chapter 5 with a general conclusion, discussion on open questions and future works.

The peer-reviewed publications associated with this thesis can be found in the following:

- Yusuf Yigit Pilavci et al. “Variance Reduction for Inverse Trace Estimation via Random Spanning Forests”. In: *GRETSI 2022 - XXVIIIème Colloque Francophone de Traitement du Signal et des Images*. Nancy, France, Sept. 2022

- Yusuf Pilavcı et al. “Variance reduction in stochastic methods for large-scale regularised least-squares problems”. In: *arXiv preprint arXiv:2110.07894* (2021)
- Yusuf Yiğit Pilavcı et al. “Graph tikhonov regularization and interpolation via random spanning forests”. In: *IEEE transactions on Signal and Information Processing over Networks* 7 (2021), pp. 359–374
- Yusuf Y Pilavci et al. “Smoothing graph signals via random spanning forests”. In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 5630–5634

Also, some materials are introduced in my master thesis [PIL19].

Background

2

> *Mathematics compares the most diverse phenomena and discovers the secret analogies that unite them.*
>
> **— Joseph Fourier**

In this chapter, we go through the necessary technical definitions and tools to be able to explain the main contributions of this thesis. We collect these diverse materials under four sections:

- **Graph theory** contains the basic definition of a graph, graph matrices (*e.g.* adjacency, Laplacian etc.), graph-related structures (*e.g.* walks, paths, cycles, trees, forests etc.), random processes on graphs.
- **Determinantal Point Process (DPP)** contains the basic definitions on DPP and their useful properties.
- **Randomized Linear Algebra (RLA)** presents the use of randomization for numerical linear algebra with emphasis on least-squares problems.
- **Random Spanning Forest (RSF)** introduces a random object on graphs, called random spanning forests, which has fascinating probabilistic properties.

As they are listed above, these concepts might seem unconnected. However they have elegant mathematical links that tie them to each other. One of the main goals of this chapter is to bring these links to spotlight.

Contents

2.1 Graph Theory

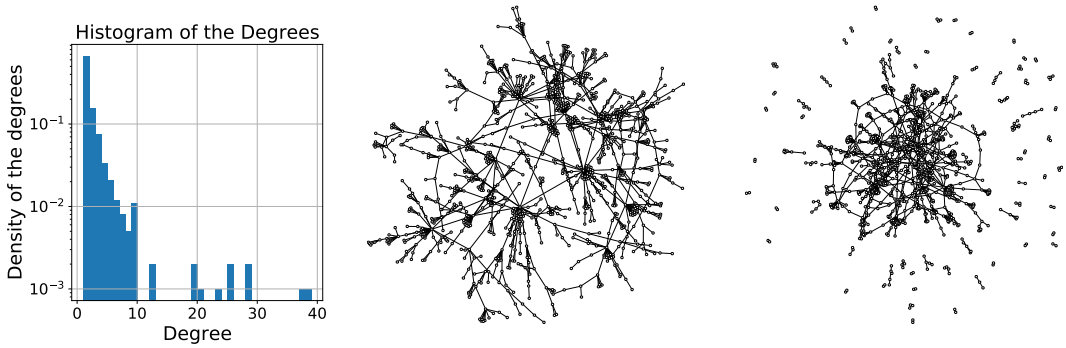
We dedicate this section to defining all graph-related objects, which will be repeatedly used in the rest.

Definition 2.1.1 (Graph)**.** A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ is a triple which consists of n nodes/vertices $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$, m edges connecting the vertices $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ and the weight function $w : \mathcal{V} \times \mathcal{V} \to \mathbb{R}_{\geq 0}$ which maps every edge $(i, j) \in \mathcal{E}$ to a positive real value, and $w(i, j) = 0$ for any pair $(i, j) \notin \mathcal{E}$.

In many problems of physics, mathematics, biology, computer science and machine learning, the data are known up to pairwise relations of data points. In such cases, graphs/networks are natural choices to interpret, visualize or analyze the data. In doing so, certain distinctions over graphs facilitate the understanding of real-life networks. Let us list them in the following paragraphs:

A *directed* graph is a graph in which every edge (i, j) has an orientation from i to j and $w(i, j)$ is not necessarily equal to $w(j, i)$. If all edges in $\mathcal{G}$ verifies $w(i, j) = w(j, i)$, then we call it an *undirected* graph as the orientation of the edges are no longer informative about the graph. The weight function w of a graph $\mathcal{G}$ maps every edge to either 1 or 0, *i.e.* $\forall (i, j) \in \mathcal{E},\ w(i, j) \in \{0, 1\}$, then $\mathcal{G}$ is called *unweighted*. We denote the neighbors of a node i as $\mathcal{N}(i) \coloneqq \{j : j \in \mathcal{V} \text{ and } (i, j) \text{ or } (j, i) \in \mathcal{E}\}$, *i.e.* the set of nodes which has an incident edge (j, i) or (i, j) in $\mathcal{G}$. The sum $d_i \coloneqq \sum_{j \in \mathcal{N}(i)} w(i, j)$ is called the *degree* of node i.

The degrees $d_1, d_2, \dots, d_n$ are limited but useful devices for characterizing graphs. They are often analyzed by statistical tools. In particular, the degree sequence



(a) Degree Distribution **(b)** Barabasi-Albert model **(c)** Kalisky model

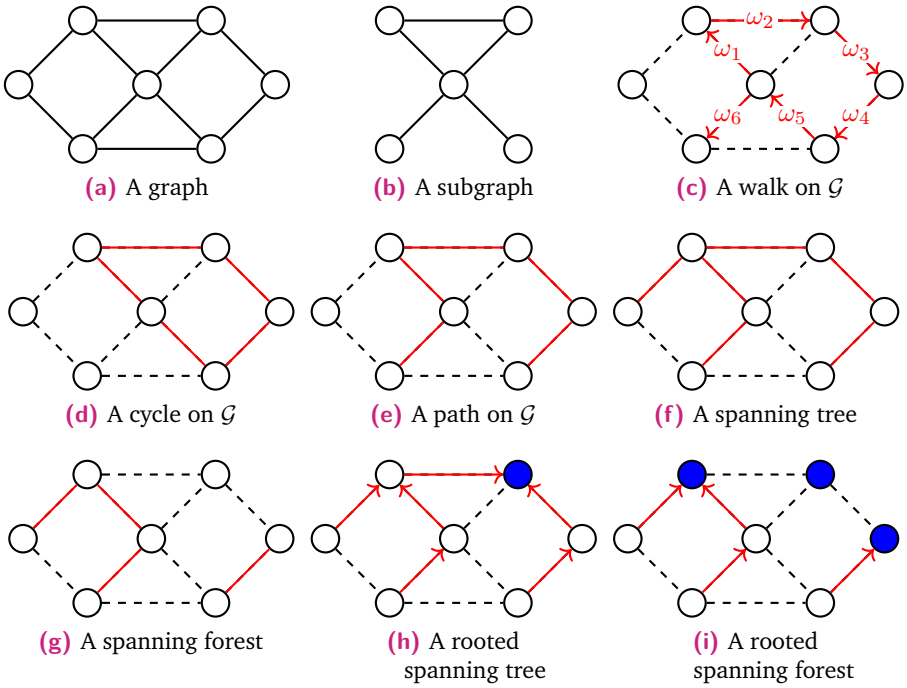
Fig. 2.1.: *One can obtain completely different graphs for the same degree distribution. First, we generate a graph (b) from Barabasi-Albert model [AB02] (a statistical model for generating a random graph) with $n = 1000$ and the model parameter $k = 1$. Then, we pass its degree distribution given in (a) to the Kalisky model [KCbH04] (a statistical model for generating a graph given a degree distribution) to generate another graph (c) that has the same degree distribution.*

$d_1, d_2, \ldots, d_n$ of a graph is considered as a sample of a probabilistic model, also called a degree distribution. In this way, one can reduce large and complex real-life networks to some degree distributions with a few hyper-parameters. However, when it comes to detecting finer details of the graph, the degree distribution might fail. We illustrate such a case with a toy example in Fig. 2.1. In a nutshell, we show that we can obtain two graphs with completely different characteristics whereas they have exactly the same degree sequence. In this case, it makes sense to take our focus on some of the substructures of a graph that contain descriptive patterns, such as connected components, paths, cycles or trees. We finish this section by giving the formal definitions of some of such substructures.

Definition 2.1.2 (Subgraph)**.** A graph $\mathcal{H} = (\mathcal{V}_s, \mathcal{E}_s, w_s)$ is a subgraph of a graph $\mathcal{G}$, if it verifies $\mathcal{V}_s \subseteq \mathcal{V}$, $\mathcal{E}_s \subseteq \mathcal{E}$ and $w_s(i,j) = w(i,j)$ for all (i,j) pairs.

Definition 2.1.3 (Walk)**.** A walk on $\mathcal{G}$ starting from node s and ending at node t is a sequence of edges $\omega = (e_1, e_2, \ldots, e_l)$ which joins a sequence of vertices $(s, \ldots, t)$.

Definition 2.1.4 (Cycle)**.** A cycle is a walk whose starting and ending point is the same node.



(a) A graph **(b)** A subgraph **(c)** A walk on $\mathcal{G}$

(d) A cycle on $\mathcal{G}$ **(e)** A path on $\mathcal{G}$ **(f)** A spanning tree

(g) A spanning forest **(h)** A rooted spanning tree **(i)** A rooted spanning forest

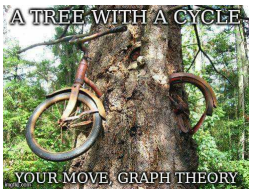
Fig. 2.2.: Given a graph in (a), this figure illustrates a subgraph (b), a walk (c), a cycle (d), a path (e), a spanning tree (f), a rooted spanning tree (g) and a rooted spanning forest (h). The blue nodes in (g) and (h) indicates the root nodes.

Definition 2.1.5 (Path)**.** A path is a walk which does not include any cycle.

Definition 2.1.6 (Connectivity)**.** Any two nodes s and t are called connected whenever there exists at least one path in $\mathcal{G}$ that joins them. Otherwise, they are called disconnected.

Definition 2.1.7 (Connected Component)**.** A connected component of a graph is a maximal connected subgraph.

Definition 2.1.8 (Spanning Tree)**.** A tree, denoted by $\tau = (\mathcal{V}_\tau, \mathcal{E}_\tau, w)$, is a subgraph which contains no cycle in it. If $\mathcal{V}_\tau = \mathcal{V}$, then τ is called a spanning tree.



Taken from short-url.at/cfkpu

Definition 2.1.9 (Rooted Spanning Tree)**.** A rooted tree is a directed tree in which every edge is oriented towards a special node called the root.

Definition 2.1.10 (Rooted Spanning Forest)**.** A rooted forest, denoted by $\phi = (\mathcal{V}_\phi, \mathcal{E}_\phi, w)$, in $\mathcal{G}$ is a set of disjoint rooted trees with $\mathcal{V}_\phi \subseteq \mathcal{V}$, $\mathcal{E}_\phi \subseteq \mathcal{E}$. Similarly, if $\mathcal{V}_\phi = \mathcal{V}$, ϕ is called a rooted spanning forest.

We illustrate some of these graph structures in Fig. 2.2

2.1.1 Graph Matrices

Matrix representations of graphs are helpful tools to examine probabilistic, algorithmic and combinatorial aspects of graphs. In this section, we will revisit the essential matrix representations of graphs.

The simplest matrix representation of a graph is the (weighted) adjacency matrix, defined as $\mathsf{W} \coloneqq [w(i,j)]_{i,j} \in \mathbb{R}^{n\times n}$. Another simple matrix, called the degree matrix, contains the degrees in a diagonal matrix D where $\forall i \in \mathcal{V}$, $\mathsf{D}_{i,i} \coloneqq d_i$. Finally, we define the combinatorial graph Laplacian as $\mathsf{L} \coloneqq \mathsf{D} - \mathsf{W}$. One can find various ways to define the Laplacian operator for graphs, such as the normalized Laplacian $\mathsf{I} - \mathsf{D}^{-\sigma}\mathsf{W}\mathsf{D}^{\sigma-1}$ with $\sigma \in \{0, 1\}$. In this thesis, we refer to L as the graph Laplacian unless otherwise stated. We stress that there are various theoretical properties of L which connect diverse concepts such as determinantal point process [BP93], combinatorics [CK78], computer science and graph signal processing [SNFOV13]. This thesis aims to give a unifying view of these concepts and leverage the connections for randomized linear algebra applications. Thus L is an object of central importance.

The edge incidence matrix $\mathsf{B} \in \mathbb{R}^{m\times n}$ is a less common but still useful representation of graphs. It depicts the orientation of the edges as follows:

$$\forall i, j, \quad \mathsf{B}_{k,i} = \begin{cases} -\sqrt{w(i,j)}, & e_k = (i,j) \in \mathcal{E} \\ \sqrt{w(j,i)}, & e_k = (j,i) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases}$$

In case of undirected graphs, we retain B in this form by encoding every undirected edge $e_k = (i, j)$ as $\mathsf{B}_{k,i} = -\sqrt{w(i,j)}$ and $\mathsf{B}_{k,j} = \sqrt{w(i,j)}$, for all $i < j$ without loss of generality. The following identity relates B with the graph Laplacian L:

$$\mathsf{L} = \mathsf{B}^\top \mathsf{B}. \tag{2.1}$$

This relation also proves the positive semi-definiteness of L.

Spectral properties, *i.e.* eigen/singular value decomposition, of these matrices, are studied within many fields, from signal processing [SNFOV13] to clustering [Von07]. Similarly, they are of central importance in this thesis. Therefore, we revisit some of these properties and their uses in the rest.

We generically write the spectral decomposition of W as $\mathsf{W} = \mathsf{P}\Xi\mathsf{R}^\top$ where $\mathsf{P} = [\mathbf{p}_1|\dots|\mathbf{p}_n]$ and $\mathsf{R} = [\mathbf{r}_1|\dots|\mathbf{r}_n]$ contain the left and right eigenvectors $\mathbf{p}_i \in \mathbb{C}^n$ and $\mathbf{r}_i \in \mathbb{C}^n$, and Ξ is the diagonal matrix containing the eigenvalues $\xi_1 \leq \xi_2 \leq \dots \leq \xi_n$. For an undirected graph, the adjacency matrix W is entry-wise nonnegative and symmetric, thus $\mathsf{P} = \mathsf{R}$. The spectrum of W is insightful for understanding the structure of the graph. For example, the multiplicity of the largest eigenvalue gives the number of connected components in the graph. Another example is that if the eigenvalues are symmetric with respect to 0, *i.e.* $\forall i \in 1, \dots, n,\ \xi_i = -\xi_{n-i}$, then the graph is bipartite.

As W is an entry-wise non-negative matrix, it is possible to examine its spectrum with the celebrated Perron-Frobenius theorem [Per07; FFFFM12]. Along with other properties, the Perron-Frobenius theorem associates the largest eigenvalue with a positive eigenvector. An influential application of this property in directed graphs takes place in the famous algorithm of Google, called `PageRank` [PBMW99] for sorting web pages by their relevance given their hyperlink graph *i.e.* whenever a page links to another, there is a directed edge between them. `PageRank` approximates different websites according to their ranking score, which is based on the principal eigenvector of the graph of the world wide web (WWW) (See Example 2.1.1).

A $25 Billion Algorithm[a]

[a] The approximate market value of Google in 2004 [BL06]

Information retrieval *i.e.* retrieving the relevant information in a humongous amount of data is a long-standing problem in computer science. In the last two decades, research in this field has provided many efficient tools with the help of machine learning and computer science. Google's web search engine has been probably the most popular one. In the early stages of search

engines, one main issue was efficiently ordering different web pages in search results according to their relevancy. In [PBMW99], the founders of Google proposed the celebrated `PageRank` algorithm to rank web pages, which relies on probability and algebraic graph theory.

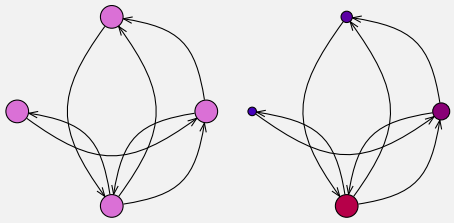


Fig. 2.3.: *Given a graph (in the left), PageRank is an algorithm to rank nodes according to their importance in the graph. In the right, we plot the graph by resizing the nodes according their PageRank scores. Bigger nodes have higher scores.*

Consider the hyperlink graph of the world wide web (WWW) in which each website is a node, and there is a directed edge whenever there is a hyperlink from one to another. Given such a graph, one seeks to assign a ranking score r_i to each node/web page i. The `PageRank` algorithm proposes to use the ranking scores r_i's which verify:

$$\forall i \in \mathcal{V}, \quad r_i = \frac{1}{d_i} \sum_{j \in \mathcal{N}(i)} w(i,j) r_j \geq 0.$$

Intuitively, the ranking scores of its neighbours define the ranking score of a node. The web pages pointed by the pages with high scores have higher scores. Besides, this formulation corresponds to solving the following eigenvector problem:

$$\mathbf{r} = \mathsf{D}^{-1}\mathsf{W}\mathbf{r}.$$

By the Perron-Frobenius theorem, we know the largest eigenvalue is 1[a] and it is associated with an entry-wise positive eigenvector $\mathbf{r}$. `PageRank` draws a simple iterative scheme, also called the Power method, to estimate $\mathbf{r}$ as follows:

$$\mathbf{r}^{(k+1)} = \left(\mathsf{D}^{-1}\mathsf{W}\right)\mathbf{r}^{(k)},$$

starting from arbitrary initialization $\mathbf{r}^0$. As k goes to infinity, $\mathbf{r}^k$ converges to $\mathbf{r}$. An illustration of this algorithm is in Fig. 2.3.

[a] This is due to the bounds on the spectral radius given by the Perron Frobenius theorem *i.e.* $\min_i \sum_{j=1}^n (\mathsf{D}^{-1}\mathsf{W})_{i,j} = 1 \leq \rho(\mathsf{D}^{-1}\mathsf{W}) \leq \max_i \sum_{j=1}^n (\mathsf{D}^{-1}\mathsf{W})_{i,j} = 1$

Similar to the adjacency matrix, the spectrum of the (combinatorial) graph Laplacian L also provides many interesting insights into the graph. We detail some of them as they are of central importance for the main concepts of this thesis. Let us denote the eigen-decomposition by $\mathsf{L} = \mathsf{U}\Lambda\mathsf{U}^\top$ where $\mathsf{U} = [\mathbf{u}_1|\dots|\mathbf{u}_n] \in \mathbb{R}^{n\times n}$ contains the eigenvectors $\mathbf{u}_i$'s in the columns and Λ is the diagonal matrix which consists of the eigenvalues $\lambda_1 = 0 \leq \lambda_2 \leq \dots \leq \lambda_n$ in its diagonal entries. Note that L is a positive semi-definite matrix as $\mathsf{L} = \mathsf{B}^\top\mathsf{B}$, and its smallest eigenvalue is 0. The multiplicity of the smallest eigenvalue at 0 equals the number of connected components. Moreover, the second smallest eigenvalue λ_2, also called algebraic connectivity or Fiedler value, is considered a measure of the connectivity of the graph. For example, if the graph has two connected components, then one also has $\lambda_2 = 0$, and it increases as one adds edges connecting the two components. Another significant property of L is due to the matrix-tree theorem [Kir47]:

Theorem 2.1.1 (Matrix-Tree Theorem)**.** *Given a graph Laplacian* L *of a connected, unweighted and undirected graph, let* $\mathcal{T}$ *be the set of all spanning trees. Then, one has:*

$$\forall i \in \mathcal{V}, \quad |\mathcal{T}| = \det \mathsf{L}_{-i|-i}$$

where $\mathsf{L}_{-i|-i}$ *is the submatrix of* L *obtained by deleting* i*-th row and column.*

Proof. Among many possible proofs of the matrix-tree theorem, we prefer to provide an algebraic one which will be re-invoked in the further chapters. The Cauchy-Binet determinant formula yields:

$$\begin{aligned}\det \mathsf{L}_{-i|-i} &= \sum_{\substack{S\subseteq\mathcal{E}\\|S|=n-1}} \det \mathsf{B}_{S|-i} \det(\mathsf{B}^\top)_{S|-i},\\ &= \sum_{\substack{S\subseteq\mathcal{E}\\|S|=n-1}} (\det \mathsf{B}_{S|-i})^2.\end{aligned}$$

Now we invoke a technical result from [Van10] which is attributed to Poincaré. To do so, we consider the square submatrices of the edge incidence matrix B in size of $(n-1)\times(n-1)$. Notice that the rows and columns of such submatrices are respectively indexed by $n-1$ edges and nodes of the graph. Let us denote these edges and nodes by $S \subset \mathcal{E}$ and $-i = \mathcal{V} \setminus i$. Theorem 2 in [Van10] states

that $\det \mathsf{B}_{S|-i}$ equals to ± 1 if and only if S is a spanning tree of $\mathcal{G}$. Otherwise it equals to 0. Plugging this result in the Cauchy-Binet formula finishes the proof:

$$\det \mathsf{L}_{-i|-i} = \sum_{\substack{S \subseteq \mathcal{E} \\ |S|=n-1}} (\det \mathsf{B}_{S|-i})^2 = \sum_{\substack{S \subseteq \mathcal{E} \\ |S|=n-1}} \mathbb{I}(S \text{ is a spanning tree})$$

where $\mathbb{I}$ is the indicator function. □

At last but not least, we detail the spectral decomposition of B. Let us denote its singular value decomposition by $\mathsf{B} = \mathsf{V}\mathbf{\Sigma}\mathsf{U}^\top$ where $\mathsf{V} = [\mathbf{v}_1|\dots|\mathbf{v}_n] \in \mathbb{R}^{m\times m}$ and $\mathsf{U} = [\mathbf{u}_1|\dots|\mathbf{u}_n] \in \mathbb{R}^{n\times n}$ are the orthonormal basis and $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ is a rectangular diagonal matrix which contains the singular values $\sigma_1 \leq \dots \leq \sigma_n$ in its diagonal entries. Note that U is also the orthonormal basis of L and $\forall i \in \mathcal{V}$, one has $\lambda_i = \sigma_i^2$ as an immediate result of the identity in (2.1). As mentioned in the proof of Theorem 2.1.1, Poincaré's theorem [Van10] makes interesting connections between the edge incidence matrix B and the spanning trees of the corresponding graph. These links are of central importance for this thesis. Thus, to conclude our visit on algebraic graph theory, we reproduce and extend this theorem for weighted graphs and spanning forests.

Proposition 2.1.2 (Poincaré for weighted graphs). *Consider a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ with an edge incidence matrix B. If $S \subseteq \mathcal{E}$ with $|S| = |\mathcal{V}| - 1$ is the edges of a spanning tree, then, $|\det \mathsf{B}_{S|-m}| = \left[\prod_{(i,j)\in S} w(i,j)\right]^{1/2}$ for all $m \in \mathcal{V}$. Otherwise, $|\det \mathsf{B}_{S|-m}| = 0$.*

Proof. See App. A.2. □

Developing on this theorem gives the following extension on spanning forests:

Theorem 2.1.3 (Poincaré for forests). *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ with an edge incidence matrix B, consider a vertex subset $R \subseteq \mathcal{V}$ and an edge subset $S \subseteq \mathcal{E}$ with $|S| = |\mathcal{V}| - |R|$. If S is an edge subset such that*

- *S forms a spanning forest with $|R|$ trees,*
- *S does not form any path between the nodes of R (for rooted forests, R is the root set.),*

then, $|\det \mathsf{B}_{S|-R}| = \prod_{r\in R}\left[\prod_{a,b\in S_r} w(a,b)\right]^{1/2} = \left[\prod_{a,b\in S} w(a,b)\right]^{1/2}$ *where* S_r *is the edge set of the tree that contains node* $r \in R$ *and* $S = \bigcup_{r\in R} S_r$*. Otherwise, this absolute determinant is equal to* 0*.*

Proof. As long as S forms a spanning forest, it includes $|R|$ disconnected trees due to the fixed number of edges in trees (*i.e.*a tree with t nodes consists of exactly $t-1$ edges). Similar to Theorem 2.1.2, the rest of the proof is also two-fold: In the first part, we consider the case that the given conditions hold. In the second, otherwise is examined to finish the proof.

Assume that S forms a spanning forest with $|R|$ trees and it does not include any path between the nodes in $R = \{r_1, \ldots, r_{|R|}\}$. Then, each tree in such forest contains exactly one distinct node $r \in R$. This allows us to write a row-column permutation of square matrix $\mathsf{B}_{S|-R}$ in the following form:

$$\operatorname{perm}(\mathsf{B}_{S|-R}) = \begin{bmatrix} \mathsf{B}^{(r_1)}_{S_{r_1}|-r_1} & & & \\ & \mathsf{B}^{(r_2)}_{S_{r_2}|-r_2} & & \\ & & \ddots & \\ & & & \mathsf{B}^{(r_{|R|})}_{S_{r_{|R|}}|-r_{|R|}} \end{bmatrix}$$

where $\operatorname{perm}(\mathsf{B}_{S|-R})$ is a particular row-column permutation of $\mathsf{B}_{S|-R}$ in which each tree's reduced edge incidence matrix $\mathsf{B}^{(r)}_{S_r|-r}$ appears in the diagonals. The absolute determinant of this block diagonal matrix writes:

$$|\det\left(\operatorname{perm}(\mathsf{B}_{S|-R})\right)| = \prod_{r\in R} |\det \mathsf{B}^{(r)}_{S_r|-r}|$$

where $|\det \mathsf{B}^{(r)}_{S_r|-r}|$ is equal to $\left[\prod_{a,b\in S_r} w(a,b)\right]^{1/2}$ due to Prop. 2.1.2. Noticing $|\det\left(\operatorname{perm}(\mathsf{B}_{S|-R})\right)| = |\det \mathsf{B}_{S|-R}|$ writes:

$$|\det \mathsf{B}_{S|-R}| = \prod_{r\in R}\left[\prod_{a,b\in S_r} w(a,b)\right]^{1/2} = \left[\prod_{a,b\in S} w(a,b)\right]^{1/2}$$

and finishes the first part of the proof. The contrary of the constraints given in the theorem yields either S includes a cycle or there exists a path between the nodes in R. In the former case, $|\det\left(\operatorname{perm}(\mathsf{B}_{S|-R})\right)| = \prod_{r\in R} |\det \mathsf{B}^{(r)}_{S_r|-r}|$ results

in 0 because whenever a subset S_r contains a cycle, $|\det \mathsf{B}^{(r)}_{S_r|-r}|$ becomes 0 due to Theorem 2.1.2. For the latter, we consider a generic path between the nodes in R as shown in Fig. 2.4.

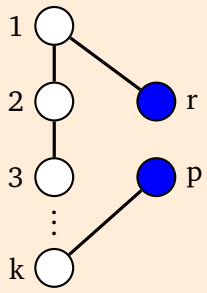


Fig. 2.4.: A generic path between nodes in R (in blue)

The reduced edge incidence matrix for a graph with this path writes:

$$\mathsf{B}_{\{S|-R\}} = \begin{bmatrix} w(r,1) & 0 & \dots & & 0 & \\ -w(1,2) & w(1,2) & 0 & \dots & 0 & \\ 0 & -w(2,3) & w(2,3) & 0\dots & 0 & 0 \\ \vdots & & & \ddots & & \\ 0 & \dots & & 0 & -w(k,p) & \\ & & \mathsf{X} & & & \mathsf{Y} \end{bmatrix}$$

Regardless of matrices X and Y (and so, the rest of the graph), we can conclude that this matrix is singular (*i.e.* $\det \mathsf{B}_{S|-R} = 0$) because it has a non-zero right null vector $\mathbf{x}^T = \begin{bmatrix} 1 & \frac{w(r,1)}{w(1,2)} & \frac{w(r,1)}{w(2,3)} & \dots & \frac{w(r,1)}{w(k,p)} & \mathbf{0} \end{bmatrix}$ which gives $\mathbf{x}^T \mathsf{B}_{S|-R} = \mathbf{0}^T$. Notice that a direct edge from r to p is a particular case of this generic path which generates a zero row in $\mathsf{B}_{S|-R}$. Moreover, choosing $r = p$ does not alter the given matrix form. This completes the second part and the proof. □

Remark 1 (Switching to rooted trees/forests)**.** Notice that in both theorems (and also for the whole thesis), we work on undirected graphs. Thus we omit the orientation within the trees and forests for the sake of simplicity. However these objects are still considered to be rooted trees/forests by setting the corresponding orientation. For example, in Theorem 2.1.1, the root of the spanning trees counted by the determinant is the deleted row/column index i *i.e.* $|\mathcal{T}_i| = \det \mathsf{L}_{-i|-i}$. This is correct because the orientation of the edges to make node i a root in a spanning tree is unique (See Fig 2.5 for an example). Thus, one has $|\mathcal{T}_i| = |\mathcal{T}| = \det \mathsf{L}_{-i|-i}$. In the case of forests and Theorem 2.1.3, the vertex set R is considered as the root set

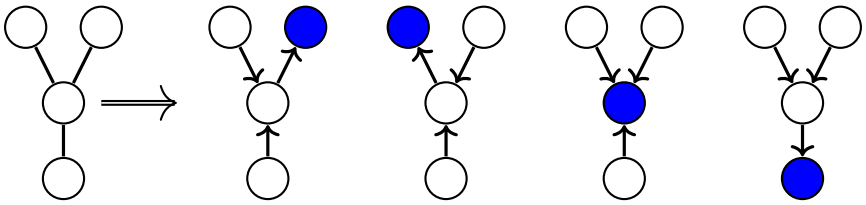


Fig. 2.5.: Given a spanning tree, there is a unique orientation per node that makes the corresponding node the root of the tree.

since they are ensured to be in separate connected components. Then the number of all spanning forests with the root set R reads:

$$|\mathcal{F}_R| = \det \mathsf{L}_{-R}.$$

2.1.2 Random Walks on Graphs

A random walk W on a graph is a random process that starts from a node s, and at every step it goes to another node with a probability $\frac{w(i,j)}{d_i}$. As a result, the probability of having a fixed walk ω in this process equals to:

$$\mathbb{P}(W = \omega | W_0 = s) = \prod_{e_k=(i,j)\in\omega} \frac{w(i,j)}{d_i}, \tag{2.2}$$

Such random walks naturally are discrete Markov chain.

Definition 2.1.11 (Discrete Markov Chain)**.** A discrete Markov chain $M(i)$ is a random process over a discrete state space $\mathcal{V}$ that satisfies the Markovian property for all $k \in \mathbb{N}, i, j_k \in \mathcal{V}$:

$$\mathbb{P}(M(k) = i | M(k-1) = j_{k-1}, \ldots, M(1) = j_1) = \mathbb{P}(M(k) = i | M(k-1) = j_{k-1}).$$

Markov chains are often described by their transition probabilities *i.e.* the probability of switching from one state to another. A transition matrix $\mathsf{P} = [\mathbb{P}(M(k) = i | M(k-1) = j)]_{i,j\in\mathcal{V}}$ is the matrix that contains these probabilities. We associate a Markov chain with a graph by setting the vertices as the state space and the edges as the transitions with non-zero probabilities. One can generate random walks distributed according to Eq. (2.2) by simulating the Markov chain with the following transition probabilities:

$$\mathbb{P}(M(k) = j | M(k-1) = i) = \frac{w(i,j)}{d_i} \tag{2.3}$$

To better understand Markov chains, we revisit some of their useful properties. We list them in the following with short descriptions:

- **Periodicity:** The period of a state x is defined as:

$$p(x) = \texttt{gcd}(n \in \mathbb{N}^+ : \mathbb{P}^n(x, x) > 0)$$

  where gcd returns the greatest common integer divisor of a given set. A state x in a Markov chain is called periodic if $p(x) > 1$. Otherwise, it is called aperiodic. If all states are aperiodic, the Markov chain is called aperiodic.

- **Transient/Recurrent States:** If a state i is recurrent, then the random walks starting i almost surely return to i while the number of steps goes to infinity. Otherwise, it is transient.

- **Ergodicity:** A state is ergodic if it is aperiodic and recurrent. If all states are ergodic, then the Markov chain is called ergodic.

In ergodic Markov chains, the probability distribution of $M(k)$ over all states converges to a distribution as k goes to infinity [Chu67]. This distribution is also called the stationary distribution and it verifies:

$$\pi^\top \mathsf{P} = \pi^\top \text{ with } \sum_{i \in \mathcal{V}} \pi_i = 1.$$

In other words, the stationary distribution of an ergodic Markov chain is the right eigenvector of its transition matrix. This result becomes intuitive when we look at the algebraic view of random walks. Let us consider a random walk starting from node i and write the vector $\delta_i \in \mathbb{R}^{|\mathcal{V}|}$ which is the Kronecker delta. After taking k steps, the probability distribution of the current state of the random walk reads:

$$\mathbf{r}_k = \delta_i^\top \mathsf{P}^k.$$

The stationary distribution, in fact, is the limit of $\mathbf{r}_k$ as k goes to infinity. By the Perron-Frobenius theorem, this limit converges to the right eigenvector of P.

2.1.2.1. Loop-Erased Random Walks

One can find many kinds of random walks by slightly deviating from the original definition in Eq. (2.2). However, all of them do not necessarily come with the Markovian property. Due to this, their distribution often is not mathematically tractable whenever we diverge from the original random walks. Thus any theoretical

(a) Before loop-erasure (b) After loop-erasure

Fig. 2.6.: Loop erasure procedure. We take a random walk starting from the node at the center (in red), apply the loop erasure defined in Definition 2.1.12

analysis usually cannot go further. Loop-erased random walks (Loop-erased Random Walks (LERW)s) are fascinating exceptions in this manner. They were first proposed by [Law79] in polymer physics. Since then, they have been "arguably the most tractable model among non-Gaussian models in statistical physics [Koz07]". More related to this thesis, they play a significant role in the sampling of Uniform Spanning Tree (UST) *i.e.* uniformly sampled spanning trees on a graph. [BP93] shows that a path from node i to j in a UST has the same distribution as LERWs. In his seminal paper [Wil96], Wilson leveraged this fact to propose the current most efficient exact algorithm to generate uniform spanning trees. [Mar00] later gave the expected complexity of this algorithm by using the law of LERWs and graph matrices such as W or L. All of these connections lie at the very heart of this thesis. Therefore, we conclude our tour on graph theory by introducing some of the essential results related to LERWs.

Let us start with the formal definition of LERWs:

Definition 2.1.12 (LERW)**.** Given a random walk $W = (e_1, e_2, \dots, e_l)$, let us denote the nodes visited in W by $V = (v_1, v_2, \dots, v_{l+1})$ such that $\forall e_i \in W,\ e_i = (v_i, v_{i+1})$. Then we define a new subset of indices $I = \{i_1, i_2, \dots, i_k\}$'s of $1, \dots, l+1$ that satisfies:

$$\begin{aligned} i_1 &= 1 \\ i_{j+1} &= \max\{i : v_i = v_{i_j}\} + 1. \end{aligned} \tag{2.4}$$

From the subset I, we form a loop-erased random walk $LE(W) \subseteq \mathcal{E}$ as follows:

$$LE(W) = \{(v_{i_1}, v_{i_2}), (v_{i_2}, v_{i_3}), \dots, (v_{i_{k-1}}, v_{i_k})\}.$$

As illustrated in 2.6, LERWs are random walks in which the cycles are deleted in the order that they show up (chronological order). In comparison to usual random walks, this description can seem complicated. However, its probability law surprisingly remains tractable in terms of the graph matrices.

Theorem 2.1.4 (Law of LERWs [Mar00]). *A loop-erased random walk $LE(W)$ on $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ that is stopped at the boundary $\Delta \subset \mathcal{V}$ has the following probability distribution:*

$$\mathbb{P}(LE(W) = \gamma) = \frac{\det \mathsf{L}_{-\Delta \cup s(\gamma)}}{\det \mathsf{L}_{-\Delta}} \prod_{(i,j)\in\gamma} w(i,j)$$

where γ is a fixed path and $s(\gamma)$ denotes the nodes visited in γ.

Proof. See Lemma 1 in [Mar00]. □

This beautiful result ties not only diverse concepts such as probability theory and graph-related algebra but also becomes handy in the development and analysis of some of well-known graph algorithms in computer science. [Mar00] uses this result as a building block for the analysis of Wilson's algorithm as Wilson's algorithm repeatedly uses LERWs interrupted at some boundary. We will in Section 2.4.1 further detail this analysis along with the formal description of Wilson's algorithm.

2.2 Determinantal Point Processes

DPPs appear in most diverse phenomena. One of the earliest occurrences is in quantum physics; The particles called fermions obey Pauli's exclusion principle, which states that two fermions can not occupy the same quantum state. In [Mac75], Macchi describes this behaviour of fermions by a DPP. Another early example is found in random matrix theory *i.e.* a field of mathematics that examines the properties of matrices whose entries are random variables [ER05]. DPPs arise in the spectral analysis of the random matrices whose each entry is an independent random variable with complex normal distribution. The random eigenvalues of such matrices are distributed according to a DPP [Joh05]. Another interesting example pops up in number theory. Consider a random sequence $S = \{X_1, \ldots, X_N\}$ of N numbers drawn between 0 and 9 uniformly. The locations whenever there is a descent in this sequence *i.e.* $\{i \in S | X_{i+1} \leq X_i\}$, is surprisingly a DPP. [KT+12] lists many interesting examples. A notable instance is associated with uniform spanning trees on graphs. The edges in a UST follow a DPP law which leads to tractable probabilities by using algebraic graph theory. We will detail this example and extend it for weighted trees in Section 2.4. Before doing so, we cover the necessary formal definitions and properties of DPPs.

2.2.1 Basics of DPPs

A point process $\mathcal{X}$ over a set Ω is a random subset of Ω. DPPs are point processes with a particular analytical form for their probability distribution.

Definition 2.2.1 (DPP)**.** A point process $\mathcal{X}$ over Ω is called a determinantal point process if its inclusion probabilities *i.e.* the probability of any subset in $\mathcal{X}$, verify for every fixed subset $S \subseteq \Omega$:

$$\mathbb{P}(S \subseteq \mathcal{X}) = \det \mathsf{K}_S$$

where K_S denotes the submatrix of the semi-definite positive matrix $\mathsf{K} \in \mathbb{R}^{|\Omega| \times |\Omega|}$ restricted to the rows and columns indexed by S. The matrix K is also called marginal kernel and its spectrum verifies $0 \preccurlyeq \mathsf{K} \preccurlyeq \mathsf{I}$.

This compact description of the inclusion probabilities yield many tractable properties. For example, if we take two objects $i, j \subset \Omega$ and take a look on their inclusion probability, we see:

$$\begin{aligned} \mathbb{P}(i, j \in \mathcal{X}) = \det \mathsf{K}_{\{i,j\}} &= \begin{vmatrix} \mathsf{K}_{i,i} & \mathsf{K}_{i,j} \\ \mathsf{K}_{j,i} & \mathsf{K}_{j,j} \end{vmatrix} \\ &= \mathsf{K}_{i,i}\mathsf{K}_{j,j} - \mathsf{K}_{i,j}\mathsf{K}_{j,i} \\ &= \mathbb{P}(i \in \mathcal{X})\mathbb{P}(j \in \mathcal{X}) - \mathsf{K}_{i,j}\mathsf{K}_{j,i} \end{aligned} \tag{2.5}$$

For simplicity, we assume K is a symmetric matrix. Then, one can easily see the *negative association*:

$$\begin{aligned} \mathbb{P}(i, j \in \mathcal{X}) &\leq \mathbb{P}(i \in \mathcal{X})\mathbb{P}(j \in \mathcal{X}) \\ \mathbb{P}(i \in \mathcal{X} | j \in \mathcal{X}) &\leq \mathbb{P}(i \in \mathcal{X}). \end{aligned}$$

Notice that having $j \in \mathcal{X}$ decreases the probability of also having $i \in \mathcal{X}$. One can see this as a repulsive process where j *repels* i.

A bit technical but useful property follows the definition:

Lemma 2.2.1 (Restriction)**.** *Given a DPP $\mathcal{X}$ on a ground set Ω with a marginal kernel K and a fixed subset $\alpha \subseteq \Omega$, $\mathcal{X} \cap \alpha$ is also drawn from a DPP with marginal kernel K_α.*

In other words, a restriction of a DPP is also a DPP.

The choice of the kernel matrix K can be various. The similarity matrices, in which each entry depicts the mutual similarity between the objects, are popular choices. In this case, as the similarity $\mathsf{K}_{i,j}$ between two objects i and j grows, we end up with lower joint probabilities of i and j, which endorses the repulsion of two similar objects, in other words, diversity. However, notice that the choice of K such that all of its principal minors are probabilities is not obvious. Therefore, L-ensembles, a more natural subclass of DPPs, are often preferred:

Definition 2.2.2 (L-ensemble)**.** An L-ensemble $\mathcal{X}$ is a point process associated with a matrix $\mathcal{L} \in \mathbb{R}^{n \times n}$ which satisfies:

$$\forall S \subseteq \Omega, \quad \mathbb{P}(\mathcal{X} = S) \propto \det \mathcal{L}_S,$$

where $\mathcal{L}$ is a positive semi-definite matrix.

An L-ensemble is a DPP [1] described by its atomic probabilities, the probability of sampling a fixed element or subset, which are easier to grasp for those outside this field. In addition, proportionality gives a higher degree of freedom by dropping some of the constraints of a proper matrix to design the DPP. Moreover, one can recover the corresponding K from a given $\mathcal{L}$ (See Thm. 2.2 in [KT+12]):

$$\mathsf{K} = \mathcal{L}(\mathcal{L} + \mathsf{I})^{-1} = \mathsf{I} - (\mathsf{I} + \mathcal{L})^{-1}. \tag{2.6}$$

An immediate result of this identity is that K and $\mathcal{L}$ are spanned by the same eigenvectors. Let us denote the eigen-pairs of K by $(\xi_i, \mathbf{x}_i)_{i \in \Omega}$ and the eigenvalues of $\mathcal{L}$ by $\ell_1 \leq \ell_2 \leq \dots \ell_n$. Notice that each $\mathbf{x}_i$ is also the eigenvector of $\mathcal{L}$ with the eigenvalue:

$$\forall i \in \Omega, \quad \ell_i = \frac{\xi_i}{1 - \xi_i}.$$

The sampling algorithm in [HKPV06; KT+12] uses this link to sample a DPP given $\mathcal{L}$. This algorithm is exact but requires the eigendecomposition of $\mathcal{L}$ as input. Therefore, one needs to diagonalize a $n \times n$ matrix as a preprocessing, which yields a time complexity $\mathcal{O}(n^3)$. This cost is avoided for large n (*i.e.* $n = 10^5$) by low-rank approximations or other approximate sampling algorithms providing a significant speed-up [DCV19].

The tractable properties of DPPs are not limited to the inclusion and atomic probabilities. One can calculate the expected number of items in $\mathcal{X}$:

[1] The contrary is not necessarily true. See [DM21]

Proposition 2.2.2. *The expectation and variance of $|\mathcal{X}|$ is as follows:*

$$\mathbb{E}[|\mathcal{X}|] = \mathrm{tr}(\mathsf{K}), \mathrm{Var}(|\mathcal{X}|) = \mathrm{tr}(\mathsf{K} - \mathsf{K}^2) \tag{2.7}$$

Proof. The proof of expectation is one line:

$$\mathbb{E}[|\mathcal{X}|] = \mathbb{E}\left[\sum_{i\in\Omega} \mathbb{I}(i \in \mathcal{X})\right] = \sum_{i\in\Omega} \mathbb{P}(i \in \mathcal{X}) = \sum_{i\in\Omega} \mathsf{K}_{i,i} = \mathrm{tr}(\mathsf{K}). \tag{2.8}$$

One can derive the variance as follows:

$$\begin{aligned}
\mathrm{Var}(|\mathcal{X}|) &= \mathbb{E}[|\mathcal{X}|^2] - \mathbb{E}[|\mathcal{X}|]^2 = \mathbb{E}[|\mathcal{X}|^2] - \left(\sum_i \mathsf{K}_{i,i}\right)^2 \\
&= \mathbb{E}\left[\left(\sum_{i\in\Omega} \mathbb{I}(i \in \mathcal{X})\right)^2\right] - \sum_{i\in\Omega} \mathsf{K}_{i,i}^2 - \sum_{i\in\Omega}\sum_{j\in\Omega\setminus i} \mathsf{K}_{i,i}\mathsf{K}_{j,j} \\
&= \mathbb{E}\left[\sum_{i\in\Omega} \mathbb{I}(i \in \mathcal{X}) + \sum_{i\in\Omega}\sum_{j\in\Omega\setminus i} \mathbb{I}(i,j \in \mathcal{X})\right] - \sum_{i\in\Omega} \mathsf{K}_{i,i}^2 - \sum_{i\in\Omega}\sum_{j\in\Omega\setminus i} \mathsf{K}_{i,i}\mathsf{K}_{j,j} \\
&= \sum_{i\in\Omega} \mathbb{P}(i \in \mathcal{X}) + \sum_{i\in\Omega}\sum_{j\in\Omega\setminus i} \mathbb{P}(i,j \in \mathcal{X}) - \sum_{i\in\Omega} \mathsf{K}_{i,i}^2 - \sum_{i\in\Omega}\sum_{j\in\Omega\setminus i} \mathsf{K}_{i,i}\mathsf{K}_{j,j} \\
&= \sum_{i\in\Omega} \mathsf{K}_{i,i} + \sum_{i\in\Omega}\sum_{j\in\Omega\setminus i} (\mathsf{K}_{i,i}\mathsf{K}_{j,j} - \mathsf{K}_{i,j}^2) - \sum_{i\in\Omega} \mathsf{K}_{i,i}^2 - \sum_{i\in\Omega}\sum_{j\in\Omega\setminus i} \mathsf{K}_{i,i}\mathsf{K}_{j,j} \\
&= \left(\sum_{i\in\Omega} \mathsf{K}_{i,i}\right) - \left(\sum_{i\in\Omega}\sum_{j\in\Omega\setminus i} \mathsf{K}_{i,j}^2\right) - \left(\sum_{i\in\Omega} \mathsf{K}_{i,i}^2\right) \\
&= \mathrm{tr}(\mathsf{K} - \mathsf{K}^2)
\end{aligned} \tag{2.9}$$

□

In terms of the eigenvalues of $\mathcal{L}$, one has:

$$\mathbb{E}\left[|\mathcal{X}|\right] = \sum_{i\in\Omega} \frac{\ell_i}{\ell_i + 1} \text{ with } \mathrm{Var}\left(|\mathcal{X}|\right) = \sum_{i\in\Omega} \frac{\ell_i}{(\ell_i + 1)^2}.$$

Other tractable properties such as conditional or marginal probabilities are also available for DPPs. We refer to [KT+12] for more details.

A fixed-sized DPP, also called k-DPP, is a determinantal point process conditioned over its size. Unlike DPPs, their probabilities are not interpretable sole determinants but ratios of determinants.

Definition 2.2.3 (Fixed-size DPPs)**.** A fixed-size DPP is a special point process in which the number of items is set to a fixed number k. The atomic probabilities for a fixed set S with $|S| = k$ read:

$$\mathbb{P}(\mathcal{X} = S) = \frac{\det \mathcal{L}_S}{\sum\limits_{\substack{S' \subseteq \Omega \\ |S'|=k}} \det \mathcal{L}_{S'}}.$$

The fixed size DPPs give a rise naturally in certain applications. For example, consider the cellular network problem introduced in [BLMV17]. The goal is to place antennas over a field so that every location receives the most reception that it can get. A natural solution to this problem is to model and sample antenna locations from a DPP. Imagine that we also have a strict constraint on the number of antennas that we can place due to the budget or environmental issues. In this case, k-DPPs come into play as we want to place exactly k antennas in the most diverse locations.

We conclude this summary by examining a special type of DPPs which are intersection of DPPs and fixed-size DPPs, called projection DPPs:

Definition 2.2.4 (Projection DPP)**.** A projection DPP is a DPP with marginal kernel K which is a projection matrix *i.e.* $\mathsf{K}^2 = \mathsf{K}$, hence the name.

Corollary 2.2.3. *Let $\mathcal{X}$ be a projection DPP. By definition, its kernel is a projection matrix, thus it only has eigenvalues that are either 1 or 0. Let $k < |\Omega|$ be the number of the non-zero eigenvalues. By using the Eqs. (2.8) and (2.9), one obtains that $\mathbb{E}[|\mathcal{X}|] = k$ with zero variance. In other words, $\mathcal{X}$ is a fixed-size DPP.*

It is also possible to tie the projection DPPs with L-ensembles:

Lemma 2.2.4 (Low rank L-ensemble [BAT19])**.** *Let $\mathcal{L}$ be a symmetric matrix with $\operatorname{rank} \mathcal{L} = k$. Let us diagonalize it as $\mathcal{L} = \mathsf{U}\mathsf{D}\mathsf{U}^\top$ where $\mathsf{U} \in \mathbb{R}^{n \times k}$ contains the non-trivial k eigenvectors and $\mathsf{D} \in \mathbb{R}^{k \times k}$ is the diagonal matrix with the non-zero eigenvalues in its diagonal entries. A fixed-size DPP with the size of k and the L-ensemble matrix $\mathcal{L}$ is equivalent to a projection DPP with the marginal kernel $\mathsf{U}\mathsf{U}^\top$.*

Proof. See result 1 in [BAT19]. □

The uniform spanning tree process on a graph is one instance in which projection DPPs show up. In a nutshell, given a graph with n nodes, one considers the random subset of edges that forms a uniform spanning tree. As aforementioned, this random process is precisely described by a DPP. Moreover, its kernel is a projection matrix with $n-1$ non-zero eigenvalues. In addition, a graph-theoretic verification for $k = n-1$ is that the number of edges needed to generate a tree that reaches all n vertices of a graph without creating a cycle is exactly $n-1$. These connections are some of the support pillars of this thesis. In Section 2.4, we detail these connections with formal descriptions.

2.3 Randomized Linear Algebra

In the last two decades, the volume of numerical data has exploded, and the need for efficient matrix algorithms has become more and more prominent. However, the classical and frequently used tools of linear algebra might not provide the required efficiency for the big data. For example, matrix inversion is essential in solving many machine learning problems. Inverting an $n \times n$ square matrix requires $\mathcal{O}(n^3)$ operations via the Gauss-Jordan elimination method. For sparse matrices, this cost is often avoided via Cholesky decomposition, which remains in $\mathcal{O}(n^3)$ in the worst case. When it comes to dealing with billions of data points *i.e.* $n = 10^9$, this computation might take days, thus be impractical. Along with the computational issues, there are also hardware limitations. For example, as the data grows exponentially, it becomes less and less feasible to fit the whole data to the RAMs (random access memories) of today's computers. Due to this, the computer needs to access the main memory multiple times, yielding runtime inefficiencies.

Randomized linear algebra (RLA) addresses both of these concerns by introducing *randomness* to linear algebra (LA). RLA proposes to imitate some of the main operations in LA with a randomized scheme, also called *sketch-and-solve* in [DM16]; first, we sample a smaller portion of the data/the corresponding matrix; then, we apply the expensive operation on this portion which can be handled cheaply. We obtain by this way an *estimate* of the result.

This scheme has been applied to some of the main problems in LA, including solving dense/sparse linear systems, least-squares (also regularized), eigen or singular value decomposition [MT20]. They have been able to outperform direct computations in solving some of these problems *e.g.* least-squares [DMMS11] or low-rank approximation problems [DKM06]. In the scope of this thesis, we take look on a more

recent RLA algorithm [DM21] for tackling regularized least-squares problems. This algorithm propose a DPP for creating sketches such that the corresponding estimates of the result are an unbiased estimate *i.e.* the expectation of the estimates equals the exact result. This algorithm can also be seen as a basis for the methods that will be introduced in Chapter 3. Therefore, we will revisit the least-squares problem and the DPP-based RLA algorithm in the following section.

2.3.1 Linear Least-Squares Problem

In linear least-squares problems, given the observations $\mathbf{b} \in \mathbb{R}^n$ and the predictors $\mathsf{A} \in \mathbb{R}^{n \times p}$, we seek the solution of:

$$\mathbf{x}^\star = \underset{\mathbf{x} \in \mathbb{R}^p}{\operatorname{argmin}} ||\mathsf{A}\mathbf{x} - \mathbf{b}||_2. \tag{2.10}$$

Depending on n and p, the solution takes a different form. In our case, we restrict ourselves to the overdetermined case *i.e.* $n > p$ where we usually have the solution which has a closed-form solution as follows:

$$\mathbf{x}^\star = (\mathsf{A}^\top \mathsf{A})^{-1} \mathsf{A}^\top \mathbf{b} = \mathsf{A}^\dagger \mathbf{b},$$

where $\mathsf{A}^\dagger$ denotes the Moore-Penrose inverse of A and we assume that A is a tall and full-rank matrix, thus $(\mathsf{A}^\top \mathsf{A})$ is invertible.

A well-known application of this formulation is linear regression [HTFF09]. In this case, each row of A contains a data point $\mathbf{a}_i^\top$ where $\mathbf{a}_i \in \mathbb{R}^p$ and $\mathbf{b}$ consists of an observation per data point. The main goal is to learn the linear relation between the data points $\mathbf{a}_i$'s and the observations b_i's. To do so, we fit a hyperplane (a line in case of $p = 1$) that is parameterized by $\mathbf{x}$ and minimizes $||\mathsf{A}\mathbf{x} - \mathbf{b}||_2$. Real life examples are limitless and each is not necessarily interesting. For solely entertainment purposes, in Example 2.3.1, we take a liberty of generating a fictional example.

Mission: Killing the Sun

According to the local news in 2016 [Hur], two men who were overwhelmed by the extremely high temperatures during summer in Adana/-Turkey started to take shots at the sun for the purpose of killing it. While the local police arrested them for the public shooting, the vital status of the sun is reported to be okay. The mayor invited the public on their social media accounts to keep their calm in the face of these sweltering temperatures and advised that shooting at the sun would not be a feasible solution.

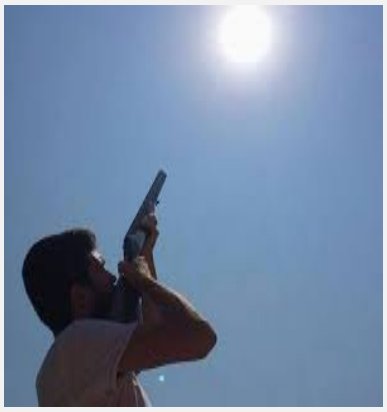


Fig. 2.7.: A man shooting at the sun

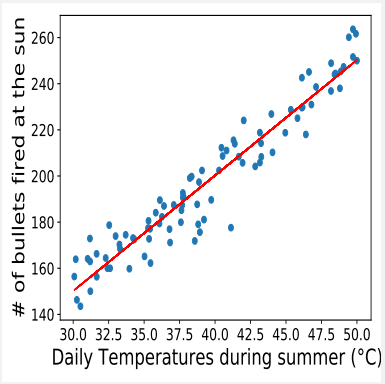


Fig. 2.8.: Line fitting between the bullets fired at the sun and temperature in Adana/Turkey

We imagine an amusing line-fitting problem between the temperatures (data points) in Adana during the summer, and the number of the bullets fired at the sun (observation). Our purpose here is to find the line that best explains the relation between the temperature and the number of bullets. Solving this problem under the least-squares formulation, one fits a line as shown in Fig 2.8.

In many real life applications, the solution of least-squares suffers from ill-posed problems *i.e.* A is not full rank, and poor generalization *i.e.* overfitting. In the former case $(A^\top A)$ becomes singular thus cannot be inverted. The latter is a long standing issue in many machine learning models. In case of Least Squares (LS), the minimizer $\mathbf{x}^\star$ might take a form which is extremely successful for minimizing $||A\mathbf{x} - \mathbf{b}||$ but completely useless to predict the observation for unseen data. This is because $\mathbf{x}^\star$ can take extreme values to minimize $||A\mathbf{x} - \mathbf{b}||$ which is not necessarily robust depending on A and $\mathbf{b}$. The regularization addresses both issues. The main idea is to add an

additional term to the optimization formula to *control* $\mathbf{x}^\star$ such that it is well-defined and bounded.

In case of L_2 regularization with a regularization matrix $\mathsf{R} \in \mathbb{R}^{p\times p}$, the problem becomes:

$$\mathbf{x}^\star = \underset{\mathbf{x}\in\mathbb{R}^p}{\operatorname{argmin}} ||\mathsf{A}\mathbf{x} - \mathbf{b}||_2 + \mathbf{x}^\top \mathsf{R}\mathbf{x} \tag{2.11}$$

and the solution still preserves a closed form as:

$$\mathbf{x}^\star = (\mathsf{A}^\top\mathsf{A} + \mathsf{R})^{-1}\mathsf{A}^\top\mathbf{b}, \tag{2.12}$$

As interpretable and straightforward as it is, this solution comes with computational issues as p grows. Inverting a $p \times p$ matrix takes $\mathcal{O}(p^3)$ operations via direct computations of the Gauss-Jordan elimination method. Some of the intermediate calculations (*e.g.* pivoting) in Gauss-Jordan can be avoided via Cholesky decomposition [Saa03], yet the number of elementary operations remains in the cubic order of p. As the exact methods become inefficient whenever p is very large, state-of-the-art methods are the approximate ones in which we estimate the result much more cheaply with a small approximation error $\epsilon > 0$. Depending on the algorithmic properties and the application, there are several schools; such as centralized/distributed [MLWFM16] or deterministic/randomized methods [MT20] etc. We revisit in the following section some of these methods.

2.3.2 Approximate Methods for Solving LS Problem

Aligned with the main themes of this thesis, we will examine approximate methods in two groups; deterministic and randomized (or stochastic). Within this taxonomy, we classify the algorithms according to whether they contain a randomized element or not. In our tour over the most used algorithms, we reproduce some of them and compare their strength and weaknesses.

2.3.3 Deterministic Methods

Iterative solvers are the most used algorithms that fall under the deterministic methods for solving linear systems. In a nutshell, they use an iteration scheme that approximates the exact solution. They usually start with an arbitrary initial solution[2], and per iteration, the solution is updated such that it gets closer and closer to the

[2]Except for the initialization, they do not necessarily include any sort of randomization.

exact solution. The iterative algorithms are fast to solve linear systems in form of $\mathsf{A}\mathbf{x} = \mathbf{b}$ by using only matrix multiplications with A. In this way, they benefit from the sparsity if A is a sparse matrix *i.e.* number of non-zero entries is in the same order of magnitude as p. In addition, their approximation error $\epsilon = ||\mathbf{x}^\star - \hat{\mathbf{x}}||$ is tractable in terms of eigen/singular values of A. Thanks to this fact, one can easily derive theoretical guarantees for convergence. In the following, we will shortly introduce the main ideas of these methods and reproduce some of the well-known algorithms and results.

Let us define the square matrix $\mathsf{B} \coloneqq \mathsf{A}^\top\mathsf{A} + \mathsf{R}$ and $\mathbf{y} \coloneqq \mathsf{A}^\top\mathbf{b} \in \mathbb{R}^p$ and rewrite the solution in 2.12 as:

$$\mathbf{x}^\star = \mathsf{B}^{-1}\mathbf{y}$$

Iterative solvers vary in computing $\mathbf{x}^\star$ depending on how they formulate the update $f : \mathbb{R}^p \to \mathbb{R}^p$ within the following iteration scheme:

$$\mathbf{x}_{k+1} \coloneqq f(\mathbf{x}_k), \quad k = 1, 2, \dots \tag{2.13}$$

In our case, it is possible to draw an iteration for a given splitting $\mathsf{B} = \mathsf{M} - \mathsf{N}$:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathsf{M}^{-1}\mathsf{N}\mathbf{x}_k + \mathsf{M}^{-1}\mathbf{y}, \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathsf{M}^{-1}(\mathbf{y} - \mathsf{B}\mathbf{x}_k). \end{aligned} \tag{2.14}$$

The basic iterative methods, *e.g.* Jacobi, Gauss-Seidel, successive over-relaxation (SOR) or Richardson, split B such that M can be inverted much easier than B. For example, the Jacobi iteration takes the following form:

$$\mathbf{x}_{k+1} \coloneqq \mathsf{D}^{-1}(\mathsf{L} + \mathsf{U})\mathbf{x}_k + \mathsf{D}^{-1}\mathbf{y},$$

where we split $\mathsf{B} = \mathsf{D} - \mathsf{L} - \mathsf{U}$, D is a diagonal matrix which contains the diagonal entries of B and L and U contains respectively the lower and upper triangular parts of B. As D is easy to invert, one only needs to access the matrix B via matrix-vector products per iteration. Other basic iterative methods slightly differs in how they decompose the matrix B. As a result, they vary in the computational load per iteration and convergences properties. However, one can still derive general results on the convergence:

Theorem 2.3.1 (Theorem 4.1 in [Saa03]). *Define the iteration matrix* $\mathsf{G} = \mathsf{M}^{-1}\mathsf{N}$ *for any splitting in form of* $\mathsf{B} = \mathsf{M} - \mathsf{N}$. *Assume* G *is non-singular, thus there exists a unique solution* $\mathbf{x}^\star$. *Then the iteration in Eq.* (2.14) *converges to the solution*

$\mathbf{x}^\star$ *if and only if the spectral radius of* G *i.e.* $\max_i |\lambda_i(\mathsf{G})|$ *where* $\lambda_i(\mathsf{G})$ *is the* i*-th eigenvalues of* G*, denoted by* $\rho(\mathsf{G})$ *is strictly less than 1.*

Proof. We refer the reader to [Saa03] for the detailed proof. □

This result indicates that not all splittings yield convergent iterations to the solution. In particular, one needs to know the spectral radius of the corresponding iteration matrix to guarantee the convergence. As $\rho(\mathsf{G})$ is not easily accessible in practice, [Saa03] suggests using $||\mathsf{G}||_2 > \rho(\mathsf{G})$ as an upper bound.

In fact, this result extends on the convergence rate *i.e.* a quantity to measure how fast the iterations converges to the solution. Formally, we define the convergence rate as follows:

$$\mu := \lim_{k\to\infty} \frac{||\mathbf{x}_{k+1} - \mathbf{x}^\star||}{||\mathbf{x}_k - \mathbf{x}^\star||}.$$

For all the basic iterative methods above, this limit equals to $\rho(\mathsf{G})$ [Saa03].

The basic iterative solvers are simple to implement and analyze. However, they are not applicable whenever the matrix B does not satisfy the constraints that are necessary for convergence *e.g.* diagonal dominance for Jacobi iterations. More robust methods with faster convergence guarantees are the Krylov subspace methods. These methods have a vast literature and practice [Saa03]. In the following, we summarize the main theory and reproduce one of the most used algorithms, called the conjugate gradient method.

Definition 2.3.1 (Krylov Subspace)**.** A Krylov subspace $\mathcal{K}_k(\mathsf{B}, \mathbf{r})$ based on the matrix B and the vector $\mathbf{r}$ is a subspace of $\mathbb{R}^n$ with a size $m \leq n$ that is defined as:

$$\mathcal{K}_k(\mathsf{B}, \mathbf{r}) = \text{span}(\mathbf{r}, \mathsf{B}\mathbf{r}, \mathsf{B}^2\mathbf{r}, \ldots, \mathsf{B}^{k-1}\mathbf{r})$$

Lemma 2.3.2. *The solution* $\mathbf{x}^\star = \mathsf{B}^{-1}\mathbf{b}$ *belongs to* $\mathcal{K}_n(\mathsf{B}, \mathbf{y})$.

Proof. The minimal polynomial of B, denoted by $q(\mathsf{B})$, is the unique polynomial with the minimal degree such that $q(\mathsf{B}) = 0$. Given the eigenvalues of B $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$, one can rewrite this polynomial as:

$$q(t) = \prod_{i=1}^{n} (t - \lambda_n).$$

Rewriting $q(t)$, one has:

$$q(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \dots \alpha_n t^n$$

with the constant term $\alpha_0 = \prod_{i=1}^{n} \lambda_i = \det \mathsf{B}$. When B is non-singular, α_0 is ensured to be non-zero. Thus, one rewrite $q(\mathsf{A}) = 0$ as:

$$\begin{aligned} -\alpha_0 \mathsf{I} &= \alpha_1 \mathsf{B} + \alpha_2 \mathsf{B}^2 + \dots \mathsf{B}^n \\ -\alpha_0 \mathsf{I} &= \mathsf{B}(\alpha_1 \mathsf{I} + \alpha_2 \mathsf{B} + \dots \mathsf{B}^{n-1}) \\ \mathsf{B}^{-1} &= -\frac{1}{\alpha_0}(\alpha_1 \mathsf{I} + \alpha_2 \mathsf{B} + \dots \mathsf{B}^{n-1}) \end{aligned}$$

Then, multiplying both sides with the vector $\mathbf{b}$ finishes the proof:

$$\mathsf{B}^{-1}\mathbf{b} = -\frac{1}{\alpha_0}(\alpha_1 \mathsf{I} + \alpha_2 \mathsf{B} + \dots \mathsf{B}^{n-1})\mathbf{y} = \mathbf{x}^\star \in \mathcal{K}_n(\mathsf{B}, \mathbf{y})$$

□

There are several algorithms proposed that are based on Krylov subspaces. The main idea is to approximate $\mathbf{x}^\star$ iteratively by following a set of candidate solutions $\mathbf{x}_i$'s that are also belong to $\mathcal{K}_k(\mathsf{B}, \mathbf{b})$. The best-known ones are Arnoldi's method, generalized minimized residual method and conjugate gradient method Let us complete this section by reproducing conjugate gradient algorithm. A generic gradient descent algorithm minimizes the following loss function

$$F(\mathbf{x}) \coloneqq \frac{1}{2}\mathbf{x}^\top \mathsf{B}\mathbf{x} - \mathbf{x}^\top \mathbf{y}.$$

Notice that $\mathbf{x}^\star$ minimizes $F(\mathbf{x})$. Given an arbitrary initial solution $\mathbf{x}_0$, an intuitive way to approach $\mathbf{x}^\star$ is to take a step from $\mathbf{x}_0$ in the opposite direction of the gradient of $F(\mathbf{x})$. An iteration that implements this idea is as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla F(\mathbf{x}_k) \tag{2.15}$$

where the gradient $\nabla F(\mathbf{x})$ reads $\mathsf{B}\mathbf{x} - \mathbf{y}$ and $\alpha \in \mathbb{R}$ adjusts the size of the step. On top of this, the conjugate gradient method suggests taking orthogonal directions with optimal step sizes such that there will be no need to take another step in the direction of previous steps. As a result, the updates form a Krylov subspace based on $\mathcal{K}_k(\mathsf{B}, \mathbf{b})$. We summarize the algorithm in Alg. 1. The convergence rate of this algorithm is given as $1 - \frac{2}{\sqrt{\kappa}}$ *i.e.* at every iteration k, one has $||\mathbf{x}_{k+1} - \mathbf{x}^\star||_2 \leq \left(1 - \frac{2}{\sqrt{\kappa}}\right)^k ||\mathbf{x}_1 - \mathbf{x}^\star||_2$ where $\kappa = \frac{\lambda_{max}(\mathsf{B})}{\lambda_{min}(\mathsf{B}}$ is the condition number.

Algorithm 1 `Conjugate Gradient Algorithm`

Inputs:
$\mathsf{B} \in \mathbb{R}^{n \times n}, \mathbf{b} \in \mathbb{R}^n$ *# Input matrix and vector*
$\mathbf{x}_1 \in \mathbb{R}^n$ *# Initial point*
$K \in \mathbb{N}^+$, *# Maximum number of iterations*
$\epsilon \in \mathbb{R}^+$ *# Tolerance*
Initialize:
$\mathbf{r}_0 \leftarrow \mathbf{b} - \mathsf{B}\mathbf{x}_1$ *# Initialize the residual*
$\mathbf{p}_0 \leftarrow \mathbf{r}_0$
for $k \leftarrow 1$ to K **do**
  $\alpha_k \leftarrow \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{r}_k^\top \mathsf{B}\mathbf{r}_k}$ *# Calculate the step size*
  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$ *# Update the iterate*
  $\mathbf{r}_{k+1} \leftarrow \mathbf{r}_k - \alpha_k \mathsf{B}\mathbf{p}_k$ *# Update the residual*
  if $||\mathbf{r}_{k+1}||_2 \leq \epsilon$ **then** *# If the residual is less than the tolerance, stop the loop*
    Break
  end if
  $\beta_k \leftarrow \frac{\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{r}_k^\top \mathbf{r}_k}$
  $\mathbf{p}_{k+1} \leftarrow \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
end for
`return` $\mathbf{x}_{k+1}$

Iterative algorithms cover a significant part of approximate methods for solving least-squares problems. They provide fast convergence with a small approximation error, while the main computational cost is due to the matrix-vector products with B. Thus they scale linearly with the number of non-zero entries in B. Moreover, it is possible to improve their performance via preconditioning [Saa03]. On the other hand, we note that, for using CG, B is restricted to the real, symmetric and positive semi-definite matrices. In addition, there are some cases where the product with B is also not available. For example, when B is too big, its vector products might be costly to evaluate.

Another branch of approximate methods, so-called randomized methods, is better adapted for dealing with these issues by leveraging probability theory and Monte Carlo simulations. In the following section, we revisit some of the relevant ones with this thesis.

2.3.4 Randomized Algorithms

The earliest randomized algorithms to solve linear systems in the form of $\mathsf{B}\mathbf{x} = \mathbf{y}$ are attributed to Ulam and von Neumann and later developed by [FL50]. These algorithms are based on random walks on a Markov chain whose initial distribution

and transition matrix are closely related to B and y, respectively. However, these algorithms require strict constraints on the spectrum of B to guarantee convergence to the desired solution. In addition, how to design the Markov chain to minimize the approximation error is still an open question. We refer to [Ökt05] for more details.

A set of algorithms given by the RLA literature addresses a broader class of matrices in the setup of the least square problem. Generically, these algorithms use the sketch-and-solve scheme to give an estimate $\tilde{\mathbf{x}}$ that is an (ϵ, δ)-approximation, *i.e.*:

$$||\mathsf{A}\tilde{\mathbf{x}} - \mathbf{b}||_2 \leq (1+\epsilon)||\mathsf{A}\mathbf{x}^\star - \mathbf{b}||_2 \text{ with probability } 1-\delta, \tag{2.16}$$

A generic description of such algorithms given by [DM16] is as follows:

Consider the ordinary least square problem in Eq 2.10 (without regularization) where the solution is the pseudo-inverse $\mathsf{A}^\dagger\mathbf{b}$. Let us define a random variable $J \in \{1, 2, \ldots, n\}$ over the row indices of A with a probability mass function $p : \{1, 2, \ldots, n\} \to (0, 1)$ that verifies $\sum_{i=1}^{n} p(i) = 1$. Then, a meta-algorithm to estimate $\mathbf{x}^\star$ takes the following form:

- Get k samples $j_1, j_2, \ldots, j_k$ of J according to p with replacement,
- Form the matrix $\tilde{\mathsf{A}}$ such that $\forall i \in \{1, \ldots, k\}, \tilde{\mathsf{A}}_{i|:} = \sqrt{\frac{1}{kp(j_i)}}\mathsf{A}_{j_i|:}$ and the vector $\tilde{\mathbf{b}}$ such that $\forall i \in \{1, \ldots, k\}, \tilde{\mathbf{b}}_i = \sqrt{\frac{1}{kp(j_i)}}\mathbf{b}_{j_i}$
- Solve the following least-squares problem to get approximation of $\mathbf{x}^\star$:

$$\tilde{\mathbf{x}} = \underset{\mathbf{x}\in\mathbb{R}^p}{\operatorname{argmin}} ||\tilde{\mathsf{A}}\mathbf{x} - \tilde{\mathbf{b}}||_2 = \tilde{\mathsf{A}}^\dagger\tilde{\mathbf{b}}. \tag{2.17}$$

As typically $k \ll n$, computing $\tilde{\mathbf{x}}$ can be handled much more cheaply than the original problem. On the other hand, the performance of this algorithm strongly depends on the probability mass function. [DM21] analyzes this scheme with some of the conventional probability mass functions *e.g.* uniform or weighted by row norms. This analysis concludes that the algorithm described above produces $(\epsilon, 0.1)$-approximations for some $k \ll n$. This is to say that one can avoid the expensive inverse operation by sampling rows of A and b and operating on these rows which, in turn, yields a fairly small approximation error.

Closer to the main themes of this thesis, more recent RLA algorithms deploy determinantal point processes [DM21]. These methods give rise to the elegant connections between the determinantal point processes and linear algebra. As a result, they benefit from certain theoretical guarantees such as unbiasedness.

[DM21] proposes two new DPP-based estimators for solving the ordinary and regularized least-squares problem. The main idea in these estimators is to sample rows by a fixed size DPP such that the solution to the Eq. (2.17) yields an unbiased estimation of $\mathbf{x}^\star$. In case of the ordinary LS, the suggested DPP is a fixed size L-ensemble with size of p and $\mathcal{L} = \mathsf{A}\mathsf{A}^\top$. Theorem 2 in [DM21] states that the corresponding DPP $\mathcal{X}$ verifies:

$$\mathbb{E}[\mathsf{A}_{\mathcal{X}}^{-1}\mathbf{b}_{\mathcal{X}}] = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^p} ||\mathsf{A}\mathbf{x} - \mathbf{b}||_2 = \mathsf{A}^\dagger\mathbf{b},$$

where $\mathsf{A}_{\mathcal{X}}$ is the submatrix of A restricted to the rows by $\mathcal{X}$. The second estimator uses the same idea to solve the regularized least-squares. In this case, $\mathcal{X}$ takes form of a DPP (not fixed size) with $\mathcal{L} = \frac{1}{\lambda}\mathsf{A}\mathsf{A}^\top$ to sample row indices. Then, it verifies:

$$\mathbb{E}[\mathsf{A}_{\mathcal{X}}^{\dagger}\mathbf{b}_{\mathcal{X}}] = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^p} ||\mathsf{A}\mathbf{x} - \mathbf{b}||_2 + \lambda||\mathbf{x}||_2 = (\mathsf{A}^\top\mathsf{A} + \lambda\mathsf{I})^{-1}\mathsf{A}\mathbf{b}.$$

In the actual problem, λ is the hyper-parameter to adjust the balance between the confidence on the predictors/observations and the regularization. As λ increases, the solution will be more regularized and vice versa. Similarly, as λ increases, the expected size of $\mathcal{X}$ becomes smaller. This means that the estimate $\mathsf{A}_{\mathcal{X}}^{\dagger}\mathbf{b}_{\mathcal{X}}$ relies less on the predictor A and the observations $\mathbf{b}$. For further analysis, such as the exact error or algorithmic complexity, we refer the reader to [DM21].

These estimators give a simple and elegant solution to least-squares problems when the direct or other approximate approaches are expensive. Within the contributions of this thesis, we will present in Chapter 3 a cheap method of reducing the expected error (or variance) of these unbiased estimators. Surprisingly, this error reduction method connects diverse concepts such as gradient descent optimization and variance reduction Monte Carlo simulations. In addition, we will also present some estimators for graph-regularized least-squares problems. Surprisingly, these estimators can be reconsidered as the DPP-based estimators explained above.

2.4 Random Spanning Forests

Similar to spanning trees, combinatorialists have been studying spanning forests to extract information about the graph. The earliest results on the enumeration, and also randomization, over spanning trees/forests dates back to Kirchhoff's matrix tree theorem [Kir47]. Later on, these results emerged in physics with exciting links to certain random models in statistical mechanics. A significant instance (for this thesis)

is that uniform spanning trees/forests are particular cases of a celebrated model in statistical physics, called random-cluster model [Gri04] which unifies the percolation theory and Ising-Potts' model *i.e.* a statistical model for ferromagnetic materials. In parallel, probabilists have extended the fruitful links between USTs (also Uniform Spanning Forest (USF)) and algebraic graph theory highlighting the connections with random walks, harmonic functions and electrical networks [BLPS01]. Then computer scientists have come into play and developed efficient algorithms for sampling USTs [Wil96]. As Wilson's algorithm [BLPS01; Ald90] remains the most efficient exact algorithm (it exactly generates a spanning tree from the uniform distribution) in the past three decades, more recent algorithms trade the exactness with the time complexity and propose approximate sampling algorithms [KM09].

With all these rich theoretical properties, USFs (along with USTs) have become apt tools to analyze complex graphs. In this thesis as well, we aim to leverage these properties for randomized linear algebra. Therefore, we dedicate the rest of this section to introduce random spanning forests (a more generic form of uniform spanning forests) and bring their deep connections with the graph Laplacian into the light.

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, consider all rooted spanning forests $\mathcal{F}$ over $\mathcal{G}$. The cardinal of this set $|\mathcal{F}|$ may vary a lot. The exact expression reads $|\mathcal{F}| = \det(\mathsf{L} + \mathsf{I})$ by extending the matrix-tree theorem [CA02]. As a result, the number of spanning forests on a graph with n can grow up to n^{n+2} (in the case of a complete graph) depending on the structure. As this number is usually very large for general graphs, analyzing a graph by enumerating all spanning forests remains a complicated task. A down-to-earth alternative is by randomization, defining the distribution of forests by associating probabilities on every spanning forest in $\mathcal{F}$. Such distributions over forests have been thoroughly studied by [BLPS01] with the limits in case of infinite graphs *i.e.* $n \to \infty$. In this thesis, we only study finite graphs and focus on the following random objects studied in [ACGM18]:

Definition 2.4.1 (RSF)**.** A random (rooted) spanning forest is a random directed subgraph whose outcome space is $\mathcal{F}$ with the following parametric distribution:

$$P(\Phi_Q = \phi) = \frac{1}{Z_Q} \prod_{i \in \rho(\phi)} q_i \prod_{(i,j) \in \phi} w(i,j) \tag{2.18}$$

where $Q = \{q_1, \ldots, q_n\}$ is a set of real parameters that verify $\min_i q_i > 0$ and Z_Q is the normalization constant over all forests. In case of $q_1 = q_2 = \ldots = q_n = q$, our notation for the corresponding RSF is Φ_q.

A particular case occurs when some q_i's are set to 0 for a subset of the nodes $V \subset \mathcal{V}$ by leaving at least one node with a non-zero parameter. The roots of the forests generated by this configuration are always contained in the set $\mathcal{V} \setminus V$. Another interesting case is as $q_i \to \infty$ for $i \in V \subseteq \mathcal{V}$. In this case, V is ensured to be a subset of $\rho(\Phi_Q)$ while the distribution takes the following form:

$$P(\Phi_Q = \phi) \propto \prod_{i \in \rho(\phi) \setminus V} q_i \prod_{(i,j) \in \phi} w(i,j).$$

The RSF Φ_Q is a fascinating object with many interesting properties. As it is of central importance in this thesis, let us reproduce some of the significant results related to Φ_Q.

2.4.1 Wilson's algorithm

One of the biggest advantages of Φ_Q is the existence of a simple and efficient algorithm to sample it. This algorithm is due to an adaptation of Wilson's algorithm [ACGM18]. Given a root node r, Wilson's algorithm originally generates samples of the random spanning tree T with the following distribution:

$$\mathbb{P}(T = \tau) \propto \prod_{(i,j) \in \tau} w(i,j), \quad \tau \in \mathcal{T}_r. \tag{2.19}$$

where $\mathcal{T}_r$ is the set of all spanning trees rooted in r. This algorithm uses LERWs to generate the paths of a spanning tree as follows:

1. Initialize the boundary $\Delta = \{r\}$ and the tree $T = \emptyset$,
2. Run a random walk starting from an arbitrary node in $\mathcal{V} \setminus \Delta$,
3. Interrupt the random walk whenever it hits Δ,
4. Erase the cycles in the order that they appeared to obtain a path γ,
5. Update T and Δ by $T \leftarrow T \cup \gamma$ and $\Delta \leftarrow \Delta \cup s(\gamma)$ where $s(\gamma)$ denotes the nodes included in the walk γ,
6. If $\Delta \setminus \mathcal{V} = \emptyset$, terminate the algorithm and return the tree T. Otherwise go to step 2.

This algorithm keeps updating T with the loop erased walks in $\mathcal{G}$ until all vertices are covered. Therefore, when it terminates, the result in T is a spanning tree. Wilson's algorithm given in Alg. 2 in fact is an efficient implementation of this description.

Here we note that `RandomSuccessor` returns a random neighbor v of the given node u with the probability $\frac{w(u,v)}{d_u}$.

Algorithm 2 `RandomTreeWithRoot` [Wil96]

Inputs:
  $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$
  $r \in \mathcal{V}$
Initialize:
  $\forall i \in \mathcal{V}$, InTree$[i] \leftarrow$ `false`
  InTree$[r] \leftarrow$ `true`
  $\forall i \in \mathcal{V}$, Next[i] $\leftarrow -1$
for $i \leftarrow 1$ to $|\mathcal{V}|$ **do**
  $u \leftarrow i$
  while not InTree$[u]$ **do**
    Next[u] $\leftarrow$ `RandomSuccessor`$(u, \mathcal{G})$
    $u \leftarrow$ Next$[u]$
  end while
  $u \leftarrow i$
  while not InTree$[u]$ **do**
    InTree$[u] \leftarrow$ `true`
    $u \leftarrow$ Next$[u]$
  end while
end for
`return` Next

Having these detailed descriptions at hand, in the rest, we will answer three important questions about Wilson's algorithm:

1. How does one arrive at the distribution in Eq. (2.19) from this algorithm?
2. What is the time complexity of the algorithm?
3. How can we adapt this algorithm to sample Φ_Q?

One answer, that is often cited, to the first question was given by Wilson in his seminal paper. This proof heavily uses a representation of the random process conducted by the algorithm, which is called the stack representation. The stack representation proposed by [Wil96], is a sophisticated tool to analyze Wilson's algorithm. In this analysis, one considers a procedure over the stack representation, called cycle popping algorithm. This algorithm also terminates with a spanning tree and the distribution of the trees is the same as the spanning tree distribution induced by Wilson's algorithm. [Wil96] derives this distribution as in Eq. (2.19). A more *down-to-earth* approach [Mar00] uses the probability law of LERW given in Theorem 2.1.4. In this case, one considers Wilson's algorithm as a series of LERWs interrupted at evolving boundaries through the algorithm. Then, one derives the

law of the spanning trees by using the law of LERWS given in 2.1.4. In the following, we first go through this analysis to give a detailed answer to question 1 as it is more compatible with the main themes of this thesis. Later, we take a slight detour to formally introduce the stack representation and cycle popping algorithm (along with the links to Wilson's algorithm) because we will need them in the following chapters.

2.4.1.1. LERWs in Wilson's Algorithm

Given the root r, $k = 0$, $T_0 = \emptyset$ and $s(T_0) = \{r\}$, Wilson's algorithm can be considered in two steps:

1. Run a LERW that starts from an arbitrary $\mathcal{V} \setminus s(T_k)$ and is interrupted at $s(T_k)$.
2. Let γ be the path generated by the LERW until interruption. Then, update
   - $T_{k+1} \leftarrow T_k \cup \gamma$,
   - $s(T_{k+1}) \leftarrow s(T_k) \cup s(\gamma)$,
   - $k \leftarrow k + 1$,

   and go to step 1.

In Wilson's algorithm, these two steps are repeated until $s(T_k) = \mathcal{V}$ or equivalently T_k is a spanning tree. In other words, each branch in T_k is a LERW interrupted at certain boundaries. In the following, we use this fact to derive the distribution for T_k.

Theorem 2.4.1 (Correctness of Wilson's algorithm [Mar00])**.** *Wilson's algorithm generates random spanning trees distributed according to the law of Eq.* (2.19).

Proof. At an arbitrary step k of Wilson's algorithm, we sample a branch of a spanning tree by using a LERW interrupted at $s(T_{k-1})$. The distribution of such random walks read:

$$\mathbb{P}(T_k = \gamma \cup T_{k-1} | T_{k-1}) = \frac{\det \mathsf{L}_{-s(T_{k-1}) \cup s(\gamma)}}{\det \mathsf{L}_{-s(T_{k-1})}} \prod_{(i,j) \in \gamma} w(i,j).$$

Assume that the algorithm terminates at step K. Then, the probability law of the resulting tree is:

$$\mathbb{P}(T_K = \tau) = \prod_{k=1}^{K} \mathbb{P}(T_k = \gamma \cup T_{k-1} | T_{k-1}).$$

After telescopic cancellations, one recovers the distribution:

$$\mathbb{P}(T_K = \tau) = \prod_{k=1}^{K} \frac{\det \mathsf{L}_{-s(T_{k-1}) \cup s(\gamma_k)}}{\det \mathsf{L}_{-s(T_{k-1})}} \prod_{(i,j) \in \gamma_k} w(i,j) = \frac{1}{\det \mathsf{L}_{-r}} \prod_{(i,j) \in \tau} w(i,j).$$

Noticing that $\det \mathsf{L}_{-r}$ is the normalization constant (same for all $r \in \mathcal{V}$) for the random spanning trees finishes the proof. □

Here the tractable law of LERWs allows us to show the correctness of Wilson's algorithm *i.e.* it samples from the correct distribution. Moreover, this analysis is easy to adapt for different types of probabilistic event, such as the probability of having a tree (not spanning) in a random spanning tree. We refer the reader to [Mar00] for more details.

2.4.1.2. Stack Representation and Cycle Popping Algorithm

Define a random infinite stack $S^{(i)} = [S_1^{(i)}, S_2^{(i)} \dots]$ per node i which verifies $\mathbb{P}(S_l^{(i)} = k) = \frac{w(i,k)}{d_i}$ or alternatively $S_j^{(i)} = \texttt{RandomSuccessor}(\mathcal{G}, i)$ at every level j. Now we build a directed graph $\mathcal{G}_S$ by looking at the top level *i.e.* $l = 1$ of all stacks. Formally, $\mathcal{G}_S = (\mathcal{V}, \mathcal{E}_S, w)$ such that $\mathcal{E}_S = \{(i,j) : S_1^{(i)} = j\}$. An immediate result is that $\mathcal{G}_S$ has $n-1$ edges that form cycles and paths and if there is no cycle, then $\mathcal{G}_S$ is a spanning tree. Given these facts, we define a cycle-popping algorithm that pops the stacks associated with the cycles until there is none left, and we are left with a random spanning tree. Wilson's algorithm in fact is an implementation of the described cycle-popping algorithm. The only difference in Wilson's algorithm is the fact that we erase the cycles in the order they appear whereas in the cycle popping algorithm described, the order can be arbitrary. The following lemma shows that the order of the cycles popped is irrelevant to the resulting spanning tree. Therefore, the result of these two algorithms is the same random spanning tree.

Lemma 2.4.2 (Invariance to the order of cycles)**.** *If the cycle-popping algorithm terminates, then the order of cycles popped does not alter the resultant spanning tree.*

Proof. Assign a color l to each edge $(i, S_l^{(i)})$ according to their level in the stack. Then we define a colored cycle c in $\mathcal{G}_S$ as a cycle formed by colored edges. Note that two colored cycle might have the same set of edges, yet they are not the same as long as they contain a different set of colors on the edges.

Let $c_1, c_2, \ldots, c_k, c$ be a sequence of colored cycles that are observed during the execution of the cycle popping algorithm. We prove the theorem by showing that the cycle c will be popped in case of popping any other feasible sequence. To do so, we look at simpler case:

Assume that c' is a cycle that can be popped instead of c_1. The question that we need to answer is:

If we start from c' instead of c_1, can we still pop c?

We answer this question by considering the following cases:

- **Case 1:** c' and $c_{1,\ldots,k}$ shares no common vertex. In this case, the answer is yes because after popping c', one can still pop the sequence $c_{1,\ldots,k}$ and reach c.
- **Case 2:** c' and $c_{1,\ldots,k}$ has some common vertices. Let c_i be the first cycle that shares some vertices with c' *i.e.* $i = \min\{i \in \mathbb{N}^+ : c_i \cap c' \neq \emptyset\}$. If c' and c_i do not have the same edges, then they share at least one vertex v that has a different successor. However as the stack of v has not been popped by $c_1, \ldots, c_{i-1}$ up until c_i, its successor in c_i must be the same with its successor in c'. Applying this reasoning to all vertices in $c' \cap c_i$ shows that c' and c_i have the same edges. Moreover, the edges have the same set of colors, because the cycles $c_1, \ldots, c_{i-1}$ share no vertex with c' and c_i. Therefore, the colors on the edges of c' and c_i remains untouched. Then under the facts $c' = c_i$ and $c_{1,\ldots,i-1} \cap c' = \emptyset$, we may follow the order $c' = c_i, c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_k$ by reaching c.

With these two cases, we generalize that if a cycle c is encoded within the stacks, the cycle popping algorithm will eventually pop it regardless of the order that

$$\mathsf{S} = \begin{array}{c} \begin{array}{ccccc} S^{(1)} & S^{(2)} & S^{(3)} & S^{(4)} & S^{(5)} \end{array} \\ \left[\begin{array}{ccccc} 2 & 1 & 4 & 3 & \emptyset \\ 5 & 3 & 4 & 2 & \emptyset \\ 2 & 4 & 5 & 1 & \emptyset \\ 4 & 2 & 5 & 1 & \emptyset \\ & & \vdots & & \end{array}\right] \end{array}$$

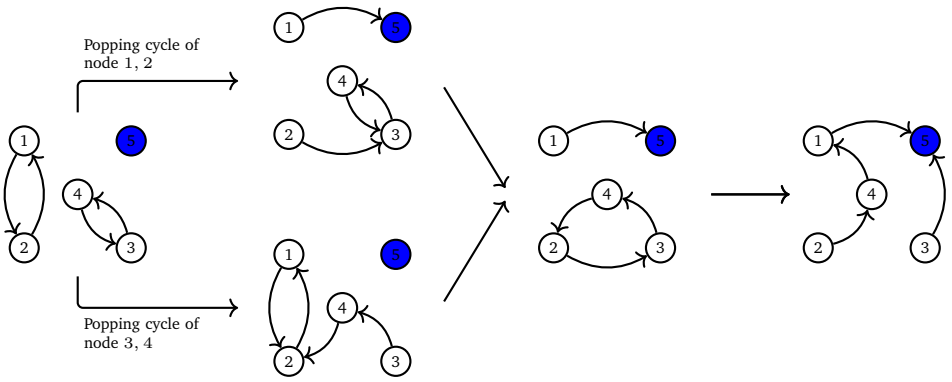


Fig. 2.9.: *A toy example on the order invariance of the popped cycles. The infinite matrix* S *contains the initial configuration of the infinite stacks in its columns. As we pop the cycles, the corresponding stacks pop the element at their first row. We follow different orders for popping the cycles but the popped cycles in the are the same ones yielding the same spanning tree as a result.*

we follow to pop cycles. In case that the algorithm terminates, the resultant structure, a spanning tree, is invariant from the order of popped cycles. □

By using this lemma, Wilson shows that the resulting tree structure by the cycle popping/his algorithm is distributed according to Eq. (2.19). As we already recovered this result in Theorem 2.4.1, we prefer not to repeat it in this manuscript.

Second Question. Now, we shift our focus to the second question; what is the time complexity of Wilson's algorithm? In this case, we refer by the time complexity to the expected number of calls of `RandomSuccessor` which is the most repeatedly called function in the algorithm. [Wil96] shows that this expectation is equal to the *mean commute time* ν, *i.e.* the expected number of steps of a random walk in a Markov chain until it returns to its initial state of the Markov chain used in Wilson's algorithm. The mean commute time is defined as:

$$\nu \coloneqq \sum_{i \in \mathcal{V}} \pi_i (t_{i,j} + t_{j,i})$$

where π is the stationary distribution of the Markov chain and $t_{i,j}$ is the expected number of steps to reach j from i. In case of Wilson's algorithm, the Markov

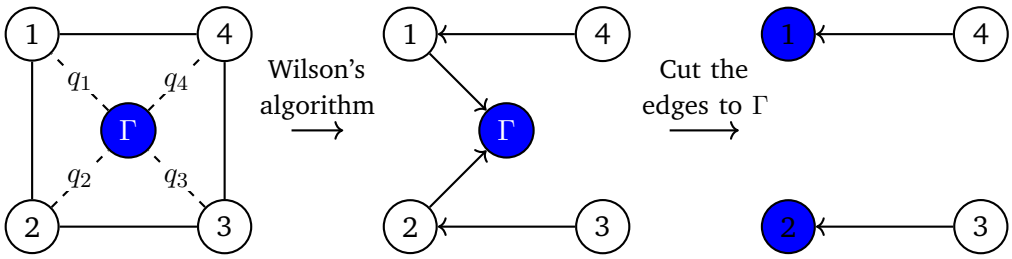


Fig. 2.10.: *We extend the original graph with an additional node* Γ*. Then we launch Wilson's algorithm to sample a spanning tree rooted in* Γ *on the extended graph. Cutting the edges incident to* Γ *gives a spanning forest distributed by the law of* Φ_Q

chain admits the transition probabilities $\mathsf{P}_{i,j} = \frac{w(i,j)}{d_i}$ except that it is absorbed at r. Then [Mar00] calculates the expected number of calls to `RandomSuccessor` in terms of P as follows:

$$\mathbb{E}[\#\text{ of calls to } \texttt{RandomSuccessor}] = \nu = \mathrm{tr}(\mathsf{I} - \mathsf{P}_{-r})^{-1}. \tag{2.20}$$

where P_{-r} is the reduced matrix by deleting the rows and columns corresponding to r.

Third Question. Finally, adapting Wilson's algorithm for spanning forests is straightforward:

- Add an additional node called Γ to $\mathcal{G}$,
- Add an edge from every node to Γ with an edge weight $w(i, \Gamma) = q_i$,
- Run Wilson's algorithm on the extended graph by choosing Γ as the root.

Alg. 3 gives an implementation of this adaptation (also See Fig. 2.10). This adaptation samples a random spanning tree on an extended graph with an additional node Γ. The distribution of these trees reads:

$$\mathbb{P}(T_\Gamma = \tau) \propto \prod_{(i,j)\in\tau} w(i,j)$$

Notice that by cutting the edges in T_Γ that are incident to Γ, one obtains a spanning forest. Moreover, the relation between these forests and trees is a bijection *i.e.* for each instance of T_Γ, one reaches a unique forest by cutting the edges incident to Γ. Then these forests admit the same distribution as T_Γ. By separating the edges linked to Γ, one recovers:

$$\mathbb{P}(\Phi_q = \phi) \propto \prod_{i\in\rho(\phi)} q_i \prod_{(i,j)\in\phi} w(i,j).$$

Finally, we obtain the time complexity of `RandomForest` by adapting Eq. (2.20):

$$\mathbb{E}\begin{bmatrix}\#\quad\text{of}\quad\text{calls}\quad\text{to}\\ \texttt{RandomSuccessor}\ \text{in}\\ \texttt{RandomForest}\end{bmatrix} = \operatorname{tr}\left((\mathsf{Q}+\mathsf{L})^{-1}(\mathsf{Q}+\mathsf{D})\right) \leq n + \frac{2m}{\min_i q_i}. \tag{2.21}$$

The upper-bound we give here is to give an understanding of this unknown trace in terms of graph parameters. However, it is usually not tight. We refer the reader to App. A.1 for the derivation of the upper bound.

Algorithm 3 `RandomForest` [ACGM18]

Inputs:
  $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$
  $Q = \{q_1, \ldots, q_n\}$
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` $\leq \frac{q_i}{q_i + \mathsf{d}[u]}$ **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

2.4.2 DPPs in Random Spanning Forests

In the scope of this thesis, the most interesting properties of RSFs are their links with DPPs. In particular, the edges and roots of Φ_q are sampled from certain DPPs whose kernel matrices are tractable in terms of the edge incidence matrix or graph Laplacian. These results have their roots from the results in random spanning trees [BP93], later summarized in [ACGM18]. We give in this section some important ones.

Theorem 2.4.3 (The roots and edges of Φ_Q are DPPs)**.** *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, the roots $\rho(\Phi_Q)$ and edges $\mathcal{S}$ of Φ_Q is an L-ensemble with:*

$$\forall V \subseteq \mathcal{V}, S \subseteq \mathcal{E}, \quad \mathbb{P}((\rho(\Phi_Q), \mathcal{S}) = (V, S)) \propto \det \mathcal{L}_{V \cup S}$$
$$\text{with } \mathcal{L} = \begin{bmatrix} \mathsf{Q}^{1/2} \\ \mathsf{B} \end{bmatrix} \begin{bmatrix} \mathsf{Q}^{1/2} & \mathsf{B}^\top \end{bmatrix} = \begin{bmatrix} \mathsf{Q} & \mathsf{Q}^{1/2}\mathsf{B}^\top \\ \mathsf{B}\mathsf{Q}^{1/2} & \mathsf{B}\mathsf{B}^\top \end{bmatrix},$$

where $\mathsf{Q} = \operatorname{diag}(Q)$.

Proof. We prove this theorem algebraically by showing that for any fixed set $V \subset \mathcal{V}$ and $S \subset \mathcal{E}$, the determinant $\det \mathcal{L}_{V \cup S}$ is proportional to the distribution of RSFs (at the right hand side):

$$\det \mathcal{L}_{V \cup S} \propto \mathbb{P}(\rho(\Phi_Q), \mathcal{S} = V, S) = \frac{1}{Z_Q} \begin{cases} \prod_{r \in V} q_r \prod_{(i,j) \in S} w(i,j), & \text{if } S \text{ forms a spanning forest with the root set } V, \\ 0, & \text{otherwise.} \end{cases}$$

Recall that the adaptation of Wilson's algorithm for forests samples a spanning tree over an extended graph with a node Γ. Let us start by writing the edge incidence matrix of this graph:

$$\mathsf{B}^{ext} = \begin{bmatrix} \mathsf{Q}^{1/2} & -\mathsf{Q}^{1/2}\mathbf{1} \\ \mathsf{B} & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{(m+n)\times(n+1)}$$

An immediate result of this definition links B^{ext} with $\mathcal{L}$:

$$\mathcal{L} = (\mathsf{B}^{ext}_{:|-\Gamma})^\top \mathsf{B}^{ext}_{:|-\Gamma}.$$

In addition, given a fixed set S' of edges on the extended graph, Theorem 2.1.3 yields

$$|\det \mathsf{B}^{ext}_{S'|-\Gamma}| = \begin{cases} \prod_{(i,j)\in S'} \sqrt{w(i,j)}, & \text{if } S' \text{ forms a spanning tree rooted in } \Gamma, \\ 0, & \text{otherwise.} \end{cases}$$

S' consists of the edges incident to Γ and the edges from $\mathcal{E}$. As aforementioned, when we delete the former group, we obtain a rooted spanning forest $\phi = (\mathcal{V}, S, w)$ with roots V are the neighbors of Γ in S'. Therefore, there is a bijection between the roots V and the edges incident to Γ in S'. Relying on this fact we rewrite:

$$|\det \mathsf{B}^{ext}_{S\cup V|-\Gamma}| = \begin{cases} \prod_{r\in V} \sqrt{q_r} \prod_{(i,j)\in S} \sqrt{w(i,j)}, & \text{if } S \text{ forms a spanning forest with the root set } V, \\ 0, & \text{otherwise.} \end{cases}$$

Then we finish the proof by writing the determinant formula for matrix multiplications:

$$\det \mathcal{L}_{V\cup S} = \det(\mathsf{B}^{ext}_{V\cup S|-\Gamma})^2 = \begin{cases} \prod_{r\in V} q_r \prod_{(i,j)\in S} w(i,j), & \text{if } S' \text{ forms a spanning forest with the root set } V, \\ 0, & \text{otherwise.} \end{cases}.$$

One recovers the probability $\mathbb{P}((\rho(\Phi_Q), \mathcal{S}) = (V, S))$ by normalizing the right hand side by Z_Q. □

This L-ensemble is in fact a projection DPP by Lemma 2.2.4. In order to see this, first notice that the roots-edges process is a fixed size DPP as $|\mathcal{S}| + \rho(\Phi_Q) = n$ is always true. Moreover, one has $\operatorname{rank} \mathcal{L} = n$. [3] Then the marginal kernel of the projection DPP can be written as $\mathsf{M} = \sum_{k=1}^{n} \mathbf{x}_k \mathbf{x}_k^T$ where $\mathbf{x}_k$ is the k-th eigenvector of $\mathcal{L}$ associated to a non-zero eigenvalue.

[3] This is, because firstly $\operatorname{rank} \mathcal{L} = \operatorname{rank} \mathsf{B}^{ext}_{:|-\Gamma}$ and $\operatorname{rank} \mathsf{B}^{ext}_{:|-\Gamma} \geq n$ due to its $n \times n$ diagonal matrix component Q. Also $\mathsf{B}^{ext}_{:|-\Gamma}$ is a $n \times (m+n)$ long matrix, one has $\operatorname{rank} \mathcal{L} \leq n$ which implies the equality $\operatorname{rank} \mathcal{L} = n$.

Remark 2 (Restriction to the roots)**.** Restricting a DPP to a subset of its outcome space also yields a DPP restricting its marginal kernel (See Lemma 2.2.1). We can calculate the marginal kernel of $\rho(\Phi_Q)$ by using this property.

Theorem 2.4.4. *$\rho(\Phi_Q)$ is a DPP with marginal kernel* $\mathsf{K} = (\mathsf{L}+\mathsf{Q})^{-1}\mathsf{Q}$.

Proof. The proof have three main steps:

1. We first derive the marginal kernel of the root-edge process in terms of blocks of $\mathcal{L}$.
2. Then we invoke Lemma 2.2.1 to derive the kernel only for the roots $\rho(\Phi_Q)$.
3. Finally, we show a DPP with the resulting kernel is equivalent to a DPP with the kernel $\mathsf{K} = (\mathsf{L}+\mathsf{Q})^{-1}\mathsf{Q}$.

For the first step, let us diagonalize $\mathcal{L} = \mathsf{X}\mathsf{D}\mathsf{X}^\top$ where $\mathsf{X} \in \mathbb{R}^{(n+m)\times n}$ contains the non-trivial eigenvectors and $\mathsf{D} \in \mathbb{R}^{n\times n}$ is a diagonal matrix whose entries contain the non-zero eigenvalues. The marginal kernel of the root-edge process reads:

$$\mathsf{M} = \mathsf{X}\mathsf{X}^\top.$$

In order to write this matrix in terms of the blocks of $\mathcal{L}$, we use the following trick:

$$\lim_{\alpha\to\infty} \alpha\mathcal{L}(\alpha\mathcal{L}+\mathsf{I})^{-1} = \mathsf{X}\mathsf{X}^\top.^{a}$$

In left hand side, we write the block matrix form of $\mathcal{L}$ within the matrix inverse:

$$\lim_{\alpha\to\infty} \alpha\mathcal{L}\left(\begin{bmatrix}\alpha\mathsf{Q}+\mathsf{I} & \alpha\mathsf{Q}^{1/2}\mathsf{B}^\top \\ \alpha\mathsf{B}\mathsf{Q}^{1/2} & \mathsf{I}+\alpha\mathsf{B}\mathsf{B}^\top\end{bmatrix}\right)^{-1} = \mathcal{L}\lim_{\alpha\to\infty}\left(\begin{bmatrix}\mathsf{Q}+\alpha^{-1}\mathsf{I} & \mathsf{Q}^{1/2}\mathsf{B}^\top \\ \mathsf{B}\mathsf{Q}^{1/2} & \alpha^{-1}\mathsf{I}+\mathsf{B}\mathsf{B}^\top\end{bmatrix}\right)^{-1}$$

Before going further calculations with the block matrix inversion formula, recall that we want to derive the kernel only for the root set. This means by Lemma 2.2.1 that we are interested in deriving only the first $n \times n$ block of M. Then let us restrict our calculations on the region of interests. This means that by the block matrix inversion, one has:

$$\mathsf{M}_{\{1,\dots,n\}} = \begin{bmatrix}\mathsf{Q} & \mathsf{Q}^{1/2}\mathsf{B}^\top\end{bmatrix}\lim_{\alpha\to\infty}\begin{bmatrix}\mathsf{S} \\ -(\mathsf{B}\mathsf{B}^\top+\alpha^{-1}\mathsf{I})^{-1}\mathsf{B}\mathsf{Q}^{1/2}\mathsf{S}\end{bmatrix}$$

where $\mathsf{S} = (\mathsf{Q} + \alpha^{-1}\mathsf{I} - \mathsf{Q}^{1/2}\mathsf{B}^\top(\mathsf{B}\mathsf{B}^\top + \alpha^{-1}\mathsf{I})^{-1}\mathsf{B}\mathsf{Q}^{1/2})^{-1}$ is the Schur's complement. After several algebraic manipulations (See Appendix A.3), this limit converges to:

$$\mathsf{M}_{\{1,\dots,n\}} = \lim_{\alpha\to\infty}(\mathsf{Q} - \mathsf{Q}^{1/2}\mathsf{B}^\top(\mathsf{B}\mathsf{B}^\top + \alpha^{-1}\mathsf{I})^{-1}\mathsf{B}\mathsf{Q}^{1/2})\mathsf{S} = \mathsf{Q}^{1/2}(\mathsf{L}+\mathsf{Q})^{-1}\mathsf{Q}^{1/2}. \tag{2.22}$$

We finish the proof by showing the equivalence of the two kernels $\mathsf{M}_{\{1,\dots,n\}}$ and K. As both $(\mathsf{L}+\mathsf{Q})$ and $\mathsf{Q}^{1/2}$ are square matrices, one has for all subsets $X \subseteq \mathcal{V}$:

$$\begin{aligned}\det(\mathsf{Q}^{1/2}(\mathsf{L}+\mathsf{Q})^{-1}\mathsf{Q}^{1/2})_X &= \det(\mathsf{Q}^{1/2})_X \det(\mathsf{L}+\mathsf{Q})^{-1})_X \det(\mathsf{Q}^{1/2})_X \\ &= \det(\mathsf{Q})_X \det(\mathsf{L}+\mathsf{Q})^{-1})_X \\ &= \det(\mathsf{Q}(\mathsf{L}+\mathsf{Q})^{-1})_X = \det \mathsf{K}_X\end{aligned}$$

□

[a] This is easy to check diagonalizing $\alpha\mathcal{L}(\alpha\mathcal{L}+\mathsf{I})^{-1}$ as $\mathsf{X}(\mathsf{I}+(\alpha\mathsf{D})^{-1})^{-1}\mathsf{X}^\top$. As $\alpha\to\infty$, the limit converges to $\mathsf{X}\mathsf{X}^\top$.

The connection of K with RSFs goes even beyond. The following theorem indicates that the off-diagonal entries $\mathsf{K}_{i,j}$ corresponds to certain probabilities in RSFs.

Theorem 2.4.5. *For a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, in a realization of RSF Φ_Q, the probability of node i to be rooted in node j reads:*

$$P(r_{\Phi_Q}(i) = j) = \mathsf{K}_{i,j} \tag{2.23}$$

Proof. Let $\mathcal{F}_{ij}$ be all spanning forests in which node i is rooted at j. Then, one writes the probability $P(r_{\Phi_Q}(i) = j)$ by using the definition of Φ_Q:

$$\begin{aligned}P(r_{\Phi_Q}(i) = j) &= \frac{\sum_{\phi\in\mathcal{F}_{ij}}\prod_{l\in\rho(\phi)} q_l \prod_{a,b\in\phi} w(a,b)}{\sum_{\phi\in\mathcal{F}}\prod_{l\in\rho(\phi)} q_l \prod_{a,b\in\phi} w(a,b)} \\ &= \frac{q_j \sum_{\phi\in\mathcal{F}_{ij}}\prod_{l\in\rho(\phi)/j} q_l \prod_{a,b\in\phi} w(a,b)}{\sum_{\phi\in\mathcal{F}}\prod_{l\in\rho(\phi)} q_l \prod_{a,b\in\phi} w(a,b)} = q_j\frac{A_{ij}}{Z}\end{aligned} \tag{2.24}$$

where A_{ij} and Z are scalars. In the rest, we leverage the revisited theorems to give close form expressions for A_{ij} and Z.

Given the extended graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', w)$ and a subset $\mathcal{S} = \mathcal{S}_i \cup \mathcal{S}_\Gamma$ with $|\mathcal{S}| = |\mathcal{V}| - 1$, one can write the resultant absolute determinant for $R = \{i, \Gamma\}$ by using Theorem 2.1.3:

$$|\det \mathsf{B}'_{\{\mathcal{S}|-i,-\Gamma\}}| = \left[\prod_{a,b\in\mathcal{S}_\Gamma} w(a,b) \prod_{a,b\in\mathcal{S}_i} w(a,b)\right]^{1/2}$$

whenever $\mathcal{S}$ forms a spanning forest on $\mathcal{G}'$ with two trees in which node i and Γ are disconnected. Similarly, one obtains the same result by plugging node j instead of i. Then, the following holds:

$$|\det \mathsf{B}'_{\{\mathcal{S}|-i,-\Gamma\}} \det \mathsf{B}'_{\{\mathcal{S}|-j,-\Gamma\}}| = \left[\prod_{a,b\in\mathcal{S}_\Gamma} w(a,b) \prod_{a,b\in\mathcal{S}_i=\mathcal{S}_j} w(a,b)\right]$$

whenever $\mathcal{S}$ forms a spanning forest on $\mathcal{G}'$ with two trees in which Γ is not connected to either i or j. This implies that i and j are contained the same tree. Notice that adding the edge (j, Γ) to such subset $\mathcal{S}$ reconstructs a spanning tree on Γ. Then, one can acquire a rooted spanning forest from this tree as done in the sampling procedure. In the resultant forest, the node j is set as the root of the tree that contains both i and j. In other words, it is an element of the forest set $\mathcal{F}_{ij}$. Moreover, the edge weight product of this forest is equal to $q_j|\det \mathsf{B}'_{\{\mathcal{S}|-i,-\Gamma\}} \det \mathsf{B}'_{\{\mathcal{S}|-j,-\Gamma\}}|$. Then, the sum of the edge weight products of all forests in $\mathcal{F}_{ij}$ equals to:

$$\sum_{\phi\in\mathcal{F}_{ij}} \prod_{l\in\rho(\phi)} q_l \prod_{a,b\in\phi} w(a,b) = q_j \sum_{\mathcal{S}\subseteq\mathcal{E}',|\mathcal{S}|=|\mathcal{V}'|-1} |\det \mathsf{B}'_{\mathcal{S}|-i,-\Gamma} \det \mathsf{B}'_{\mathcal{S}|-j,-\Gamma}| \quad (2.25)$$

By using Lemma A.4.1, we exchange the absolute value operator with the sign of the expression. Then, by using the Cauchey-Binet, one has

$$\begin{aligned}\sum_{\phi\in\mathcal{F}_{ij}} \prod_{l\in\rho(\phi)} q_l \prod_{a,b\in\phi} w(a,b) &= (-1)^{i+j} q_j \sum_{\mathcal{S}\subseteq\mathcal{E}',|\mathcal{S}|=|\mathcal{V}'|-1} \det \mathsf{B}'_{\mathcal{S}|-i,-\Gamma} \det \mathsf{B}'_{\mathcal{S}|-j,-\Gamma} \\ &= (-1)^{i+j} q_j \det \mathsf{L}'_{-i,-\Gamma|-j,-\Gamma}\end{aligned} \quad (2.26)$$

where L' is the graph Laplacian of $\mathcal{G}'$. This completes the computation of A_{ij} in Eq. (2.24). In a similar fashion, one computes Z as $\det \mathsf{L}'_{-\Gamma|-\Gamma}$ by doing the

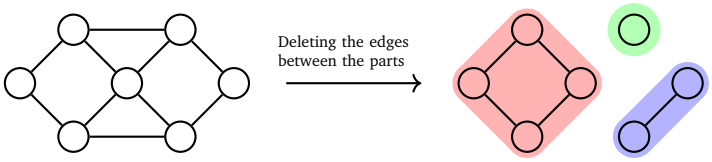


Fig. 2.11.: Fixing the partition created by Φ_Q ensures that the edges between parts are never sampled. Moreover, every tree sampled in such process has to be a spanning tree of a part. Therefore, given the partition, sampling Φ_Q boils down to sampling independent spanning trees in each part.

same calculations without i and j. Noticing $\mathsf{L}'_{-\Gamma|-\Gamma} = (\mathsf{L}+\mathsf{Q})$ reads the famous Cramer's formula for inverse matrices:

$$P(r_{\Phi_Q}(i)=j) = \frac{A_{ij}}{Z} = q_j \frac{(-1)^{i+j}\det(\mathsf{L}+\mathsf{Q})_{-i,|-j}}{\det(\mathsf{L}+\mathsf{Q})} = q_j(\mathsf{L}+\mathsf{Q})^{-1}_{i,j} = \mathsf{K}_{i,j}$$

□

2.4.2.1. Random Partitions induced by Φ_Q

A partition of the vertex set is any set $\mathcal{P} := (\mathcal{V}_1, \dots, \mathcal{V}_K)$ that satisfies $\forall k, k' \in \{1, \dots, K\}, \mathcal{V}_k \cap \mathcal{V}_{k'} = \emptyset$ and $\cup_{k=1}^K \mathcal{V}_k = \mathcal{V}$. A straight-forward partition in a spanning forest ϕ is given by its connected components, or equivalently trees. We denote this partition by $\pi(\phi) := (\mathcal{V}_1, \mathcal{V}_2, \dots, \mathcal{V}_{|\rho(\phi)|})$. Let us enumerate the trees of ϕ from 1 to $|\rho(\phi)|$. We define the tree number of a node as the number of the tree that it belongs to and it is denoted by $t(\pi(\phi), i)$ where $t : (\pi(\mathcal{F}), \mathcal{V}) \to \{1, \dots, |\rho(\phi)|\}$ maps every node to its tree number.

Another surprising theoretical result on Φ_Q emerges when we look at the conditional probabilities of roots over the random partition $\pi(\Phi_Q)$:

Proposition 2.4.6 (Roots in a fixed partition [ACGM18])**.** *For a fixed subset $V \subseteq \mathcal{V}$ and a fixed partition $\mathcal{P}$, the conditional probability of roots for a fixed partition of random spanning forests is:*

$$\mathbb{P}(\rho(\Phi_Q) = V | \pi(\Phi_Q) = \mathcal{P}) = \prod_{u \in V} \frac{q_u}{\sum_{v \in \mathcal{V}_{t(\mathcal{P},v)}} q_v}. \tag{2.27}$$

Proof. Consider the subgraphs of $\mathcal{G}$ by deleting the edges in between the subsets given by the fixed partition (See Fig. 2.11). Notice that sampling Φ_Q on a fixed partition $\mathcal{P}$ is equivalent to sampling random spanning trees independently on

each such subgraph. Therefore we need to look at the probability distribution of the roots in these trees to derive the conditional probability. To do so, let us denote the random spanning tree restricting Φ_Q to only spanning trees by T. The probability distribution of T reads:

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

Then, the probability of T being rooted in rooted in node r is:

$$\mathbb{P}(T \in \mathcal{T}_r) = \frac{q_r \sum_{\tau_r \in \mathcal{T}_r} \prod_{(i,j)\in\tau_r} w(i,j)}{\sum_{r'\in\mathcal{V}} q_{r'} \sum_{\tau_{r'}\in\mathcal{T}_{r'}} \prod_{(i,j)\in\tau_{r'}} w(i,j)}. \tag{2.28}$$

Recall that given a undirected spanning tree, for each node r, there exists exactly one orientation of edges which makes r root. Therefore, the sum $\sum_{\tau_r\in\mathcal{T}_r} \prod_{(i,j)\in\tau_r} w(i,j)$ is a constant for all $r \in \mathcal{V}$. Then, after the cancellation in both numerator and denominator, one has:

$$\mathbb{P}(T \in \mathcal{T}_r) = \frac{q_r}{\sum_{r'\in\mathcal{V}} q_{r'}}.$$

As this is the probability of each tree sampled in Φ_Q conditioned over the partition $\mathcal{P}$, recalling the independent nature of the trees finishes the proof:

$$\mathcal{P}(\rho(\Phi_Q) = V | \pi(\Phi_Q) = \mathcal{P}) = \prod_{u\in V} \frac{q_u}{\sum_{v\in\mathcal{V}_{t(\mathcal{P},v)}} q_v}.$$

□

This last result on RSFs covers all the technical results that we need for the rest. In the next section, we take a break before proceeding to the main materials of the thesis and summarize some of the important results.

2.5 Conclusion

Overall, we have covered many results from diverse concepts in this section. We finalize this chapter by highlighting the ones that will be repeatedly used in the

following chapters. Therefore, we kindly suggest the reader to keep these following couple of pages at reach as a *cheat-sheet* through the rest of the journey.

2.5.1 Graph Laplacian

One matrix to rule them all...

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, the graph Laplacian $\mathsf{L} = \mathsf{D} - \mathsf{W}$ is a simple but resourceful object in graph theory, combinatorics, probability theory, computer science, signal processing and machine learning. Closer to the main themes of this thesis, in this section, we take a look on two significant results on certain determinants of L:

We first look at its link with the spanning trees that can be found on $\mathcal{G}$. This link is the celebrated matrix-tree theorem which gives the number of all the spanning trees on a unweighted graph in a simple closed-form expression:

$$\forall i \in \mathcal{V}, \quad |\mathcal{T}| = \det \mathsf{L}_{-i|-i}.$$

We later generalize it for weighted graphs:

$$\forall i \in \mathcal{V}, \quad \det \mathsf{L}_{-i|-i} = \sum_{\tau \in \mathcal{T}} \prod_{(i,j) \in \mathcal{E}_\tau} w(i,j).$$

These results extend on rooted spanning forests on $\mathcal{G}$. In Theorem 2.1.3, we adapt Poincaré's results for spanning forests that yields a useful device to count all spanning forests on $\mathcal{G}$ given the root set R:

$$\forall R \subseteq \mathcal{V}, \quad \det \mathsf{L}_{-R} = \sum_{\phi \in \mathcal{F}_R} \prod_{(i,j) \in \mathcal{E}_\phi} w(i,j).$$

Second, we give the links of L with a special type of random walk, called loop-erased random walks. These random walks are obtained by erasing cycles as they appear in the usual random walks. Due to the cycle erasure, they are not Markovian random processes *i.e.* the next vertex to land does not solely depend on the current vertex. Unlike many non-Markovian process, LERWs surprising have a tractable distribution, which can be written in terms of principle minors of L:

$$\mathbb{P}(LE(W) = \gamma) = \frac{\det \mathsf{L}_{-\Delta \cup s(\gamma)}}{\det \mathsf{L}_{-\Delta}} \prod_{(i,j) \in \gamma} w(i,j)$$

where $\Delta \subset \mathcal{V}$ is the boundary where the LERW $LE(W)$ is stopped, γ is a fixed path and $s(\gamma)$ denotes the nodes visited in γ.

2.5.2 Determinantal Point Processes and Randomized Linear Algebra

A determinantal point process $\mathcal{X}$ is a special type of point process over a set Ω that verifies:

$$\mathbb{P}(S \subseteq \mathcal{X}) = \det \mathsf{K}_S.$$

for some square matrix K. This definition also yields that:

$$\mathbb{E}[|\mathcal{X}|] = \mathbb{E}\left[\sum_{i\in\Omega} \mathbb{I}(i \in \mathcal{X})\right] = \sum_{i\in\Omega} \mathbb{P}(i \in \mathcal{X}) = \sum_{i\in\Omega} \mathsf{K}_{i,i} = \mathrm{tr}(\mathsf{K}).$$

The key feature of DPPs is that they can encode *negative association* between different elements in Ω. For a symmetric K, one can examine this phenomena by looking at the joint probabilities in the case of two elements $i, j \in \Omega$:

$$\mathbb{P}(i, j \in \mathcal{X}) = \mathbb{P}(i \in \mathcal{X})\mathbb{P}(j \in \mathcal{X}) - \mathsf{K}_{i,j}\mathsf{K}_{j,i} \leq \mathbb{P}(i \in \mathcal{X})\mathbb{P}(j \in \mathcal{X}).$$

As the calculation suggests, having node i in $\mathcal{X}$ reduce the chances of also having node j in $\mathcal{X}$.

Although DPPs by definition seem a bit abstract, real-life examples and use cases of DPPs are numerous. In our case, we are interested in their use in randomized linear algebra. RLA is a set of numerical linear algebra tools that replaces expensive direct operations with randomized approximations. We consider solving least-squares problem (also regularized) with such approximation scheme. In this case, we are given some observations $\mathbf{b} \in \mathbb{R}^n$ and predictors $\mathsf{A} \in \mathbb{R}^{n\times p}$ and we solve:

$$\mathbf{x}^\star = \underset{\mathbf{x}\in\mathbb{R}^p}{\operatorname{argmin}} ||\mathsf{A}\mathbf{x} - \mathbf{b}||_2.$$

The exact solution $\mathbf{x}^\star$ reads:

$$\mathbf{x}^\star = \mathsf{A}^\dagger \mathbf{b}.$$

In the regularized case, it is:

$$\mathbf{x}^\star = (\mathsf{A}^\top \mathsf{A} + \lambda \mathsf{R})^{-1} \mathsf{A}^\top \mathbf{b}.$$

Generically these inversions requires $\mathcal{O}(np^2)$ computations with the direct methods. RLA proposes randomized methods to avoid this expensive computation. In a nutshell, these methods first sample a smaller portion *i.e.* some rows of A and (also $\mathbf{b}$ depending on the randomization) at random. Then, they do the expensive

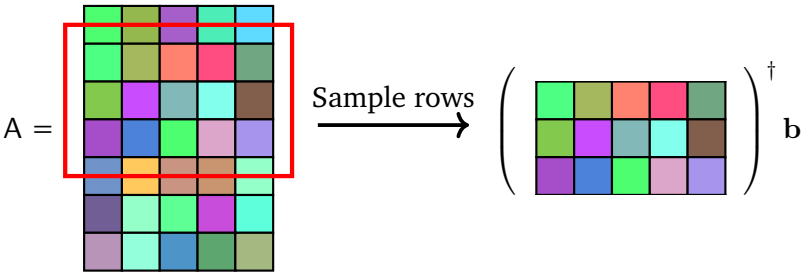
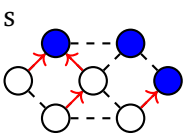


Fig. 2.12.: An illustration of the sample-and-solve scheme. First we sample a small portion of the matrix A*, then proceed to the expensive operation.*

operations on the sampled portion which avoid the expensive global operations (Also see Fig. 2.12). Indeed, the type of the randomization *i.e.* the probability distribution is of central importance in such a scheme. This is where DPPs come into play. Sampling the rows from certain DPPs yield unbiased estimators for both least-squares and regularized least-squares solution. Closer to this, one main contribution of this thesis is a new estimator whose variance is significantly reduced w.r.t. the DPP-based estimators (More details in Section 2.2). In addition, we in Chapter 3 introduce elegant estimators for graph-regularized least-squares problems *i.e.* the solution is regularized to be smooth on a graph. Surprisingly these estimators are also DPP-based estimators as the random process (random spanning forests) that they use is a DPP.

2.5.3 Random Spanning Forests

The last but not least concept we present in this chapter is the random spanning forests *i.e.* process of randomly selecting spanning forest on a given graph. In fact, most of the theoretical results we rely on in this thesis is closely related with the RSFs. We denote a RSF by Φ_Q and write its distribution as follows:



$$P(\Phi_Q = \phi) = \frac{1}{Z_Q} \prod_{i \in \rho(\phi)} q_i \prod_{(i,j) \in \phi} w(i,j)$$

where $Q = \{q_1, \ldots, q_n\}$ with $\min_i q_i > 0$ and Z_Q is the normalization constant over all forests. Let us highlight some of the significant properties and theorems of Φ_Q:

- **Wilson's algorithm for RSFs.** Wilson's algorithm is originally developed for generating spanning trees at uniform. With a slight modification, it can be used for generating samples of Φ_Q. We give this modified algorithm Alg. 3, we

also note the time complexity of this algorithm in terms of number of random variables sampled:

$$\mathbb{E}\begin{bmatrix}\#\quad \text{of}\quad \text{calls}\quad \text{to}\\ \texttt{RandomSuccessor in}\\ \texttt{RandomForest}\end{bmatrix} = \operatorname{tr}\left((\mathsf{Q}+\mathsf{L})^{-1}(\mathsf{Q}+\mathsf{D})\right),$$

where $\mathsf{Q} = \operatorname{diag}(Q)$. While it is much more efficient than naive sampling *i.e.* enumerating all forests and choosing one at random, it is known as the most efficient algorithm for generating spanning/trees.

- **The root-edge process.** The roots and edges in Φ_Q are sampled from an L-ensemble with:

$$\forall V \subseteq \mathcal{V}, S \subseteq \mathcal{E}, \quad \mathbb{P}((\rho(\Phi_Q), \mathcal{S}) = (V, S)) \propto \det \mathcal{L}_{V \cup S}$$
$$\text{with } \mathcal{L} = \begin{bmatrix}\mathsf{Q}^{1/2}\\ \mathsf{B}\end{bmatrix}\begin{bmatrix}\mathsf{Q}^{1/2} & \mathsf{B}^\top\end{bmatrix} = \begin{bmatrix}\mathsf{Q} & \mathsf{Q}^{1/2}\mathsf{B}^\top\\ \mathsf{B}\mathsf{Q}^{1/2} & \mathsf{B}\mathsf{B}^\top\end{bmatrix}$$

  A consequence of this along with the restriction lemma (Lemma 2.2.1) is that the root set $\rho(\Phi_Q)$ is a DPP with the marginal kernel $\mathsf{K} = (\mathsf{Q}+\mathsf{L})^{-1}\mathsf{Q}$.

- $\mathbb{P}(r_{\Phi_Q}(i) = j) = \mathsf{K}_{i,j}$. This identity is another surprising result between spanning forests/trees and Laplacian-based algebra. Its proof is an algebraic proof mainly based on Poincaré's theorem for counting spanning forests with root constraints[4].

- **Roots in fixed partitions.** The final theorem we visit gives the distribution of the roots when we generate a sample of Φ_Q conditioned over a fixed partition *i.e.* $\Phi_Q|\pi(\Phi_Q) = \mathcal{P}$. If the partition is fixed, sampling Φ_Q boil downs to sampling independent trees within each subset of the partition. Then within each subset, the probability having a node i as the root becomes $\frac{q_i}{\sum_{j\in\mathcal{V}_{t(\mathcal{P},i)}} q_j}$ where $\mathcal{V}_{t(\mathcal{P},i)}$ is the subset that contains i. By independence of the trees, one has:

$$\mathcal{P}(\rho(\Phi_Q) = V|\pi(\Phi_Q) = \mathcal{P}) = \prod_{u\in V}\frac{q_u}{\sum\limits_{v\in\mathcal{V}_{t(\mathcal{P},v)}} q_v}.$$

  Note that for $q_1 = q_2 = \cdots = q_n = q$, this boils down to a uniform distribution.

With this last point, we reach to the end of our short(!) summary on the technical background for the next material.

[4] It is worth to note that to best of our knowledge, this is a new proof compared to the existing probabilistic and combinatorial proofs [ACGM18; CK78; CA02]

3 Graph Tikhonov Regularization and Interpolation with RSFs

> *C'est par la logique qu'on démontre, c'est par l'intuition qu'on invente.*
>
> — **Henri Poincaré**

This chapter is dedicated to presenting the contributions of the thesis for solving the Graph Tikhonov Regularization (GTR) (a graph adaptation of regularized linear regression) and Graph Interpolation (GI) problems. These contributions are two-fold; i/ the RSF-based Monte Carlo estimators for solving GTR and GI and ii/ their improved versions via variance reduction techniques. In order to do so, we first give formal definitions of these problems and show how they relate to each other. Then we leverage the elegant theory of RSFs to define the methods proposed that yield unbiased estimates for these problems. In parallel, we give several ways to improve the estimators. Under certain conditions, these techniques can substantially decrease their variance with a cheap additional computational cost. In addition, we find several other use cases for these estimators where one needs to avoid expensive computations within intermediate operations, such as semi-supervised learning on graphs, Newton's method and Alternating Direction Method of Multipliers (ADMM). In parallel, we also provide empirical illustrations or comparisons of the techniques proposed with state-of-the-art methods.

Contents

3.1 Problem Definition

As in many signal processing applications, the measurements over graph signals are either noisy or incomplete *i.e.* missing over some subset of nodes. A big chunk of tools to overcome these issues proposes to *smooth* out the signal over the given graph. In the case of denoising, smoothing means that we eliminate specific components of the measurements that are highly varying over the graph by assuming that the original signal is smooth on the graph. Similarly, in an interpolation setup, we complete the missing parts of the signal so that it remains smooth on the underlying graph. A unifying approach tackles both problems with a single optimization problem. This approach adapts the Tikhonov regularization for graph signals and draws the following optimization problem:

Problem 3.1.1. *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ and measurements $\mathbf{y} = [y_1, \ldots, y_p]^\top \in \mathbb{R}^p$ over the vertices $V \subseteq \mathcal{V}$ with $|V| = p$, we attempt to recover the original signal by solving:*

$$\hat{\mathbf{x}} = \underset{\mathbf{x} \in \mathbb{R}^n}{\operatorname{argmin}} \, ||\mathsf{M}\mathbf{y} - \mathbf{x}||_{\mathsf{Q}}^2 + \mathbf{x}^\top \mathsf{L}\mathbf{x}. \tag{3.1}$$

where $||\mathbf{a}||_{\mathsf{Q}} = \mathbf{a}^\top \mathsf{Q}\mathbf{a}$ is the Mahalonobis distance and Q is a diagonal matrix where each $\mathsf{Q}_{i,i}$ is a node-wise regularization parameter. We define the selection matrix $\mathsf{M} \in \mathbb{R}^{n \times p}$ as a rectangular diagonal matrix where $\mathsf{M}_{i,i} = \mathbb{I}(i \in V)$. This problem admits a closed form solution in the following form:

$$\hat{\mathbf{x}} = (\mathsf{Q} + \mathsf{L})^{-1} \mathsf{Q}\mathsf{M}\mathbf{y} \tag{3.2}$$

This generic linear regression formulation, in fact, has been revisited in various contexts, from solving heat equations in physics to semi-supervised learning on graphs in machine learning. Among many, two cases exemplify the primary uses of GTR and GI:

Graph signal filtering: Graph signal filtering means eliminating certain components of the graph signal and retain the parts of interest. The common practice in linear filtering schemes is through the frequency analysis of the signal *i.e.* eliminating certain frequency components. A fundamental tool in this type of analysis is the

In this case, one considers $\mathsf{M} = \mathsf{I} \in \mathbb{R}^{n \times n}$ and $\mathsf{Q} = q\mathsf{I} \in \mathbb{R}^{n \times n}$

Fourier transform that interprets the signal in terms of its frequency components. However, the adaptation of the Fourier transform for graph signals is not evident due to the irregularity of the signal domain. By analogy, there exist multiple definitions of Fourier transform [RBTGV19; TGB18]. The one that we are interested in, takes the eigenvalues $\Lambda = \{\lambda_1, \dots, \lambda_n\}$ and the eigenvectors $\mathsf{U} = [\mathbf{u}_1 | \dots | \mathbf{u}_n]$ of the graph Laplacian as the graph frequencies and the graph Fourier basis, respectively. Then the graph Fourier transform of a signal is defined as:

$$\hat{\mathbf{y}} \coloneqq \mathsf{U}^\top \mathbf{y}. \tag{3.3}$$

The vector $\hat{\mathbf{y}}$ contains the Fourier coefficients of the signal corresponding to the different frequency components. One recovers the original signal via the inverse transform $\mathbf{y} = \mathsf{U}\hat{\mathbf{y}}$. Given these definitions, we describe a filtering operation in three steps; i/ first compute the Fourier coefficients as $\hat{\mathbf{y}} = \mathsf{U}^\top \mathbf{y}$, ii/ then filter out these coefficients by computing $\hat{\mathbf{y}}' = \operatorname{diag}(g(\Lambda))\hat{\mathbf{y}}$ where $g : \Lambda \to \mathbb{R}^n$ is the transfer function of the filter, iii/ finally compute the filtered signal $\mathbf{y}' = \mathsf{U}\hat{\mathbf{y}}'$ by going back to the original domain via the inverse transform. We can summarize these three steps in the following expression:

$$\mathbf{y}' = \mathsf{U}^\top \operatorname{diag}(g(\Lambda))\mathsf{U}\mathbf{y}.$$

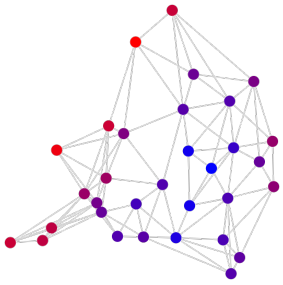
Depending on the characteristic of the transfer function g, this filtering scheme eliminates some components and amplifies others. For example, for the ideal low-pass filters or k-bandlimited filters, which keep only the first k frequency components and eliminate the rest (of the high-frequency components), the transfer function becomes:

$$g_k(\lambda) = \begin{cases} 1 & \text{if } \lambda < \lambda_k, \\ 0 & \text{otherwise.} \end{cases}$$

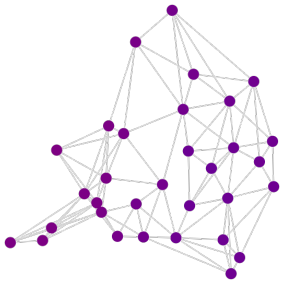
Interestingly, the solution $\hat{\mathbf{x}} = \mathsf{K}\mathbf{y}$ where $\mathsf{K} = q(\mathsf{L} + q\mathsf{I})^{-1}$ can be considered under this graph filtering scheme. This is because L and K share the same eigenvectors and one can write:

$$\mathsf{K} = \mathsf{U} \operatorname{diag}(g(\Lambda))\mathsf{U}^\top \text{ with } g_q(\lambda) = \frac{q}{q+\lambda}.$$

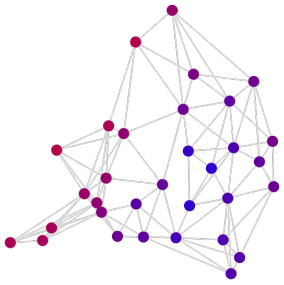
In other words, the solution $\hat{\mathbf{x}}$ is a filtered graph signal with the transfer function $g_q(\lambda) = \frac{q}{q+\lambda}$. Moreover, $g_q(\lambda)$ is a low-pass filter as it diminishes for larger values of the graph frequencies. In Fig. 3.1, we illustrate this behavior on a graph built by temperature sensors over Bretagne/France. Here, the graph signal is the average temperature measurements collected from various locations in Bretagne. We apply the filter $g_q(\lambda)$ on this signal with three different values of q. We observe that



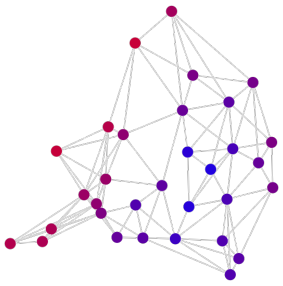
(a) Temperature network over Bretagne/France.



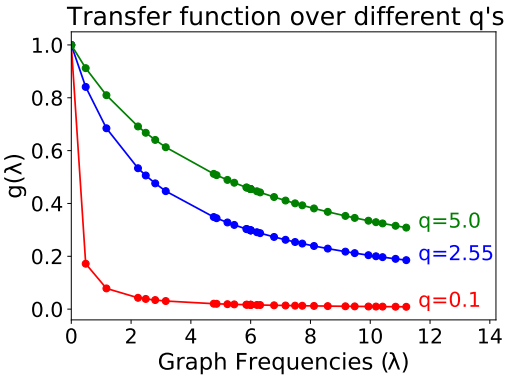
(b) Filtered signal with $q = 0.1$



(c) Filtered signal with $q = 2.55$



(d) Filtered signal with $q = 5.0$



(e) Transfer function of the filter

Fig. 3.1.: An Illustration of the filtering properties of $g_q(\lambda)$ for different q values. In this example, we take the temperature dataset collected in [PV17]. We build a $k = 5$ nearest neighbour graph from the locations of the temperature sensors over Bretagne/France. The signal is the average temperatures measured at each sensor and depicted by the colours (blue is colder).

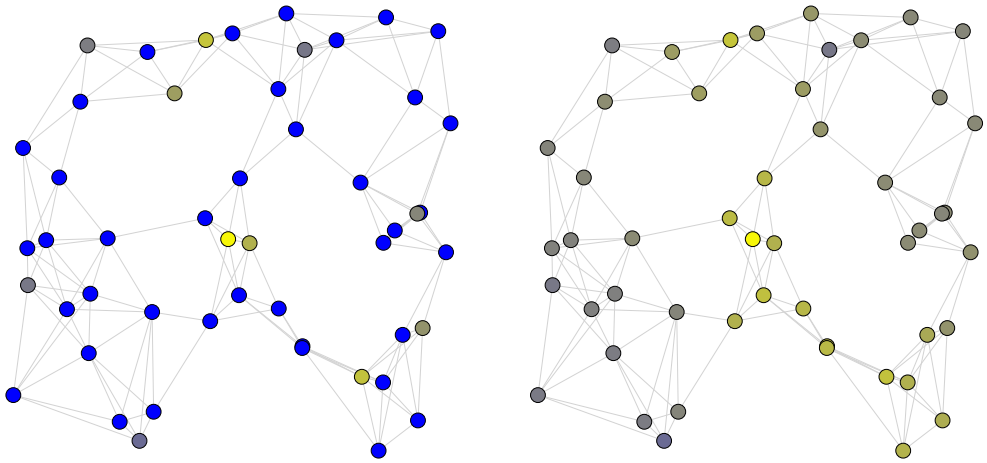
for decreasing values of q, the filter eliminates more and more high frequency components. In turn, the output becomes smoother on the underlying graph.

Dirichlet boundary problem on graphs: In solving differential equations, the Dirichlet boundary condition specifies the values of the solution at a given boundary. As it is a generic constraint, it can be used with any differential system. However, we are interested in the case when it is imposed on Laplace's equation. Laplace's equation is a particular partial differential equation (PDE) in which we seek a multivariate function $u(x_1, \ldots, x_p)$ that satisfies:

In this case, for some subset $V \subseteq \mathcal{V}$ one has $\mathsf{M} \in \mathbb{R}^{n \times |V|}$ with $\mathsf{M}_{i,i} = \mathbb{I}(i \in V)$ and $\mathsf{Q} = \lim_{q \to \infty} q\mathsf{M}\mathsf{M}^\top$.

$$\Delta u = \sum_{i=1}^{p} \frac{\partial^2 u}{\partial x_i^2} = 0,$$

$$\text{s.t. } \forall x_1, \ldots, x_p \in \delta D, \quad u(x_1, \ldots, x_p) = g(x_1, \ldots, x_p).$$



(a) A sensor network with missing signal (in blue)

(b) The signal after completion

Fig. 3.2.: *We complete the missing signal by the solution in Eq.* (3.4).

where Δ is the continuous Laplacian operator, δD is the boundary of the closed region $D \subset \mathbb{R}^p$ and the function g gives the fixed boundary values. The heat diffusion, fluid flow, or electrostatics are only some of the physical phenomena in which this PDE or its variants appear [HKLW12; Mar15].

By analogy, this problem translates to the discrete graph domain. In this case, the subset $V \subset \mathcal{V}$ is considered as the boundary and we seek a solution over $U = \mathcal{V} \setminus V$ of the following linear system:

$$\begin{aligned} &\forall i \in U, \quad \sum_{j \in \mathcal{N}(i)} w(i,j)(x_i - x_j) = 0, \\ &\text{s.t. } \forall i \in V, \quad x_i = g(i). \end{aligned} \tag{3.4}$$

In turn, we have a closed-form solution for this problem:

$$x_i = \begin{cases} ((\mathsf{L}_{U|U})^{-1}\mathsf{L}_{U|V}\mathbf{g})_i, & \text{if } i \in U \\ g(i), & \text{otherwise,} \end{cases} \tag{3.5}$$

where $\mathbf{g} = [g(i)]_{i \in U}$. We illustrate this scheme in Fig. 3.2. In this illustration, we are given a graph with a lot of unlabeled nodes (unlabeled nodes in blue). We propagate the information contained in the labeled nodes by using the solution in Eq. 3.5. This solution is also called the harmonic function. A notable application of harmonic functions is the electrical networks *i.e.* networks of resistors (a circuit element that

has a proportional relation between the potential difference applied to it and the current flowing through it). In these networks/graphs, each resistor is an edge, and each point where two resistors meet is a node. In this abstraction, the conductances of the resistors are denoted by the edge weights. A typical problem is the calculation of the currents flowing through the resistors whenever there is a potential difference between some nodes. This problem boils down to solving the problem in Eq. 3.4. To show this, let us denote the potentials/voltages at each node i by x_i and assume that the potentials in V are given by $g(i)$'s for all $i \in V$. By Ohm's law, the current flowing in edge (i, j) equals to $w(i, j)(x_i - x_j)$. Then, by Kirchoff's current law state that the current sum $\sum_{j \in \mathcal{N}(i)} w(i, j)(x_i - x_j)$ at each node i must be equal to zero. As a result, we recover the problem in Eq. (3.4).

3.2 State-of-the-Art

In these applications and in many more that boil down to this form, the efficient calculation of $\hat{\mathbf{x}}$ is of utmost necessity. The main computational cost is inverting a matrix involving the graph Laplacian L. In the worst-case scenario, the naive computations go through the Gauss-Jordan elimination process that takes $\mathcal{O}(n^3)$ elementary operations. This cost is prohibitive as n increases. This shortcoming is solved by using either direct methods after useful matrix factorizations (*e.g.* Cholesky decomposition) or approximate methods (*e.g.* iterative solvers). The former approaches provide the exact solution by avoiding some cumbersome intermediate operations. They usually yield an efficient computation. However, in the worst case, the number of operations scales up to $\mathcal{O}(n^3)$. Therefore, state-of-the-art is the approximate methods. Among those, the strongest ones are the iterative solvers, notably the conjugate gradient descent algorithm with proper preconditioning. The time complexity of Conjugate Gradient Descent (CGD) scales linearly with the number of non-zero entries in the inverted matrix, which is the number of edges m in our case. More details on the taxonomy of the approximate methods are in Section 2.3.3.

This chapter will present the RSF-based algorithms to approximate $\hat{\mathbf{x}}$. Appealing properties of these estimators are that:

- They illustrate a practical use of elegant links between RSFs and the graph Laplacian.
- They give unbiased estimates of $\hat{\mathbf{x}}$ (*i.e.* their expectation equals to $\hat{\mathbf{x}}$),
- The time complexity of both estimators scale linearly with m,

- The theoretical analysis is tractable.

On the other hand, the intrinsic weakness of these methods is their convergence rate to $\hat{\mathbf{x}}$. They are subject to the Monte Carlo convergence rate $\mathcal{O}(N^{-1/2})$ where N is the number of Monte Carlo samples. Therefore, they require many samples to reach high accuracy. As this issue remains the main limiting factor in all Monte Carlo estimators, the vast literature on Monte Carlo has focused on decreasing the expected error of the estimators. In order to do so, many algorithms have been proposed [Owe13]. The main line of the algorithms proposed suggests using an additional statistic to efficiently eliminate some of the variance of the estimator. We manage to adapt two of them for the RSF-based estimators. We detail these adaptation in Section 3.3.1 and 3.3.2.

3.3 Estimating $\hat{\mathbf{x}}$ via RSFs

Consider the case $V = \mathcal{V}$. Then, one recovers:

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

Now, recall Theorem 2.4.4 which states that the matrix K is the kernel matrix of the DPP of the random roots of Φ_Q. Moreover, it verifies the following identity:

$$\mathbb{P}(r_{\Phi_Q}(i) = j) = \mathsf{K}_{i,j}.$$

We present in this section two unbiased estimators of $\hat{\mathbf{x}}$ that take *their roots* from this beautiful result. Moreover, they easily extend to the case where there are missing measurements *i.e.* $V \subset \mathcal{V}$.

Definition 3.3.1 ($\tilde{\mathbf{x}}$)**.** Let $\mathbf{y} \in \mathbb{R}^n$ be the (complete) measurements over $\mathcal{V}$. Define an estimator $\tilde{\mathbf{x}} = [\tilde{x}_1, \dots, \tilde{x}_n]$ as follows:

$$\forall i \in \mathcal{V}, \quad \tilde{x}_i \coloneqq y_{i_r} \text{ with } i_r = r_{\Phi_Q}(i). \tag{3.6}$$

The expectation and variance analysis is tractable and uncover the link between $\tilde{\mathbf{x}}$ and $\hat{\mathbf{x}}$:

Proposition 3.3.1 (Expectation and Variance of $\tilde{\mathbf{x}}$)**.** *Two main properties of* $\tilde{\mathbf{x}}$ *are that:*

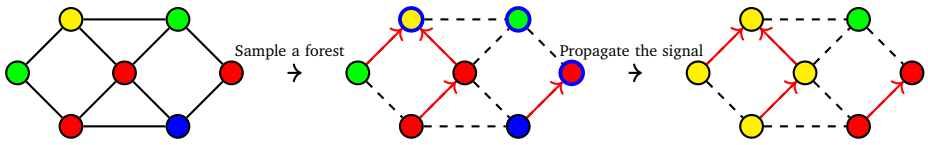


Fig. 3.3.: An illustration of $\tilde{\mathbf{x}}$. Given the graph and the signals (as colours on the nodes), we first sample a rooted spanning forest via Wilson's algorithm. Then we propagate the signal at the roots through the corresponding trees.

i The estimator $\tilde{\mathbf{x}}$ is unbiased for $\hat{\mathbf{x}}$,

ii The weighted sum of the node-wise variance of $\tilde{\mathbf{x}}$ reads:

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

Proof. The proof of (i) is easy. Define a random matrix $\tilde{\mathsf{S}} \coloneqq [\mathbb{I}(r_{\Phi_Q}(i) = j)]_{i,j}$. An immediate result of this definition is:

$$\mathbb{E}[\tilde{\mathsf{S}}] = \mathsf{K}.$$

Noticing $\tilde{\mathbf{x}} = \tilde{\mathsf{S}}\mathbf{y}$ finishes the first part of the proof:

$$\mathbb{E}[\tilde{\mathbf{x}}] = \mathbb{E}[\tilde{\mathsf{S}}]\mathbf{y} = \mathsf{K}\mathbf{y} = \hat{\mathbf{x}}.$$

To prove (ii), we replace the variance with its definition:

$$\begin{aligned}
\sum_{i \in \mathcal{V}} q_i \operatorname{Var}(\tilde{x}_i) &= \sum_{i \in \mathcal{V}} q_i \left(\mathbb{E}[\tilde{x}_i^2] - \mathbb{E}[\tilde{x}_i]^2 \right) \\
&= \sum_{i \in \mathcal{V}} q_i \left(\sum_{j \in \mathcal{V}} \mathbb{P}(r_{\Phi_Q}(i) = j) y_j^2 - \hat{x}_i^2 \right). \\
&= \sum_{i \in \mathcal{V}} q_i \left(\sum_{j \in \mathcal{V}} \mathsf{K}_{i,j} y_j^2 - \hat{x}_i^2 \right).
\end{aligned}$$

Then define $\mathbf{y}^{(2)} = [y_1^2, \ldots, y_n^2]$ and $\mathbf{q} = [q_1, \ldots, q_n]$. Rewriting the summation above, one has:

$$\begin{aligned}
\sum_{i \in \mathcal{V}} q_i \operatorname{Var}(\tilde{x}_i) &= \left(\sum_{i \in \mathcal{V}} q_i (\mathsf{K}\mathbf{y}^{(2)})_i \right) - \left(\sum_{i \in \mathcal{V}} q_i (\mathsf{K}\mathbf{y})_i^2 \right) \\
&= \mathbf{q}^\top \mathsf{K}\mathbf{y}^{(2)} - \mathbf{y}\mathsf{K}^\top \mathsf{Q}\mathsf{K}\mathbf{y}.
\end{aligned}$$

Noticing that $\mathbf{q}$ is the eigenvector of $(\mathsf{Q}+\mathsf{L})^{-1}$ with the eigenvalue 1 finishes the proof:

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

□

Implementing $\tilde{\mathbf{x}}$ is straightforward; first sample a spanning forest via Wilson's algorithm, then within each tree of the forest, propagate the signal value of the root throughout the tree (See Fig. 3.3). In order to generate N samples of $\tilde{\mathbf{x}}$, we need to sample N spanning forests via Wilson's algorithm. As this is the heaviest computation, one can calculate the time complexity of the proposed method in terms of the total number of steps taken by the random walks in Wilson's algorithm to sample N forests. Let us denote this number K_N. By using Eq. (2.21), one has:

$$\mathbb{E}[K_N] = N\mathbb{E}[K_1] = N\operatorname{tr}(\mathsf{Q}+\mathsf{L})^{-1}(\mathsf{D}+\mathsf{Q}).$$

Notice that if $\mathbf{y} = c\mathbf{1}$, the variance of $\tilde{\mathbf{x}}$ is zero. Since K is a smoothing operator on the graph, an intuitive thinking is that there is *nothing to smooth out* in $\mathbf{y}$. If there is some variation in $\mathbf{y}$, the variance of $\tilde{\mathbf{x}}$ becomes non-zero. In fact it is maximized when $\mathbf{y}$ equals to the right eigenvector of K that corresponds to the smallest eigenvalue. In this case, the quadratic form $\mathbf{y}^\top\mathsf{K}^\top\mathsf{Q}\mathsf{K}\mathbf{y}$ is minimized and the variance is maximized. An intuition for a such vector can be taken from the particular case $\mathsf{Q} = q\mathsf{I}$. As previously discussed, this is the case of graph signal filtering by using the Laplacian eigenvectors as the Fourier basis. Then the eigenvector discussed is $\mathbf{u}_n$ the eigenvector of L with the largest eigenvalue λ_n or the highest graph frequency. In other words, the vector which has the highest variation on the graph results in the highest variance in $\tilde{\mathbf{x}}$ (Also see Fig. 3.4).

It is possible to adapt $\tilde{\mathbf{x}}$ when we only have measurements over a subset $V \subset \mathcal{V}$. In this case, $\mathsf{M} \in \mathbb{R}^{n\times p}$ takes a form of a rectangular selection matrix. Let $\mathbf{y}_V = \mathsf{M}\mathbf{y} = [\mathbf{y}^\top, \mathbf{0}^\top] \in \mathbb{R}^n$ Then it is possible to write $\hat{\mathbf{x}}$ in the following form:

$$\hat{\mathbf{x}} = (\mathsf{Q}+\mathsf{L})^{-1}\mathsf{Q}\mathbf{y}_V.$$

Then, we can design $\tilde{\mathbf{x}}$ to estimate this inversion. In this configuration, we sample Φ_Q by setting:

$$q_i = \begin{cases} \mathsf{Q}_{i,i}, & \text{if } i \in V, \\ 0, & \text{otherwise.} \end{cases}$$

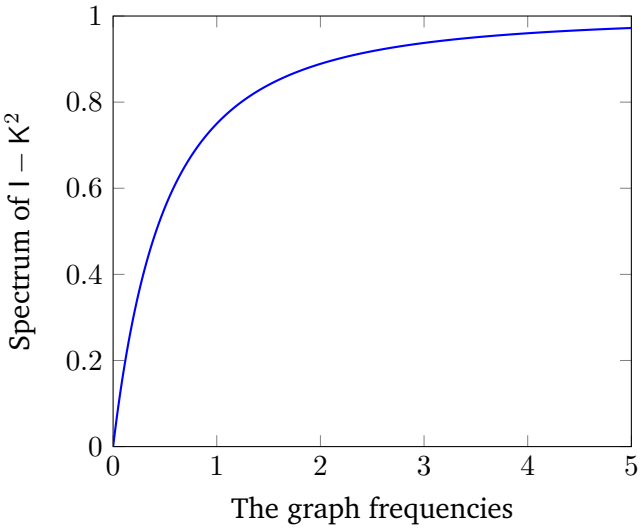


Fig. 3.4.: *In this example, we consider the case of graph signal filtering with $q = 1$. The variance of $\tilde{\mathbf{x}}$ then becomes $\mathbf{y}^\top(\mathsf{I}-\mathsf{K}^2)\mathbf{y}$. We plot the spectrum of the matrix $(\mathsf{I}-\mathsf{K}^2)$ w.r.t. the graph Laplacian eigenvalues/graph frequencies. As a filter, $(\mathsf{I}-\mathsf{K}^2)$ acts like a high-pass filter. Thus the quadratic formula $\mathbf{y}^\top(\mathsf{I}-\mathsf{K}^2)\mathbf{y}$ will take higher values when $\mathbf{y}$ is a highly varying signal on the graph.*

and we calculate $\tilde{\mathbf{x}}$ over these forests. Notice that the roots of the forests generated by this configuration are always a subset of V. Therefore, the propagated values in $\tilde{\mathbf{x}}$ are always the known signal measurements. Indeed, $\tilde{\mathbf{x}}$ efficiently estimates $\hat{\mathbf{x}}$. However, there is still a big room for improvement without compromising the run-time efficiency. In our case, we find techniques that yield significant improvements in the literature on variance reduction techniques [Owe13]. Besides their cheap cost, these techniques give us a way to leverage our additional knowledge embedded in the nature of the RSFs (*e.g.* certain statistics) to substantially decrease the expected error/variance of $\tilde{\mathbf{x}}$. In the following, we delve into the application of these VR techniques, yielding two new estimators for $\hat{\mathbf{x}}$.

3.3.1 Variance Reduction via Conditional Monte Carlo

Let X be a random variable with an unknown expectation $\mathbb{E}[X] = \mu_X$. The classical Monte Carlo algorithm would take N samples of X and compute the empirical mean:

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i.$$

Now assume that there exists another random variable Y for which the conditional expectation $\mathbb{E}[X|Y]$ is easy to calculate. The conditional Monte Carlo technique also

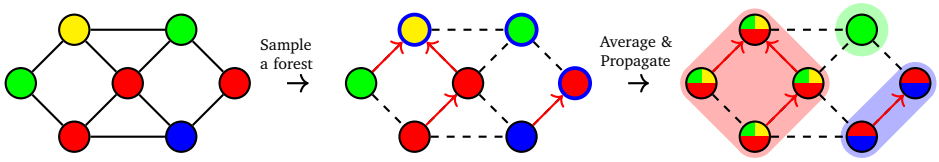


Fig. 3.5.: An illustration of $\bar{\mathbf{x}}$. We follow the same first step with $\tilde{\mathbf{x}}$. Then we propagate the (weighted) average signal within each tree. We assume $q_1 = \cdots = q_n = q$.

called Rao-Blackwellization [Bla47; Rao45], allows us to use our knowledge on Y to reduce the error of the naive estimation on μ_X. In doing so, it defines a new estimator as follows:

$$\bar{Z} \coloneqq \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}[X_i|Y].$$

As $\mathbb{E}[\mathbb{E}[X|Y]] = \mathbb{E}[X]$, $\bar{Z}$ is an unbiased estimator. Moreover, it has a reduced variance due to the law of total variance:

$$\mathrm{Var}(X) = \mathbb{E}[\mathrm{Var}(X|Y)] + \mathrm{Var}(\mathbb{E}[X|Y]). \tag{3.7}$$

In our case, we find a simple way of deploying this idea for $\tilde{\mathbf{x}}$:

Definition 3.3.2 ($\bar{\mathbf{x}}$)**.** Given a fixed partition $\mathcal{P}$, we define the conditional expectation $\bar{\mathbf{x}} = [\bar{x}_1, \ldots, \bar{x}_n]^\top \in \mathbb{R}^n$ as follows:

$$\begin{aligned} \forall i \in \mathcal{V}, \quad \bar{x}_i &\coloneqq \mathbb{E}[\tilde{x}_i | \pi(\Phi_Q) = \mathcal{P}] \\ &= \sum_{j=1}^{n} y_j \mathbb{P}(r_{\Phi_Q}(i) = j | \pi(\Phi) = \mathcal{P}) \\ &= \sum_{j=1}^{n} \frac{y_j q_j}{\sum_{k \in \mathcal{V}_{t(\mathcal{P},i)}} q_k} = \frac{\sum_{j \in \mathcal{V}_{t(\mathcal{P},i)}} q_j y_j}{\sum_{k \in \mathcal{V}_{t(\mathcal{P},i)}} q_k}. \end{aligned} \tag{3.8}$$

In a practical sense, after sampling Φ_Q, the new estimator $\bar{\mathbf{x}}$ is simply the empirical weighted mean within each tree, thus it is easy to implement. Note that for $q_1 = \cdots = q_n = q$, it boils down to the simple averaging. This is illustrated in Fig. 3.5. After sampling a spanning forest, one only needs to compute local averages and propagate them within trees for calculating $\bar{\mathbf{x}}$. On the theory side, one can easily ensure that $\bar{\mathbf{x}}$ is unbiased and has a lower variance than $\tilde{\mathbf{x}}$ by Eq. 3.7. Moreover, we have a closed form expression for the variance of the new estimator:

Proposition 3.3.2 (Variance of $\bar{\mathbf{x}}$). *The weighted sum of the node-wise variance of $\bar{\mathbf{x}}$ equals to:*

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

Proof. We start by replacing the variance by its definition:

$$\begin{aligned}\mathbb{E}[||\bar{\mathbf{x}} - \hat{\mathbf{x}}||_{\mathsf{Q}}^2] = \sum_{i\in\mathcal{V}} q_i \operatorname{Var}(\bar{x}_i) &= \sum_{i\in\mathcal{V}} q_i \left(\mathbb{E}[\bar{x}_i^2] - \mathbb{E}[\bar{x}_i]^2\right), \\ &= \sum_{i\in\mathcal{V}} q_i\mathbb{E}[\bar{x}_i^2] - ||\hat{\mathbf{x}}||_{\mathsf{Q}}^2 \\ &= \sum_{i\in\mathcal{V}} q_i\mathbb{E}[\bar{x}_i^2] - \mathbf{y}^\top\mathsf{K}^\top\mathsf{QK}\mathbf{y}.\end{aligned} \tag{3.9}$$

As in Prop. 3.3.1, we define a random matrix $\bar{\mathsf{S}}$ that verifies $\bar{\mathbf{x}} = \bar{\mathsf{S}}\mathbf{y}$. Here $\bar{\mathsf{S}}$ can be considered as a weighted averaging operator. Then, rewriting $\bar{\mathbf{x}} = \bar{\mathsf{S}}\mathbf{y}$, one has:

$$\begin{aligned}\sum_{i\in\mathcal{V}} q_i \operatorname{Var}(\bar{x}_i) &= \sum_{i\in\mathcal{V}} q_i\mathbb{E}[(\bar{\mathsf{S}}\mathbf{y})_i^2] - \mathbf{y}^\top\mathsf{K}^\top\mathsf{QK}\mathbf{y}, \\ &= (\mathbb{E}[||\bar{\mathsf{S}}\mathbf{y}||_{\mathsf{Q}}^2]) - \mathbf{y}^\top\mathsf{K}^\top\mathsf{QK}\mathbf{y}.\end{aligned} \tag{3.10}$$

Here we invoke the identity[a] $\bar{\mathsf{S}}\mathsf{Q}\bar{\mathsf{S}} = \mathsf{Q}\bar{\mathsf{S}}$:

$$\begin{aligned}\sum_{i\in\mathcal{V}} q_i \operatorname{Var}(\bar{x}_i) &= \mathbb{E}[\mathbf{y}^\top\bar{\mathsf{S}}^\top\mathsf{Q}\bar{\mathsf{S}}\mathbf{y}] - \mathbf{y}^\top\mathsf{K}^\top\mathsf{QK}\mathbf{y}, \\ &= \mathbf{y}^\top\mathbb{E}[\mathsf{Q}\bar{\mathsf{S}}]\mathbf{y} - \mathbf{y}^\top\mathsf{K}^\top\mathsf{QK}\mathbf{y}, \\ &= \mathbf{y}^\top(\mathsf{QK} - \mathsf{K}^\top\mathsf{QK})\mathbf{y}.\end{aligned} \tag{3.11}$$

□

[a] This result might not be seem straight-forward but easy to check by calculating the results of the matrix products

Similar to $\tilde{\mathbf{x}}$, the variance of $\bar{\mathbf{x}}$ is zero for a constant measurement vector $\mathbf{y} = c\mathbf{1}$ and it becomes non-zero when $\mathbf{y}$ contains some variation between nodes. However, $\bar{\mathbf{x}}$ has a significant difference for highly varying signals. This is easy to observe in case of graph signal filtering which is illustrated in Fig. 3.6.

Overall, $\bar{\mathbf{x}}$ gives theoretical guarantees for the improvement in estimating $\hat{\mathbf{x}}$ while it does not compromise the run-time efficiency. One can calculate the local weighted averages by a single pass over the signal $\mathbf{y}$, yielding $\mathcal{O}(n)$ time complexity, which is dominated by the cost of sampling a forest. In practice, we observe over various graphs that $\bar{\mathbf{x}}$ brings substantial performance improvement. Nevertheless, we

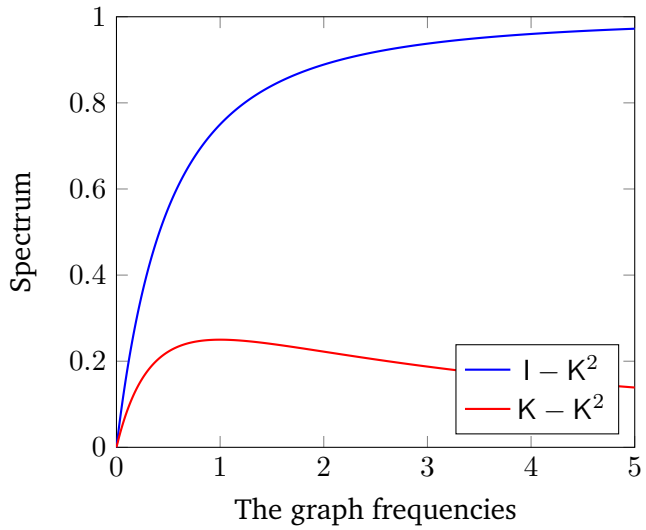


Fig. 3.6.: *In this example, we again consider the case of graph signal filtering with* $q = 1$. *The variance of* $\bar{\mathbf{x}}$ *then becomes* $\mathbf{y}^\top(\mathsf{K} - \mathsf{K}^2)\mathbf{y}$. *We plot the spectrum of the matrix* $(\mathsf{K} - \mathsf{K}^2)$ *w.r.t. the graph Laplacian eigenvalues/graph frequencies and compare it with the case of* $\tilde{\mathbf{x}}$. *As a filter,* $(\mathsf{K} - \mathsf{K}^2)$ *acts like a band-pass filter i.e. its transfer function only takes high values for a particular range of frequencies, called the pass-band, that are neither high, nor low. Thus the quadratic formula* $\mathbf{y}^\top(\mathsf{K} - \mathsf{K}^2)\mathbf{y}$ *will take higher values whenever* $\mathbf{y}$ *is mostly composed by the frequency components that are in the pass-band. Also note that in comparison to* $\tilde{\mathbf{x}}$, $\bar{\mathbf{x}}$ *is always better as illustrated.*

are capable of even going beyond this performance by applying another variance reduction technique. We detail this technique in the next section.

3.3.2 Variance Reduction via Control Variates

Consider again the problem of estimating μ_X from the samples of the random variable X. The control variate (CV) method proposes to take another random variable Y, also called control variate, that has:

- a known or easy to calculate expectation μ_Y,
- non-zero correlation with the estimated quantity *i.e.* $\mathrm{Cov}(X, Y) \neq 0$.

Given this ingredient, we define another estimator as follows:

$$Z \coloneqq X - \alpha(Y - \mu_Y),$$

where $\alpha \in \mathbb{R}$ is a hyper-parameter. An immediate result is that Z is an unbiased estimator of μ_X regardless of the value of α, as $\mathbb{E}[Z] = \mathbb{E}[X] - \alpha(\mathbb{E}[Y] - \mu_Y) = \mu_X$. However, the variance analysis is closely related to the value of α. Let us write the variance of Z in terms of the variance of X, Y and α:

$$\begin{aligned} \mathrm{Var}(Z) &= \mathrm{Var}(X) + \mathrm{Var}(\alpha Y) - 2\,\mathrm{Cov}(X, \alpha Y), \\ &= \mathrm{Var}(X) + \alpha^2\,\mathrm{Var}(Y) - 2\alpha\,\mathrm{Cov}(X, Y). \end{aligned} \tag{3.12}$$

To ensure $\mathrm{Var}(Z) < \mathrm{Var}(X)$, one has to verify that:

$$\alpha^2\,\mathrm{Var}(Y) - 2\alpha\,\mathrm{Cov}(X, Y) < 0$$

Assuming $\alpha > 0$, one obtains:

$$\alpha < \frac{2\,\mathrm{Cov}(X, Y)}{\mathrm{Var}(Y)}.$$

This bound ensures variance reduction. Yet, we can still go beyond this analysis and derive the optimum value for α. To do so, we take the derivative of $\mathrm{Var}(Z)$ with respect to α and set it to zero.

$$\frac{\partial \mathrm{Var}(Z)}{\partial \alpha} = 2\alpha\,\mathrm{Var}(Y) - 2\,\mathrm{Cov}(X, Y) = 0.$$

Solving this equation gives:

$$\alpha^\star = \frac{\mathrm{Cov}(X,Y)}{\mathrm{Var}(Y)}. \tag{3.13}$$

Plugging $\alpha^\star$, we get a reduced variance as follows:

$$\mathrm{Var}(Z) = \mathrm{Var}(X) - \frac{\mathrm{Cov}(X,Y)^2}{\mathrm{Var}(Y)}.$$

In many applications, $\alpha^\star$ is not necessarily accessible as it is a ratio of two variances that are not often available. A common practice to overcome this issue is to estimate it from the samples of X and Y:

$$\hat{\alpha} = \frac{\widehat{\mathrm{Cov}}(X,Y)}{\widehat{\mathrm{Var}}(Y)},$$

where $\widehat{\mathrm{Cov}}(X,Y)$ and $\widehat{\mathrm{Var}}(Y)$ are sample variance formulas. In doing so, one can either use extra samples of X and Y to compute $\hat{\alpha}$ or use the same samples for computing both $\hat{\alpha}$ and Z. The former option gives an unbiased estimator at the cost of having extra samples. In the latter, the resulting estimator might have a bias as $\hat{\alpha}$ and Z are calculated from the same samples. However, in [Owe13], the author discusses that the bias diminishes faster than the Monte Carlo error w.r.t. the number of samples, *i.e.* $\mathcal{O}(N^{-1})$ compared to $\mathcal{O}(N^{-1/2})$. Therefore, as N grows, the bias becomes smaller compared to the Monte Carlo approximation error.

We find a simple adaptation of the control variate technique on the forest-based estimators $\tilde{\mathbf{x}}$ and $\bar{\mathbf{x}}$. This adaptation is mainly inspired by the gradient descent algorithm for solving Prob. 3.1.1. Let us briefly revisit this algorithm:

Definition 3.3.3 (Gradient Descent)**.** Consider the following optimization problem:

$$\begin{aligned} \hat{\mathbf{x}} &= \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^n} F(\mathbf{x}) \\ &\text{s.t. } F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathsf{K}\mathbf{x} - \mathbf{x}^\top \mathbf{y}. \end{aligned}$$

Notice that the solution of this problem is *also* $\hat{\mathbf{x}} = \mathsf{K}\mathbf{y}$. The gradient descent algorithm suggests iteratively approximating the solution. It picks an arbitrary initial vector $\mathbf{x}_0 \in \mathbb{R}^n$ and updates it as follows:

$$\mathbf{x}_{k+1} \coloneqq \mathbf{x}_k - \alpha \nabla F(\mathbf{x}_k), \quad k = 1, 2, \dots$$

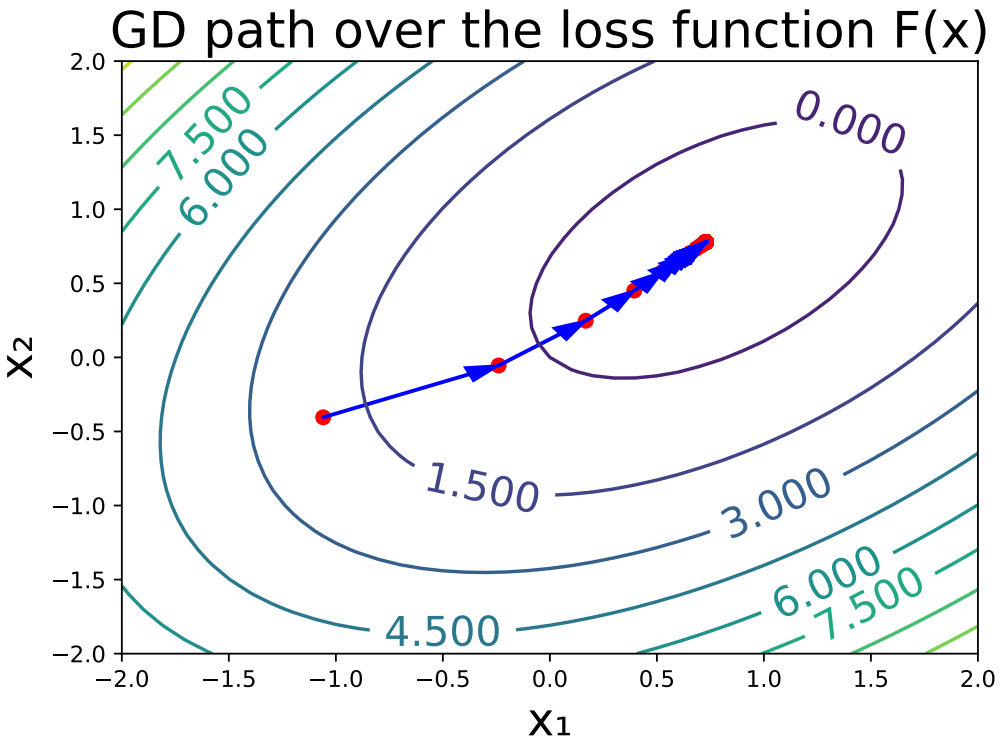


Fig. 3.7.: We plot the loss function $F(\mathbf{x})$ for $\mathbf{x} = [x_1, x_2]^\top \in \mathbb{R}^2$ and mark the path followed by the gradient descent algorithm to reach the point that minimizes $F(\mathbf{x})$ ($q = 1.0$ and $\alpha = 0.001$).

where $\nabla F(\mathbf{x}_k) = (\mathsf{K}^{-1}\mathbf{x}_k - \mathbf{y})$ and $\alpha \in \mathbb{R}$. This iteration updates the solution by taking a step in the opposite direction of the gradient of the loss function $F(\mathbf{x})$ which is the steepest descent to the solution (See Fig. 3.7).

We present another unbiased estimator by proposing to use the gradient descent update as the control variate.

Definition 3.3.4. Given the forest estimate $\bar{\mathbf{x}}$, we apply a single step of gradient descent algorithm, which yields the following estimator:

$$\bar{\mathbf{z}} \coloneqq \bar{\mathbf{x}} - \alpha(\mathsf{K}^{-1}\bar{\mathbf{x}} - \mathbf{y}).$$

The unbiasedness of $\bar{\mathbf{z}}$ is easy to see as $\mathbb{E}[\bar{\mathbf{z}}] = \mathbb{E}[\bar{\mathbf{x}}] - \alpha(\mathsf{K}^{-1}\mathbb{E}[\bar{\mathbf{x}}] - \mathbf{y}) = \mathsf{K}\mathbf{y}$. The variance highly depends on the choice of α as discussed before. From Eq. (3.13), the optimal choice is:

$$\alpha^\star = \frac{\operatorname{tr}(\operatorname{Cov}(\mathsf{K}^{-1}\bar{\mathbf{x}}, \bar{\mathbf{x}}))}{\operatorname{tr}(\operatorname{Cov}(\mathsf{K}^{-1}\bar{\mathbf{x}}))} = \frac{\operatorname{tr}(\mathsf{K}^{-1}\operatorname{Cov}(\bar{\mathbf{x}}))}{\operatorname{tr}(\mathsf{K}^{-2}\operatorname{Cov}(\bar{\mathbf{x}}))},$$

where $\mathrm{Cov}(\bar{\mathbf{x}}) = [\mathrm{Cov}(\bar{x}_i, \bar{x}_j)]_{i,j} \in \mathbb{R}^{n\times n}$ is the auto-covariance matrix of $\bar{\mathbf{x}}$. Computing $\alpha^\star$ requires computing $\mathrm{Cov}(\bar{\mathbf{x}})$ which is not available without expensive operations. On the other hand, one can calculate $\hat{\alpha}$ from the samples of $\bar{\mathbf{x}}$ much more cheaply. In numerical experiments, this choice of α often reduces the variance near optimal despite the aforementioned bias. Yet, one may still want to ensure the variance reduction regardless of how much it is reduced. When this is the case, we suggest using the classical arguments of gradient descent literature for the step size. By doing so, we find a cheap constant for α that guarantees variance reduction:

Proposition 3.3.3. *Define the maximum degree of a graph as* $d_{max} \coloneqq \max_{i\in\mathcal{V}}\{d_i\}$. *Fixing* $\alpha = \frac{2q}{q+2d_{max}}$ *guarantees that:*

$$||\bar{\mathbf{z}} - \hat{\mathbf{x}}||_2 \leq ||\bar{\mathbf{x}} - \hat{\mathbf{x}}||_2. \tag{3.14}$$

Proof. The inequality in (3.14) yields a step size such that the result gets closer to the solution at every step. Expanding the definition of $\bar{\mathbf{z}}$:

$$\begin{aligned} ||\bar{\mathbf{x}} - \hat{\mathbf{x}} - \alpha(\mathsf{K}^{-1}\bar{\mathbf{x}} - \mathbf{y})||_2 &\leq ||\bar{\mathbf{x}} - \hat{\mathbf{x}}||_2, \\ ||(\mathsf{I} - \alpha\mathsf{K}^{-1})(\bar{\mathbf{x}} - \hat{\mathbf{x}})||_2 &\leq ||\bar{\mathbf{x}} - \hat{\mathbf{x}}||_2. \end{aligned} \tag{3.15}$$

This inequality is satisfied if the eigenvalues $\mu_1, \ldots, \mu_n$ of the matrix $(\mathsf{I} - \alpha\mathsf{K}^{-1})$ verifies:

$$|\mu_i| \leq 1, \quad \forall i \in \{1, \ldots, n\},$$

In terms of the graph Laplacian eigenvalues, one has:

$$\begin{aligned} \forall i \in \{1, \ldots, n\}, \quad |1 - \alpha\frac{\lambda_i}{q} - \alpha| &\leq 1, \\ 0 \leq \alpha &\leq \frac{2q}{\lambda_i + q}. \\ 0 \leq \alpha &\leq \frac{2q}{\lambda_n + q}. \end{aligned}$$

Recalling $\lambda_n \leq 2d_{max}$ [Van10] finishes the proof. □

Indeed, this choice of α is cheaper compared to $\hat{\alpha}$ and introduces no biases. However, it might provide a much smaller variance reduction compared to optimal in certain cases. This is mostly because $\lambda_n \leq 2d_{max}$ is not necessarily a tight bound, and the constraint in (3.14) might be too excessive for taking a safe step, and thus it might slow down the algorithm. In order to clearly illustrate these differences, we consider the following case study:

Different Choices of α

In this example, we consider the signal denoising problem on two graphs generated by two random models, namely random k-regular and Barabasi-Albert. We are motivated to choose these models as they have opposite characteristics in terms of their degree distribution. While the random k-regular model always generates graphs in which all degrees are fixed to k, the graphs generated by the Barabasi-Albert model typically have spread degree distributions. For simplicity, we take a random signal generated from $\mathcal{N}(\mathbf{0}, \mathrm{I})$. We apply smoothing via the direct computation (to compute $\hat{\mathbf{x}}$) and the forest-based estimators $\bar{\mathbf{x}}$ and $\bar{\mathbf{z}}$. In Fig. 3.8, we plot the mean squared error of these estimate with $\hat{\mathbf{x}}$ for varying values of α.

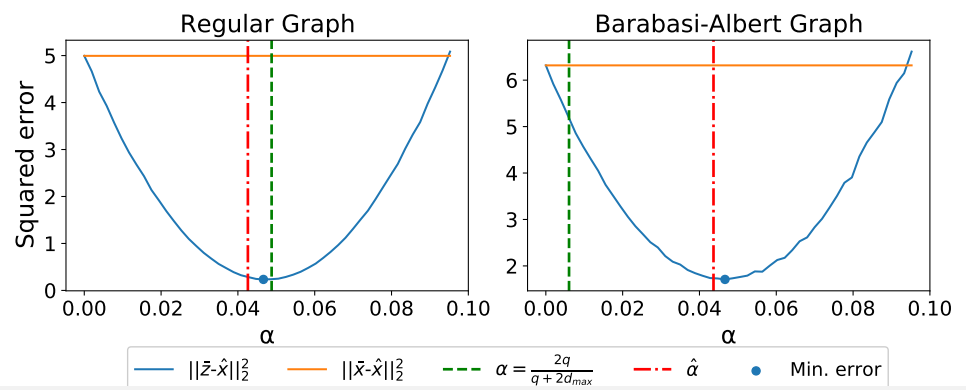


Fig. 3.8.: Empirical mean squared error of $\bar{\mathbf{x}}$ (orange horizontal line) and $\bar{\mathbf{z}}$ (blue parabola) w.r.t α on two graphs generated by random models. On the left is a random regular graph with $n = 1000$ and $m = 10000$. On the right is a Barabasi-Albert model with parameter $k = 10$ resulting in $n = 1000$ and $m = 9900$. The green (resp. red) vertical dashed line shows $\alpha = \frac{2q}{q+2d_{max}}$ (resp. the estimated $\hat{\alpha}$ from the samples). The blue dot represents the best possible variance reduction obtained for $\alpha = \alpha^\star$. The number of Monte Carlo samples is set to $N = 10$, and the error results are averaged over 200 realizations. The signal is a random vector generated from $\mathcal{N}(\mathbf{0}, \mathrm{I})$.

In these plots, we focus on three points that correspond to $\alpha^\star$ (the optimal solution), $\hat{\alpha}$ (the estimate from samples) and $\alpha = \frac{2q}{q+2d_{max}}$ (the constant from Prop. 3.3.3). In the case of a regular graph, both options for α give a very close performance to the optimal choice $\alpha^\star$. This is because $\lambda_n \leq 2d_{max}$ in a regular graph in this case gives a tight upper bound thus $\alpha = \frac{2q}{q+2d_{max}}$ is very close to the optimal choice. This drastically changes when we switch to a highly irregular graph. In this case, $\hat{\alpha}$ continues to give a performance next to the optimal, whereas choosing $\alpha = \frac{2q}{q+2d_{max}}$ becomes too excessive for reducing the error.

This covers our analysis of variance reduction techniques on the forest-based estimators. Indeed, the variance reduction literature is much broader than these two techniques. We observe two main issues in bringing other methods to play; the first and fundamental issue is that it is not often obvious how to adapt these methods efficiently for our particular case. The variance reduction methods often require some prior knowledge on the estimated quantity that is not necessarily available in our case. The second is that even if the necessary conditions for applying a VR method are satisfied, anticipating if it would substantially decrease the variance or not is another question. This requires a good understanding of the random variable and its distribution. In particular, we need to know how $\bar{\mathbf{x}}$ (or other forest estimates) varies with different samples of Φ_Q, which is not straightforward to grasp. In the case of the proposed methods, we are able to overcome these issues. Yet it may be still possible to go beyond with other methods as long as these issues are solved.

In the following sections, we compare the proposed methods with the State-Of-The-Art (SOTA) algorithms in a graph signal denoising setup. Then, we will take a look at possible use cases of the proposed estimators within certain machine learning and optimization applications. In particular, we will look into certain cases where the computation $\mathsf{K}\mathbf{y}$ is needed, and the expensive direct computation can be avoided by the forest-based estimators.

3.3.3 Empirical Comparisons with SOTA Algorithms

In this section, we present our empirical comparisons of the forest-based estimators $\bar{\mathbf{x}}$ and $\bar{\mathbf{z}}$ with the state-of-the-art algorithms, namely Chebyshev polynomial approximation (Pol) [SNFOV13] and the conjugate gradient (CG) method (with and without preconditioning) [Saa03] in approximating the exact solution $\hat{\mathbf{x}}$. The experimentation setup we use is heavily adapted from our journal paper [PABT21b].

3.3.3.1. Experimentation Setup

Datasets. We run our experiments over three benchmark datasets, namely Cora, Citeseer and Pubmed. The following table summarizes their properties: The first two datasets Cora and Citeseer are not connected graphs. Therefore, in those graphs, we only keep the largest connected component and eliminate the rest.

Graph signals. In this experimentation, we consider the graph signal denoising problem over various graphs, in which we are given the noisy measurements $\mathbf{y} = \mathbf{x} + \epsilon$

Tab. 3.1.: Benchmark Graph datasets

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

with $\epsilon \in \mathcal{N}(\mathbf{0}, \sigma^2)$ and the underlying graph. By solving the problem in (3.1), we compute the denoised signal as follows:

$$\hat{\mathbf{x}} = \mathsf{K}\mathbf{y} \text{ with } \mathsf{K} = q(q\mathsf{I} + \mathsf{L})^{-1} \tag{3.16}$$

We assume the original signal $\mathbf{x}$ is a k-bandlimited signal which verifies:

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

where the $\mathbf{u}_i$'s are the lowest k graph Fourier basis and the α_i's are the lowest k graph Fourier coefficients of the signal. In the experiments, we artificially generate such signals by generating α_i's at random. Then, we add Gaussian noise in order to obtain the noisy measurements $\mathbf{y}$. In the whole experimentation, we set k to 50, 50 and 500 for Cora, Citeseer and Pubmed datasets, respectively. The noise variance σ is adjusted per signal such that the signal-to-noise ratio (SNR) of the noisy signal equals to 2.

Parameter tuning. Over all graphs and graph signal realizations, we set the parameter q to the value that gives the best denoising performance (that is, to the value of q minimizing $||\mathbf{x} - \hat{\mathbf{x}}||_2$).

Performance Metrics. We compare all the algorithms by looking at two performance metrics. Given the solution by an algorithm $\mathbf{x}^\star$, we look at the approximation error $||\hat{\mathbf{x}} - \mathbf{x}^\star||_2$ and the reconstruction error $||\mathbf{x} - \mathbf{x}^\star||_2$. While the former measures the quality of the approximation to $\hat{\mathbf{x}}$, the latter gives the denoising performance of the algorithm.

Experimentation Procedure. The procedure we follow in the experiments is threefold:

- Generate a noisy signal $\mathbf{y}$,
- Tune the hyperparameter q by a grid-search,
- Run the algorithms and measure the run-time, approximation and reconstruction error.

We note that all the experiments are implemented in Julia programming language and run in a single thread of a laptop.

3.3.3.2. Results

As aforementioned, the algorithms in comparisons are: the RSF based estimators $\bar{\mathbf{x}}$ and $\bar{\mathbf{z}}$, Chebyshev polynomial approximation and conjugate gradient [1] method with and without preconditioning. In the preconditioned case, we use the Algebraic Multigrid (AMG) algorithm [2] as the preconditioner. In all of these algorithms, there is an iteration parameter. This parameter typically adjust the trade-off between the run-time and the approximation error of the algorithm. These parameters are:

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

During the experiments, we vary these parameters in order to observe the performance of the algorithms over changing run-time. For each algorithm and each graph, it is swept through 17 logarithmically spaced values between 1 and 100: $\{1, 2, \ldots, 62, 78, 100\}$, corresponding to the 17 plotted markers forming each curve.

In Fig. 3.9, we present our results over 3 benchmark graph datasets. From the approximation error results, one can see that the deterministic methods CG and Chebyshev polynomial approximation converges very quickly to the exact solution whereas the forest estimators are stuck with a Monte Carlo convergence rate (which is linear in log-scale). Therefore, the RSF-based methods cannot compete with the deterministic methods for obtaining a low approximation error. On the other hand, from the results on the reconstruction error, we observe that the forest-based estimators are comparable with the other methods. In other words, it is not necessary to have a good approximation in order to get a good reconstruction of the signal. This is because there is no point to compute an inaccurate quantity (in this case, it is $\hat{\mathbf{x}}$) with a high accuracy (See [Bot12]). In addition, we can see by both performance metrics (more clearly in reconstruction error), $\bar{\mathbf{z}}$ is superior to $\bar{\mathbf{x}}$. The control variate estimator $\bar{\mathbf{z}}$ achieves better approximation and reconstruction without taking too much additional time.

[1] Julia implementation https://julialinearalgebra.github.io/IterativeSolvers.jl/dev/
[2] Julia implementation https://github.com/JuliaLinearAlgebra/AlgebraicMultigrid.jl

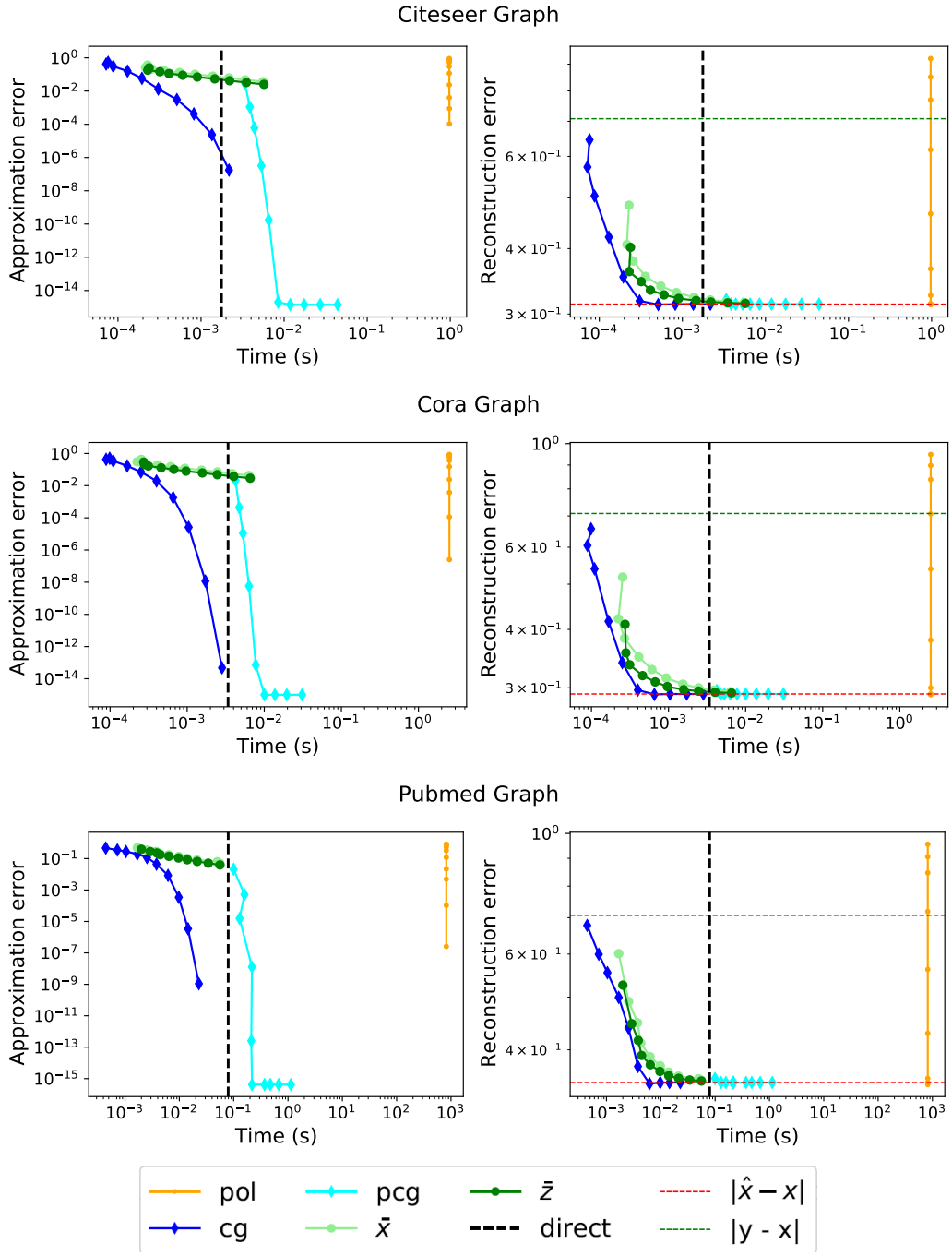
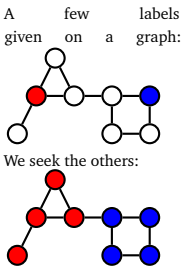


Fig. 3.9.: Approximation and reconstruction error of the algorithms CG (blue and cyan), polynomial approximation (orange) and the forest estimators $\bar{\mathbf{x}}$ (light green) and $\bar{\mathbf{z}}$ (dark green) for solving graph signal denoising problem. The dark dashed line indicates the time taken by the direct solution (backslash operator in Julia). In the reconstruction error plots, we also add the error of the initial measurements $\mathbf{y}$ in green dashed line and the error of the exact solution $\hat{\mathbf{x}}$ in red line.

3.4 Several Use Cases of the RSF-based Estimators

The scope of Eq. (3.1) is not limited to the signal denoising problem. We already know that solving this optimization problem under certain constraints corresponds to solving the Dirichlet boundary problem and semi-supervised learning on graphs. On top of that, we encounter this formulation in solving some graph-related optimization problems. In these cases, solving these problems requires repeatedly solving Prob. (3.1.1) which leaves us with the expensive computation of $\hat{\mathbf{x}}$ at every step. In the following, we show how forest-based estimators can be used in order to circumvent this issue. We conclude each section by giving some illustrations of the forest estimators.

3.4.1 Semi-supervised Learning on Graphs



We consider the node classification problem on graphs. Given a few label information over vertices, the main goal is to correctly infer the labels for the other nodes by using the graph structure. Among many possible formulations and solutions, a well-known baseline algorithm is due to the semi-supervised learning algorithm proposed in [Zhu05] and later unified under Prob. 3.1.1 by [AMGS12]. A formal definition of the problem and the algorithm is as follows:

Problem 3.4.1 (Semi-supervised learning for node classification)**.** *Let us consider a node classification problem over K classes. We denote the vertices with the known labels by $V \subset \mathcal{V}$. In a node classification problem, we typically have $|V| \ll n$ as we have a few labeled nodes and define a label encoding as follows:*

$$\forall i \in \mathcal{V}, k \in \{1, \dots, K\}, \quad \mathsf{Y}_{i,k} = \begin{cases} 1 & \text{, if node } i \text{ belongs to class } k \\ 0 & \text{, otherwise} \end{cases}$$

Given the matrix $\mathsf{Y} \in \mathbb{R}^{n \times K}$, the main goal is to infer labels for the vertices in $\mathcal{V} \setminus V$. In a very classical setup, we evaluate a function $F : \mathcal{V} \times \{1, \dots, K\} \to \mathbb{R}$ (also called classification function) such that for each node $i \in \mathcal{V} \setminus V$ we assign its label by computing $\arg\max_{k \in \{1,\dots,K\}} F(i,k)$. A baseline method for computing all evaluations $\mathsf{F} = [F(i,k)]_{i,k}$ boil down a particular case of Prob. 3.1.1:

$$\hat{\mathsf{F}} = \underset{\mathsf{F} \in \mathbb{R}^{n \times K}}{\operatorname{argmin}} \, ||\mathsf{Y} - \mathsf{F}||_{\mathsf{Q}}^2 + \operatorname{tr}(\mathsf{F}^\top \mathsf{L} \mathsf{F}). \tag{3.17}$$

The solution to this optimization scheme is a smooth function over the graph, which still preserves the original labelling information encoded in Y. *In other words,* $\hat{\mathsf{F}}$ *gives a diffused version of the given labels in* Y *through the neighborhoods of the graph. As in Prob. 3.1.1, one has the closed form solution as follows:*

$$\hat{\mathsf{F}} = (\mathsf{Q} + \mathsf{L})^{-1}\mathsf{Q}\mathsf{Y}. \tag{3.18}$$

Again, this leaves us with the expensive computation of $\mathsf{K}\mathbf{y}$ *for computing each column of* $\hat{\mathsf{F}}$.

Adapting the forest-based estimators for approximating all columns $\hat{\mathsf{F}}$ is easy; one only needs to sample Φ_Q by passing $Q = \mathrm{diag}(\mathsf{Q})$ and compute $\bar{\mathbf{x}}$ (or $\bar{\mathbf{z}}$) per class k by passing $\mathbf{y} = \mathsf{Y}_{:|k}$. As we can re-use the same forests to estimate each column $\hat{\mathsf{F}}$, we avoid the cumbersome of re-sampling.

Different configuration for Q yields a different type of regularization. For example, for $\mathsf{Q} = q\mathsf{D}$, the solution becomes $\hat{\mathsf{F}} = q(q\mathsf{I} + \mathsf{L}_{rw})^{-1}\mathsf{Y}$ where one obtains a regularization with the random walk Laplacian $\mathsf{L}_{rw} = \mathsf{D}^{-1}\mathsf{L}$. On top of this, one can obtain a regularization with the normalized graph Laplacian $\mathsf{L}_n = \mathsf{D}^{-1/2}\mathsf{L}\mathsf{D}^{-1/2}$ by slight modifications on the formulation [AMGS12]. All of these cases are still straightforward to adapt for the RSF estimators. We find a particular case of this formulation in which the adaptation of the forest estimators might not be obvious. This case is due to the celebrated algorithm for semi-supervised learning called label propagation [Zhu05]. Let us detail this algorithm by showing how it is linked to the formulation in Eq. (3.17). Then, we show how to use the RSF-based estimators to approximate the solution given by this algorithm.

As described in [Zhu05], label propagation consists of the following steps:

1. Set $k \leftarrow 1$.
2. Set $\mathsf{F}^{(k)} \leftarrow \mathsf{D}^{-1}\mathsf{W}\mathsf{Y}$ and increment k,
3. Reassign $\mathsf{F}^{(k)}_{V|:} \leftarrow \mathsf{Y}_{V|:}$,
4. Go to step 2 unless $\mathsf{F}^{(k)}$ satisfies a convergence criteria *e.g.* $||\mathsf{F}^{(k)} - \mathsf{F}^{(k-1)}|| \leq \epsilon$ for some $\epsilon > 0$.

This iterative algorithm propagates the known labels through the neighborhoods at every step. In doing so, it uses the random walk operator $\mathsf{P} = \mathsf{D}^{-1}\mathsf{W}$ to diffuse

the information encoded in Y. As a result it converges to a closed form solution as follows:

$$\hat{\mathsf{F}}_{i,k} = \begin{cases} \mathsf{Y}_{i,k} & \text{if } i \in V \\ \left(-(\mathsf{L}_{U|U})^{-1}\mathsf{L}_{U|V}\mathsf{Y}_{V|:}\right)_{i,k} & \text{otherwise} \end{cases}, \tag{3.19}$$

where $U = \mathcal{V} \setminus V$. This solution, in fact, is a limited case of the solution of Eq. (3.17). We show this case in the following proposition:

Proposition 3.4.2 (Label Propagation)**.** *For the solution in Eq.* (3.18)*, consider the configuration of the hyperparameters:*

$$\mathsf{Q}_{i,i} = \begin{cases} q & \text{if } i \in V \\ 0 & \text{otherwise} \end{cases}$$

Then, the limit $\lim_{q\to\infty} \hat{\mathsf{F}}$ *converges to the solution given in Eq.* (3.19)*.*

Proof. Let $\mathsf{I}_V \in \mathbb{R}^{n\times n}$ be a matrix with $(\mathsf{I}_V)_{i,i} = 1$ whenever $i \in V$ and zero elsewhere. For the given parameter setting, the matrix K takes a form of:

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

We re-write this inverse in block matrix form (without loss of generality, the nodes in V are enumerated $1, \ldots, |V|$):

$$\mathsf{K} = \begin{bmatrix} q^{-1}\mathsf{L}_{V|V} + \mathsf{I} & q^{-1}\mathsf{L}_{V|U} \\ q^{-1}\mathsf{L}_{U|V} & q^{-1}\mathsf{L}_{U|U} \end{bmatrix}^{-1} \mathsf{I}_V.$$

The non-zero blocks of this matrix can be computed by block matrix inversion formula:

$$\mathsf{K} = \begin{bmatrix} \mathsf{A} & 0 \\ -(\mathsf{L}_{U|U})^{-1}\mathsf{L}_{U|V}\mathsf{A} & 0 \end{bmatrix}.$$

where $\mathsf{A} = (q^{-1}\mathsf{L}_{V|V} + \mathsf{I} - q^{-1}\mathsf{L}_{V|u}(\mathsf{L}_{u|u})^{-1}\mathsf{L}_{u|V})^{-1}$. As α goes to infinity, one has

$$\lim_{q\to\infty} \mathsf{K} = \lim_{q\to\infty} \begin{bmatrix} \mathsf{A} & 0 \\ -(\mathsf{L}_{U|U})^{-1}\mathsf{L}_{U|V}\mathsf{A} & 0 \end{bmatrix} = \begin{bmatrix} \mathsf{I} & 0 \\ -(\mathsf{L}_{U|U})^{-1}\mathsf{L}_{U|V}\mathsf{I} & 0 \end{bmatrix}.$$

□

This proposition in fact gives us a way to adapt the RSF estimators for approximating the solution in Eq. (3.19). We sample Φ_Q with the given parameter setting. By definition, we only have non-zero q values over the vertices in V. As $q \to \infty$, the

loop-erased random walks in Wilson's algorithm are always interrupted whenever they reach a node in V. Therefore, all sampled forests are conditioned to be rooted in V. Moreover, $\bar{\mathbf{x}}$ converges to $\tilde{\mathbf{x}}$, as the roots in Φ_Q given the partitions are no longer random as we have a predefined root set V.

We finish this section by illustrating the forest estimates in solving graph based SSL over some benchmark datasets, namely Cora, Citeseer and Pubmed. These datasets were previously introduced in Table 3.1

In the simulations, we consider a setup where $k \ll n$ vertices per class are labeled and the goal is to classify other vertices given the labeled ones and the underlying graph. In doing so, we use two solutions, namely label propagation (LP) [Zhu05] and generalized SSL (gSSL) [AMGS12]. In both solutions, one needs to calculate the classification function F by computing $\hat{\mathbf{x}}$ per each class under different parametrizations. As we previously discussed, one can estimate both cases by using the forest-based estimator $\bar{\mathbf{x}}$ without taking expensive matrix inverses. In the following illustration, we compare the classification accuracy yielded by $\bar{\mathbf{x}}$ with that of $\hat{\mathbf{x}}$. In doing so, we use the following procedure:

- Select $k \ll n$ vertices are selected at random per class as the labeled nodes,
- Compute the classification functions F_{LP} and F_{gSSL} given LP and gSSL by using $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}$ per class.
- For each vertex i, assign $\arg\max_c \mathsf{F}i, c$ as its class and calculate the classification accuracy as the ratio of correctly predicted labels to the total number of predictions.

In Fig. 3.10, we plot the classification accuracy of LP, gSSL and their forest estimation while varying the number of labeled vertices per class k. We average the results over 50 realizations. The empirical results show that the forest estimator for gSSL gives close estimates for the exact solution in Citeseer and Cora whereas it fails to provide the same performance in Pubmed. This is probably due to insufficient number of labeled vertices or poor hyper-parameter choice for the forest estimators. Moreover, one can also deduce $\bar{\mathbf{x}}$ need much less forest realizations to reach the exact solution of LP rather than the generalized SSL. However, sampling a forest for LP might take more time compared to the time needed for gSSL. For example, in the Pubmed graph, for $k = 20$, 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.

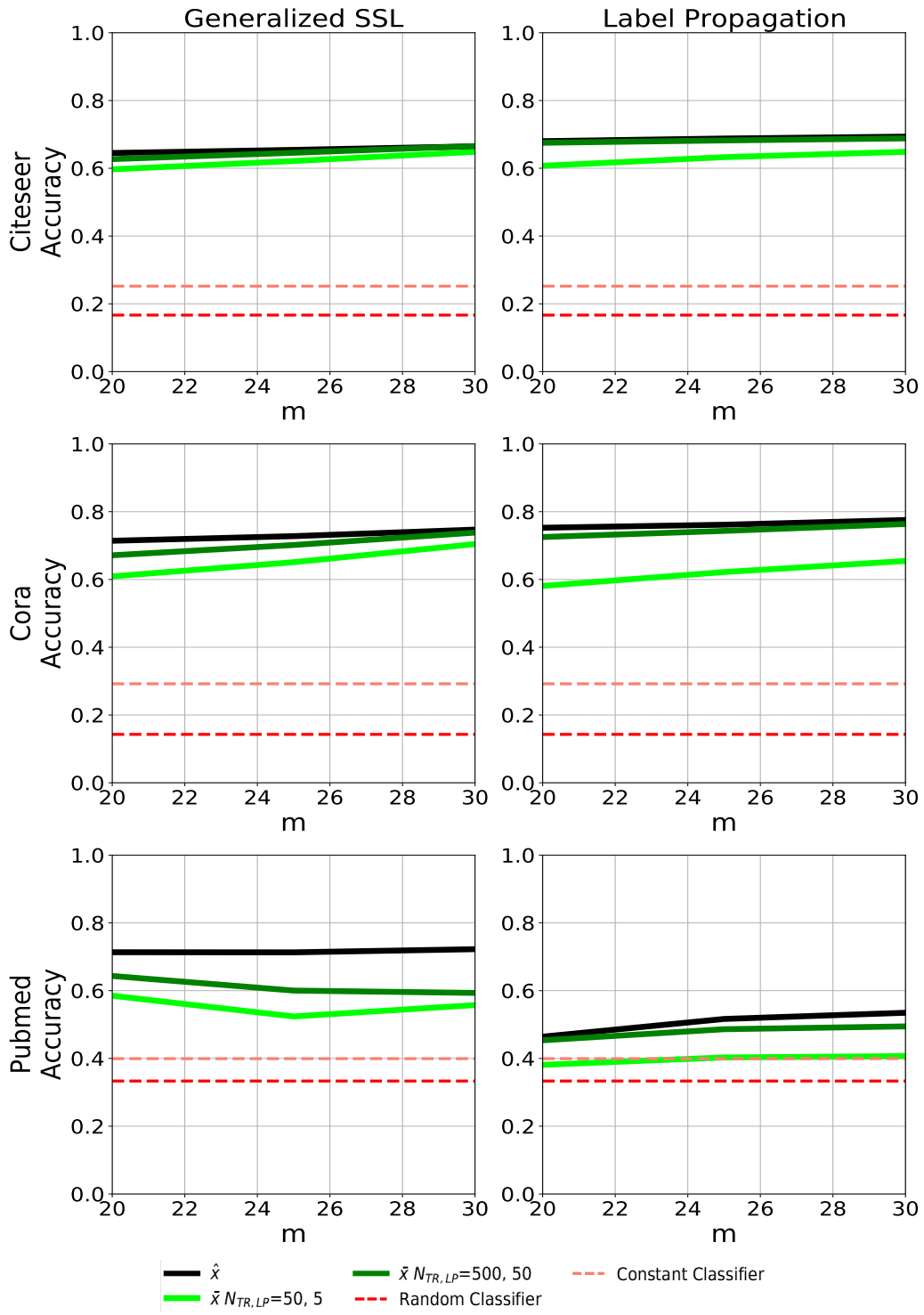


Fig. 3.10.: *The accuracy vs number of labeled vertices per class. We give the classification accuracy of gSSL (the first column) and LP (second column) and their forest estimates over the datasets Cora, Citeseer and Pubmed. In each plot, the performance of the exact solution is given in the black line. For gSSL, we compute* $\bar{\mathbf{x}}$ *over* 50 *(light green) and* 500 *(dark green) forest realization. In the case of LP, the number of realization becomes* 5 *(light green) and* 50 *(dark green). We also add the performance of a classifier that returns a label at random (random classifier) and returns the same label all the time (constant classifier).*

3.4.2 RSF-based Quasi Newton's Method

The graph Laplacian regularization is not restricted to the least-squares problem. In many other problems on graphs, one minimizes the loss function:

$$L(\mathbf{x}) = f_{\mathbf{y},q}(\mathbf{x}) + \mathbf{x}^\top \mathsf{L}\mathbf{x}$$

to find the solution:

$$\mathbf{x}^\star = \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}}\, L(\mathbf{x}).$$

In case of LS *i.e.* $f_{\mathbf{y},q}(\mathbf{x}) = q||\mathbf{x}-\mathbf{y}||_2^2$, this problem has a closed-form solution. However, this is not the case for many other formulations such as $f_{\mathbf{y},q}(\mathbf{x}) = q||\mathbf{x}-\mathbf{y}||_1$ or cross-entropy loss function. In such cases, the optimization literature is capable of serving a large diversity of algorithms to approximate the solution. For convex and differentiable $f(\mathbf{x},\mathbf{y},q)$, which we are interested in this thesis, the popular approach is the gradient descent algorithms which are generically introduced in 2.3.3. As long as we have access to the gradient $\nabla f_{\mathbf{y},q}(\mathbf{x})$, one can approach the solution with a convergence rate. If the loss function is also twice-differentiable, a faster approach called Newton's method (sometimes called Newton-Raphson method) brings the second derivatives into the game and derives the following iteration [HTFF09]:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathsf{H}^{-1}\nabla L(\mathbf{x}) \tag{3.20}$$

where $\mathsf{H} = \left[\frac{\partial^2 L(\mathbf{x})}{\partial x_i \partial x_j}\right]_{i,j}$ is the Hessian matrix which contains the partial second derivatives of the loss function. Let us assume that for $i \neq j$, $\frac{f_{\mathbf{y},q}(\mathbf{x})}{\partial x_i \partial x_j} = 0$. This implies that only prior correlation between the estimated parameters are induced by the graph. Also, by the convexity one has $\frac{\partial^2 f_{\mathbf{y},q}(\mathbf{x})}{\partial x_i^2} \geq 0$. Then the Hessian matrix boils down to a positive semi-definite matrix $\mathsf{H} = \mathsf{L} + \operatorname{diag}(\mathbf{g})$ where $g_i = \frac{\partial^2 f_{\mathbf{y},q}(\mathbf{x})}{\partial x_i^2}$ for all $i \in \mathcal{V}$.

Newton's method yields much faster convergence than the gradient descent. Intuitively it takes a more direct route to the solution by using extra information on the curvature. However, this extra information comes with a cost. Mainly, computing the inverse of the Hessian matrix is expensive when n is very large. To avoid this expensive operation, H^{-1} is often replaced with its approximations. This branch of algorithms is called quasi-Newton methods. SR1 (Symmetric rank-one) [CGT91], BFGS (Broyden–Fletcher–Goldfarb–Shanno) [Fle70] and DFS (Davidon–Fletcher–Powell) [Dav91] are some of the well-known algorithms that approximate H^{-1} deterministically. As these algorithms are cheap approximations, their convergence rate is naturally slower (super-linear) compared to the rate of Newton's

method (quadratic). However, in large dimensions, they may reach the solution more quickly.

Interestingly, the forest-based estimators also can be used for approximating the update in Newton's iteration. In particular, at every iteration k, we first sample RSFs with the parameter set $Q = \{g_1, \ldots, g_n\}$. Then we evaluate $\bar{\mathbf{x}}$ over these forests with the measurements $\mathbf{y} = \left[\frac{\nabla L(\mathbf{x}_k)_i}{g_i}\right]_{i \in \mathcal{V}}$. It is straightforward to see that this setup $\bar{\mathbf{x}}$ gives an unbiased estimate of the update at every iteration. However, as the updates are stochastic, the convergence might not be guaranteed. Nevertheless, in practice, it is possible to guide the iterations towards the solution by adjusting the step size. In this case, we consider the following iteration:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \beta \bar{\mathbf{x}},$$

where $\beta \in \mathbb{R}^n$. Coupling with line search approaches [BBV04], it is possible to keep the iterations within *safe* ranges. We illustrate this technique on an image denoising application where we consider a Poisson prior for $f_{\mathbf{y},q}(\mathbf{x})$.

Removing Shot Noise from an Image

Fig. 3.11.: The original (left) and noisy image (right). The peak signal-to-noise ratio for the noisy image is 17.13

In image de-noising, it is important to know or assume the probabilistic noise model correctly. A big chunk of de-noising algorithms [LBU10] assumes that the noise is generated by a Gaussian or Poisson process. While the former (Gaussian) assumption can be easily motivated by the celebrated central limit theorem, the latter (Poisson) is due to a physical phenomenon that is more specific to the image processing domain. An imaging device is an array of sensors where each sensor measures the intensity of the light by counting the photons hitting the sensor. This count turns out to be random at a constant level of light. Due to this randomness, the measurements collected from the sensors are noisy. This type of noise is called shot or Poisson noise (See Fig. 3.11). Further derivations deduce that the Poisson distribution is a natural model for the random number of photons hitting each sensor, hence the name.

We use the graph Laplacian regularization in this illustration to remove the shot noise. Here we consider a 2D-grid graph in which each sensor is a node connected to the sensor on its left, right, top and bottom.

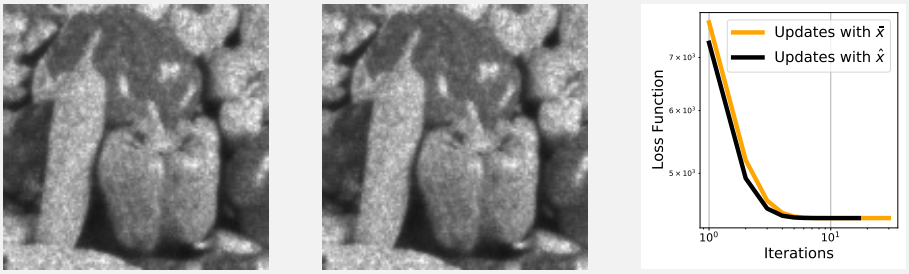


Fig. 3.12.: The images denoised by Newton's method where the updates calculated by $\hat{\mathbf{x}}$ (left) and $\bar{\mathbf{x}}$ (middle). On the right, we plot the loss function through the iterations of Newton's method. We assume a 2D-grid graph where $n = 128 \times 128$. The hyperparameter is set to 0.1, and α is selected at every iteration by approximate line search algorithm. For the forest-based updates, we sample $N = 40$ forests. The peak signal-to-noise ratios on the final images are found as 24.84 for both images.

Let us denote the random number of photons counted by the sensor i by Y_i. Then we assume:

$$\mathbb{P}(Y_i = y_i | \lambda = \exp(x_i)) = \frac{\exp(x_i)^{y_i} \exp(-\exp(x_i))}{y_i!},$$

where λ is the parameter of the Poisson distribution. Given this prior, we adapt our fidelity term with the following log-likelihood function:

$$f_{\mathbf{y},q}(\mathbf{x}) = -q \sum_{i \in \mathbb{V}} \log \mathbb{P}(Y_i = y_i | \lambda = \exp(x_i)) = \sum_{i \in \mathbb{V}} -y_i x_i + \exp(x_i) + \log y_i!.$$

Plugging this leaves us with the following optimization problem:

$$\mathbf{x}^\star = \underset{\mathbf{x} \in \mathbb{R}^n}{\operatorname{argmin}}\, q \left(\sum_{i \in \mathbb{V}} -y_i x_i + \exp(x_i) \right) + \frac{1}{2} \mathbf{x}^\top \mathsf{L} \mathbf{x}.$$

As this problem does not have a closed-form solution and the loss function is convex and twice-differentiable, Newton's method is suitable for approximating the solution. Let us quickly write the gradient and the Hessian for this loss function:

$$\forall i \in \mathcal{V}, \quad \nabla L(\mathbf{x})_i = (\mathsf{L}\mathbf{x})_i + q(\exp(x_i) - y_i) \quad \text{and} \quad \mathsf{H} = \mathsf{L} + q \operatorname{diag} \mathbf{g},$$

where $\forall i \in \mathcal{V},\ g_i = \exp(x_i) \geq 0$. Then the forest-based estimation of the Newton's update boils down evaluating $\bar{\mathbf{x}}$ under the parameter setting $Q = \{q\exp(x_1), , \ldots, q\exp(x_n)\}$ and $\mathbf{y} = \left[\frac{\nabla L(\mathbf{x}_k)_i}{qg_i}\right]_{i\in\mathcal{V}}$. In Fig. 3.12, we give an illustration of this setup.

3.4.3 L-1 Regularization on Graphs

The graph-based regularization can be found in various forms. A generalization can be made by the p-Laplacian operator:

$$r_{\mathcal{G},p}(\mathbf{x}) = \left[\sum_{(i,j)\in\mathcal{E}} w(i,j)^{p/2}|x_i - x_j|^p\right].$$

In case $p = 2$, one recovers $r_{\mathcal{G},2}(\mathbf{x}) = \mathbf{x}^\top \mathbf{L}\mathbf{x}$. In this section, we focus on another popular choice, $p = 1$. When applied, this type of regularization, sometimes called the edge LASSO (Least Absolute Shrinkage and Selection Operator), minimizes the variation of the solution over the graph in the LASSO sense. In turn, the solution usually forms a piece-wise constant signal over the vertices. Let us rewrite the optimization problem by combining it with this new regularization:

$$\mathbf{x}^\star = \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \frac{q}{2}||\mathbf{x} - \mathbf{y}||_2^2 + ||\mathbf{B}\mathbf{x}||_1. \tag{3.21}$$

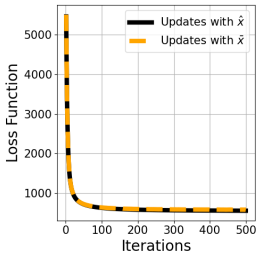
This formulation yields a convex optimization problem. However, it does not have a closed-form solution, unlike the case of $p = 2$, as the corresponding loss function is not differentiable everywhere. Yet, there are various algorithms to compute the minimizer $\mathbf{x}^\star$. The most straightforward approaches, such as ISTA, use the proximal gradient methods. More advanced methods such as ADMM and IRLS provide faster convergence to $\mathbf{x}^\star$ by some extra operations. In the case of ADMM and IRLS, this operation boils down to inverting a matrix in the form of the regularized Laplacian at every iteration. This section shows how to adapt the forest estimators for approximating this expensive inversion. We restrict ourselves to the case of ADMM. However, one can find the same adaptation for IRLS in our journal paper [PABT21b].

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

(b) Noisy Measurements $\mathbf{y}$, PSNR = 13.98

(c) ADMM with $\hat{\mathbf{x}}$ updates, PSNR=22.44

(d) ADMM with $\bar{\mathbf{x}}$ updates, PSNR=22.44



(e) Loss function through iterations

Fig. 3.13.: *We illustrate ADMM algorithm for image denoising application. We are given a noisy measurements $\mathbf{y} = \mathbf{x} + \epsilon$ where the original image $\mathbf{x}$ is unknown and the noise is Gaussian i.e. $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma = 0.2)$. We denoise the image by solving Eq (3.21) and the regularization parameter q is found as 3.3 by a grid search. In (c) and (d), we give the solutions by ADMM the algorithm where the updates are calculated via the exact and forest-based solutions. In both cases, ADMM is terminated after 500 iterations and $\rho = 0.2$. The forest updates are averaged over 10 forests realizations per iteration. Finally, in (e), we plot the objective function across the iterations.*

The ADMM algorithm for the problem in Eq. (3.21) takes the following three steps iteration scheme [BPCPE+11]:

$$\begin{aligned}
\mathbf{x}_{k+1} &= \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} \left(\frac{q}{2}||\mathbf{x}-\mathbf{y}||_2^2 + \frac{\rho}{2}||\mathsf{B}\mathbf{x}-\mathbf{z}_k+\mathbf{u}_k||_2^2 \right) \\
\mathbf{z}_{k+1} &= \underset{\mathbf{z}\in\mathbb{R}^m}{\operatorname{argmin}} \left(||\mathbf{z}||_1 + \frac{\rho}{2}||\mathsf{B}\mathbf{x}_{k+1}-\mathbf{z}+\mathbf{u}_k||_2^2 \right) \\
\mathbf{u}_{k+1} &= \mathbf{u}_k + (\mathsf{B}\mathbf{x}^{k+1} - \mathbf{z}^{k+1}).
\end{aligned}$$

The parameters $\mathbf{x}_k$, $\mathbf{y}_k$ and $\mathbf{u}_k$ are arbitrarily initialized and are updated at every step, and ρ is a user-defined parameter. The most time-consuming part in this iteration is the first step, where we need to compute the inverse:

$$\mathbf{x}_{k+1} = (q\mathsf{I} + \rho\mathsf{L})^{-1}(q\mathbf{y} + \rho\mathsf{B}^\top\mathbf{z}_k - \rho\mathsf{B}^\top\mathbf{u}_k).$$

Instead of directly computing this inverse at every step, one can save a significant amount of time by replacing them with forest-based estimates. Moreover, unlike in the case of Newton's method, we do not have to re-sample RSFs at every step. After generating N samples of $\Phi_{q/\rho}$, we can quickly re-evaluate the forest estimates at every step k by simply updating the input signal.

We give a proof of concept for this forest-based ADMM algorithm by a simple example in Fig. 3.13. In this illustration, we consider the image denoising problem with Gaussian noise. In order to denoise the given measurements in Fig. 3.13b, we solve Eq. (3.21) in which the underlying graph is 128×128 2-D grid. In doing so, we use the ADMM algorithm with two types of updates for calculating $\mathbf{x}_{k+1}$ at every iteration; these are the exact computation and the forest-based estimation via $\bar{\mathbf{x}}$. In qualitative results (Figs. 3.13c and 3.13d), we observe that the final result is almost the same for both types of updates. By monitoring the loss function through the iterations, we also observe that the forest-based updates, for a few realizations of forests ($N = 10$), closely follow the exact updates in minimizing the objective function.

3.5 Conclusion

Network structured data often come with some signal/information over the vertices, called graph signals. These kinds of signals are subject to noise or incompleteness. Many classical signal processing tools have been adapted for graph signals in dealing with these issues. A well-known technique gives a unified solution to these two

problems by solving the optimization problem 3.1.1. This problem admits a linear closed-form solution with $\mathbf{y}$. Moreover, it appears within several graph-related contexts, namely graph signal filtering and Dirichlet boundary problem. In the former, we see that a particular case of this solution, *i.e.* $\mathsf{Q} = q\mathsf{I}$ and $\mathbf{y} \in \mathbb{R}^n$, corresponds to filtering a graph signal with the low-pass transfer function $g(\lambda) = \frac{q}{q+\lambda}$. In the latter, we consider discrete partial differential equations defined on graphs that model various physical phenomena. We show that the solution of these equations is again a particular case ($\mathsf{Q} = \lim_{q\to\infty} \mathsf{I}_V$) of the solution of Prob. 3.1.1.

Indeed this one-line solution is of central importance in various fields. However, directly computing it might not be straightforward as n grows. The time complexity for computing the inverse is $\mathcal{O}(n^3)$, which is prohibitive whenever n is around thousands or millions. Having state-of-the-art as the approximate methods in large n, our contributions in this chapter are in avoiding the expensive computations in estimating $\hat{\mathbf{x}}$. Let us finish by summarizing them:

- We propose an unbiased estimator $\tilde{\mathbf{x}}$ for $\hat{\mathbf{x}}$ based on Φ_Q,
- The expected time complexity of this algorithm is found as $\mathrm{tr}(\mathsf{Q}+\mathsf{L})^{-1}(\mathsf{Q}+\mathsf{D}) \leq \frac{2m}{q}+n$ which remains comparable with SOTA algorithms in terms of the number operations at every sample (resp. every iteration for CGD),
- We show how to adapt two variance reduction techniques, namely conditional Monte Carlo and control variate, to significantly improve the performance of $\tilde{\mathbf{x}}$ (in approximating $\hat{\mathbf{x}}$), yielding two new estimators $\bar{\mathbf{x}}$ and $\bar{\mathbf{z}}$,
- We show three different real-life use cases in which these estimators can accelerate the complicated intermediate steps.
- The proposed methods are easy to implement and they have a low memory footprint *i.e.* $\mathcal{O}(n)$. Moreover, they are easy to parallelize as we can start random walks from arbitrary nodes for sampling RSFs.

Given these fruitful results and diverse applications, in the next chapter, we shift our focus on the estimation of other useful quantities, including $\mathrm{tr}(\mathsf{K})$ or effective resistances. In turn, we find forest-based algorithms and significant improvements on the existing ones.

4 RSF Estimation of Some Important Graph Quantities

> *All exact science is dominated by the idea of approximation.*
>
> **— Bertrand Russell**

The forest estimators for $\hat{\mathbf{x}}$ are not the only outcome of our *hike* through random spanning forests. Estimating the trace $\mathrm{tr}(\mathsf{K})$, the effective resistances and other graph filters than $\frac{q}{q+\lambda}$ are three other objectives which we can achieve by using RSFs. In a higher abstraction, the first problem can be considered as the estimation of the trace of an inverted matrix where the inverse is expensive to take. Various applications require such an estimation. Closer to our work, the estimation of $\mathrm{tr}(\mathsf{K})$ is of central importance in the hyper-parameter tuning in Prob. 3.1.1 *i.e.* tuning of q. Similarly, effective resistances have a broad set of applications [SS11; SXGRS20; AALG17; WPKV14]. Initially studied in electrical networks, effective resistances define a measure over the vertices, which has many theoretical links with random walks and USTs. However, directly computing them might be computationally demanding as it requires access to the entries of the pseudo inverse $\mathsf{L}^\dagger$. Finally, we show that the graph filters that can be obtained by RSFs are not limited to a single transfer function $\frac{q}{q+\lambda}$. In fact, one can reach a wide family of filters via RSFs by slight modifications. We dedicate this chapter to presenting our novel RSF algorithms for estimating $\mathrm{tr}(\mathsf{K})$ and the effective resistances and we show how to leverage RSF estimators for other types of graph filters.

Contents

4.1 Estimation of the Trace of the Regularized Inverse of the Laplacian

Reconsider the simplified version of the denoising problem 3.1.1:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}}\, q||\mathbf{x}-\mathbf{y}||_2^2 + \mathbf{x}^\top \mathsf{L}\mathbf{x}.$$

Recall that q is a user-defined parameter. However, the *good* value of q in the sense of denoising performance cannot be explicitly known. Fortunately, there are several ways, such as AIC, BIC, cross-validation methods and SURE [HTFF09], to find a near-optimal value for q. All of these algorithms either need to compute the diagonal entries $\mathsf{K}_{i,i}$'s, or the trace $\operatorname{tr}(\mathsf{K})$ where $\mathsf{K} = q(\mathsf{L}+q\mathsf{I})^{-1}$. However, in both cases, one needs to calculate K for different values of q. As aforementioned, the direct computation comes with a prohibitive cost for large n. This leaves us with approximate methods. The state-of-the-art algorithm for approximating the trace is Hutchinson's estimator [Hut89]:

Definition 4.1.1 (Hutchinson's estimator)**.** Let $\mathbf{p} \in \mathbb{R}^n$ be a random vector with each element $p_i \in \{-1, 1\}$ and independently distributed as:

$$\mathbb{P}(p_i = \pm 1) = \frac{1}{2}.$$

Given N samples from $\mathbf{p}$, Hutchinson's estimator for the trace of K is defined as follows:

$$h := \frac{1}{N}\sum_{k=1}^{N} \mathbf{p}_{(k)}^\top \mathsf{K}\mathbf{p}_{(k)}.$$

It is easy to show that this estimator is unbiased with the permutation property of the trace and $\mathbb{E}[\mathbf{p}\mathbf{p}^\top] = \mathsf{I}$:

$$\mathbb{E}[h] = \frac{1}{N}\sum_{k=1}^{N}\mathbb{E}[\mathbf{p}^\top \mathsf{K}\mathbf{p}] = \frac{1}{N}\sum_{k=1}^{N}\operatorname{tr}(\mathbb{E}[\mathbf{p}\mathbf{p}^\top]\mathsf{K}) = \operatorname{tr}(\mathsf{K}).$$

Moreover, the variance is tractable:

$$\operatorname{Var}(h) = \frac{2}{N}\left(\operatorname{tr}(\mathsf{K}^2) - \sum_{i=1}^{n}\mathsf{K}_{i,i}^2\right) = \frac{2}{N}\sum_{i\neq j}^{n}\mathsf{K}_{i,j}^2.$$

In other words, the variance is equal to the squared sum of the off-diagonal entries in K.

In some other methods, such as Girard's estimator, the distribution of $\mathbf{p}$ differs, but they still provide an unbiased estimation. In fact, as long as the random vector $\mathbf{p}$ satisfies $\mathbb{E}[\mathbf{p}] = \mathbf{0}$ and $\mathbb{E}[\mathbf{p}\mathbf{p}^\top] = \mathsf{I}$, one obtains an unbiased estimator by the averaging of quadratic terms. However, Hutchinson's method is the estimator that provides the minimum variance [Hut89]. Even so, it is still subject to two significant bottlenecks:

- Computation of each quadratic term $\mathbf{p}_{(k)}^\top \mathsf{K}\mathbf{p}_{(k)}$ requires computing $\mathsf{K}\mathbf{p}_{(k)}$ which requires inverting $\frac{1}{q}(\mathsf{L} + q\mathsf{I})$,
- As it is a Monte Carlo estimator, it has the convergence rate of $\mathcal{O}(N^{-1/2})$, thus one might need many samples to reach an very accurate result.

Taking the inverse is not practical with the exact methods, such as Gaussian elimination. Therefore, the solution is often calculated via iterative methods such as gradient descent algorithms. In dealing with the second issue, one can find many generic variance reduction methods in the literature [Owe13]. However, it is not obvious how to adapt these methods for Hutchinson's estimator to obtain considerable performance gain.

Previously in [BTGAA19], the authors (my supervisors) proposed a simple estimator for $\operatorname{tr}(\mathsf{K})$ with strong theoretical properties:

$$s \coloneqq |\rho(\Phi_q)|.$$

The expectation and variance of this estimator are already noted as:

$$\mathbb{E}[s] = \operatorname{tr}(\mathsf{K}) \text{ and } \operatorname{Var}(s) = \operatorname{tr}(\mathsf{K} - \mathsf{K}^2).$$

Moreover, they give an empirical comparison with Girard's estimator coupled with several iterative methods for computing $\mathsf{K}\mathbf{p}$. The experimentation over various graphs shows that the forest estimator often gives the best or at least comparable performance with state-of-the-art. Inspired by these results, we give two efficient ways of reducing the variance of the RSF trace estimator. These ways are well-known generic variance reduction techniques in the Monte Carlo literature. However, their adaptation for a given Monte Carlo estimator is not evident or sometimes not possible. The main contribution of this section shows how to adapt these variance reduction techniques for the forest-based trace estimator [BTGAA19] yielding two novel estimators for $\mathrm{tr}(\mathsf{K})$.

4.1.1 Variance Reduction for Trace Estimation

We adapt two variance reduction techniques, namely the control variate method and stratified sampling, for reducing the variance of $|\rho(\Phi_q)|$.

4.1.1.1. Control Variate for Trace Estimation

In this section, we adapt the control variate technique introduced in Section 3.3.2. Let us briefly recall this technique and how it is applied to the previous estimator for $\hat{\mathbf{x}} = \mathsf{K}\mathbf{y}$. Given the estimator $\bar{\mathbf{x}}$, we seek another random quantity with a known expectation. In our case, we chose this quantity as $\alpha(\mathsf{K}^{-1}\bar{\mathbf{x}} - \mathbf{y})$ and the resulting estimator is $\bar{\mathbf{z}} = \bar{\mathbf{x}} - \alpha(\mathsf{K}^{-1}\bar{\mathbf{x}} - \mathbf{y})$. Let us redefine $\bar{\mathbf{z}}$ as a matrix-vector product $\bar{\mathbf{z}} \coloneqq \bar{\mathsf{Z}}\mathbf{y}$ where $\bar{\mathsf{Z}} = \bar{\mathsf{S}} - \alpha(\mathsf{K}^{-1}\bar{\mathsf{S}} - \mathsf{I})$ and $\bar{\mathsf{S}}$ verifies $\bar{\mathbf{x}} = \bar{\mathsf{S}}\mathbf{y}$. We already know that $\mathbb{E}[\bar{\mathsf{Z}}] = \mathsf{K}$. Then, a new unbiased estimator of $\mathrm{tr}(\mathsf{K})$ is defined by taking the trace of $\bar{\mathsf{Z}}$:

$$\begin{aligned}\bar{s} &\coloneqq s - \alpha(\mathrm{tr}(\mathsf{K}^{-1}\bar{\mathsf{S}} - \mathsf{I})) \\ &= s - \alpha\left(n - s - \frac{1}{q}\sum_{i=1}^{n}\sum_{j\in\mathcal{N}(i)} w(i,j)\bar{\mathsf{S}}_{i,i}\mathbb{I}(r_{\Phi_q}(i) \neq j)\right),\end{aligned} \tag{4.1}$$

where $s = \mathrm{tr}(\tilde{\mathsf{S}}) = \mathrm{tr}(\bar{\mathsf{S}}) = |\rho(\Phi_q)|$. Here the sum over the neighbors of all the vertices accumulates the weighted diagonal entries of $\mathsf{S}_{i,i}$ over the edges that are between different parts of the partition induced by Φ_q. We illustrate these edges in Fig. 4.1.

For certain values of α, $\bar{s}$ has a reduced variance. In fact, the choices are quite the same as in $\bar{\mathbf{z}}$. Plugging the identity matrix $\mathsf{I} \in \mathbb{R}^{n\times n}$ instead of $\mathbf{y}$ in Prop. 3.3.3, we

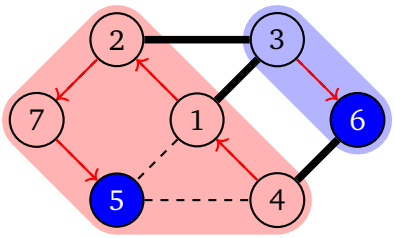


Fig. 4.1.: The edges between trees in a spanning forest (in bold). One needs to track these edges to compute the control variate introduced in Eq. 4.1. For computing the control variate induced by $\tilde{\mathsf{S}}$, one only needs the edges between trees that are incident to the roots. In this example, only edge that fits this description is $(4, 6)$.

see that $\alpha = \frac{2q}{q+2d_{max}}$ guarantees variance reduction. It is also possible to adapt the formulas of $\hat{\alpha}$ for estimating the optimal case $\alpha^\star$. Finally, the heuristic value $\alpha = \frac{q}{q+d_{avg}}$ performs very well in practice across various graphs.

Under a proper selection of α, this technique can substantially decrease the variance. However, it also requires some additional computations. In particular, the additional computation after sampling a forest for computing $\bar{s}$ requires traversing the edges of the graph. By replacing $\bar{\mathsf{S}}$ with $\tilde{\mathsf{S}}$, this cost can be decreased at the expense of accuracy because for computing the control variate in Eq. (4.1), one only needs to accumulate some edge weights only around the roots instead of all vertices. In this case, one does not have to compute $\bar{\mathsf{S}}_{i,i}$'s and only need to traverse the neighbours of the roots. However, this version, denoted by $\tilde{s}$, is worse than $\bar{s}$ in approximating $\mathrm{tr}(\mathsf{K})$ as $\tilde{\mathsf{S}}$ is less accurate $\bar{\mathsf{S}}$.

4.1.1.2. Stratified Sampling for Trace Estimation

Another variance reduction technique that is applicable to our case is stratified sampling. In a nutshell, stratified sampling is a divide & conquer strategy for improving Monte Carlo estimation. Let us go back to the generic example of Monte Carlo estimation where we approximate the expectation μ_X from the samples of the random variable X. Now consider another random variable Y whose outcome space $\Omega(Y)$ is divided into K strata (disjoint parts) $C_1, C_2, \ldots, C_K$ such that $\Omega(Y) = \cup_{k=1}^{K} C_k$. We assume that Y verifies for all C_k:

- The probabilities $\mathbb{P}(Y \in C_k)$ are known or easy to compute,
- The conditional random variable $X|Y \in C_k$ is easy to sample.

These constraints might seem a bit technical and hard to satisfy for all cases. However, whenever such a random variable Y is available, stratified sampling can decrease the variance significantly by defining a new estimator as follows:

$$Z \coloneqq \sum_{k=1}^{K} \left(\frac{1}{N_k} \sum_{j=1}^{N_k} X_j | Y \in C_k \right) \mathbb{P}(Y \in C_k)$$

N_k is the number of samples taken from the conditional random variable $X|Y \in C_k$. The stratified estimator Z is also unbiased since:

$$\begin{aligned} \mathbb{E}[Z] &= \sum_{k=1}^{K} \left(\frac{1}{N_k} \sum_{j=1}^{N_i} \mathbb{E}[X_j | Y \in C_k] \right) \mathbb{P}(Y \in C_k) \\ &= \sum_{k=1}^{K} \mathbb{E}[X|Y \in C_k] \mathbb{P}(Y \in C_k) = \mathbb{E}[X] = \mu_X. \end{aligned}$$

Also, the variance of Z reads:

$$\mathrm{Var}(Z) = \sum_{k=1}^{K} \mathbb{P}(Y \in C_k)^2 \frac{1}{N_k} \mathrm{Var}(X|Y \in C_k).$$

This variance is guaranteed to be less than that of the original estimator for certain configurations of N_k's. For example, if one can collect samples of $X|Y \in C_k$ such that $N_i \propto \mathbb{P}(Y \in C_k)$ for all k, the variance of Z becomes:

$$\begin{aligned} \mathrm{Var}(Z) &= \sum_{k=1}^{K} \frac{1}{N\mathbb{P}(Y \in C_k)} \mathrm{Var}(X|Y \in C_k) \mathbb{P}(Y \in C_k)^2 \\ &= \sum_{k=1}^{K} \frac{1}{N} \mathrm{Var}(X|Y \in C_k) \mathbb{P}(Y \in C_k) = \frac{1}{N} \mathbb{E}_Y[\mathrm{Var}(X|Y \in C_k)] \leq \frac{1}{N} \mathrm{Var}(X). \end{aligned}$$

The last inequality is due to the law of total variance. In fact, the optimal configuration for variance reduction is:

$$\forall k, \quad N_k \propto \mathrm{Var}(X|Y \in C_k) \mathbb{P}(Y \in C_k)$$

However, this is not practical in most cases as the conditional variance $\mathrm{Var}(X|Y \in C_k)$ is often not known explicitly. In the case of forest-based trace estimation, we find a novel way to apply stratified sampling to reduce the variance. This adaptation yields a new estimator for $\mathrm{tr}(\mathsf{K})$ which often provides the best empirical performance.

$$\mathsf{S} = \begin{matrix} S^{(1)} & S^{(2)} & S^{(3)} & S^{(4)} & S^{(5)} & S^{(6)} & S^{(7)} \end{matrix}$$
$$\begin{bmatrix} 2 & 7 & 6 & 1 & \Gamma & \Gamma & 1 \\ \Gamma & 7 & 2 & 5 & 4 & 4 & 2 \\ 3 & 7 & 1 & 5 & 1 & 4 & 1 \\ & & & \vdots & & & \end{bmatrix}$$

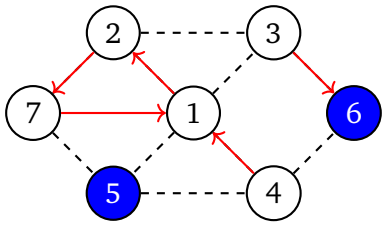


Fig. 4.2.: An example of the stack representation (left) and the graph induced by the first layer (right). The blue nodes indicate the roots sampled at the first layer.

In order to explain our stratification scheme for trace estimation, we need to recall some aspects of the stack representation of Wilson's algorithm. Recall the infinite stack $S^{(i)} = [S_1^{(i)}, S_2^{(i)}, \dots]$ defined for each node i. Every element in $S^{(i)}$ is an independent random variable with the probability law $\mathbb{P}(S_j^{(i)} = k) = \frac{w(i,k)}{d_i}$. In the case of sampling RSFs, we add a new node Γ to this representation and slightly change the probability distribution as:

$$\mathbb{P}(S_j^{(i)} = k) = \begin{cases} \frac{w(i,j)}{q+d_i}, & k \in \mathcal{N}(i), \\ \frac{q}{q+d_i}, & k = \Gamma. \end{cases}$$

Note that whenever a node has Γ at the top of its stack, it becomes a root. Now we define a new random set by only looking at the first layer of the stacks:

$$\rho_i(\Phi_q) \coloneqq \{v \in \mathcal{V} : S_j^{(v)} = \Gamma \text{ and } j \leq i\}.$$

In other words, the random set $\rho_i(\Phi_q)$ contains the random roots in Φ_q sampled until their i-th stack popped. See Fig. 4.2 for an example at $i = 1$. In Wilson's algorithm, this set can be seen as the roots sampled up until i-th visit by the random walks. Notice that as $i \to \infty$, we recover the original roots $\rho(\Phi_q)$. For stratification, we look at the cardinality of this set at $i = 1$. As all the stacks are independently sampled at the beginning *i.e.* $i = 1$, this cardinality equals a sum of Bernoulli random variables:

$$|\rho_1(\Phi_q)| = \sum_{v \in \mathcal{V}} \text{Ber}\left(\frac{q}{q+d_i}\right),$$

where $\text{Ber}(p) \in \{0, 1\}$ denotes a Bernoulli random variable that is equal to 1 with probability p. Our main idea here is to use $|\rho_1(\Phi_q)|$ as the additional random variable in stratification of the forest trace estimator s. Indeed, to do so, we need to show that this new random variable aligns with the constraints of stratified sampling.

The first constraint is to have probabilities that are easy to compute. In order to verify that this is the case with $|\rho_1(\Phi_q)|$, let us look at its probability distribution. By definition, $|\rho_1(\Phi_q)|$ is a sum of independent Bernoulli variables where each does not

necessarily have the same parameter. Such random variables, also known as Poisson-Binomial random variables, are one of the classical random variables in probability theory [Wan93]. As a result, their distribution has been well studied, and efficient direct or approximate numerical computations have already been proposed [Hon13]. In the case of large n, the normal approximations (motivated by the central limit theorem) are able to provide good performance at a very small cost. In this approach, we approximate the distribution of $|\rho_1(\Phi_q)|$ with a normal distribution $\mathcal{N}(\mu, \sigma^2)$ where $\mu = \mathbb{E}[|\rho_1(\Phi_q)|] = \sum_{v\in\mathcal{V}} \frac{q}{q+d_i}$ and $\sigma^2 = \mathrm{Var}(|\rho_1(\Phi_q)|) = \sum_{i\in\mathcal{V}} \frac{qd_i}{(q+d_i)^2}$. This approximation highly simplifies any computation related to the probabilities that we need to calculate as the cumulative/probability mass functions are explicitly known. Indeed, one can go beyond this approximation via more sophisticated approaches. However, by doing so, we do not obtain much improvement in practice for trace estimation. This is probably because the Monte Carlo error is dominant by far.

The second constraint dictates that the conditional random variable $|\rho(\Phi_q)|\,||\rho_1(\Phi_q)| \in C_k$ must be easy to sample for each stratum C_k. The sampling procedure per k can be considered two-fold:

- Generate a sample S of $\rho_1(\Phi_q)$ that verifies $|\rho_1(\Phi_q)| \in C_k$,
- Sample $|\rho(\Phi_q)|$ given the sample generated at the first step.

The second step is easy by a simple modification of Wilson's algorithm for forests. As S are the roots that are sampled at *the first sight*, in Wilson's algorithm, we set S as the predefined set of roots, and we do not allow any other to be a root at the first visit of random walks. The forest generated this way is a sample of $\Phi_q|\rho_1(\Phi_q) = S$. For the first step *i.e.* generating S that verifies $|S| \in C_k$, the naive approach is the rejection sampling where we keep sampling from $\rho_1(\Phi_q)$ until the constraint $|S| \in C_k$ is satisfied. This approach may terminate very quickly if $\mathbb{P}(|\rho_1(\Phi_q)| \in C_k) \gg 0$. Indeed, this constraint is directly related to how the strata $C_1, \ldots, C_K$ is chosen. In our case, we approximate the distribution of $|\rho_1(\Phi_q)|$ by a normal distribution. Thus, we can design the strata $C_1, \ldots, C_K$ so that $\mathbb{P}(|\rho_1(\Phi_q)| \in C_k) \gg 0$ is verified for all $k \in \{1, \ldots, K\}$. Both in theory and practice, we do not see a particular benefit of having strata with small probabilities for each stratum. Therefore, we choose to use rejection sampling in the application of trace estimation.

Up to this point, we show that $|\rho_1(\Phi_q)|$ is a theoretically and practically suitable random quantity for stratification of s. Then let us define the stratified estimator for $\mathrm{tr}(K)$ as follows:

$$s_{st} := \sum_{i=1}^{K} \frac{1}{N_i} \left(\sum_{\substack{j=1 \\ |\rho_1(\phi^{(j)})| \in C_i}}^{N_i} |\rho(\phi^{(j)})| \right) \mathbb{P}(|\rho_1(\Phi_q)| \in C_i). \tag{4.2}$$

This concludes the methods that we propose to reduce the variance of the forest-based trace estimators. In the next section, we give empirical comparisons of these two approaches (control variate and stratified sampling) with the state-of-the-art algorithms over various real-life graphs.

4.1.2 Empirical Results on Trace Estimation

This section is adapted from the results section of our paper [PABT22]. In these results, we compare the forest-based trace estimators s, $\bar{s}$, $\tilde{s}$ and s_{st} with Hutchinson's estimator combined with several linear solvers.

Firstly, notice that all methods in this comparison are Monte Carlo methods. Recall that the variance of a Monte Carlo estimator over N samples reads:

$$\sigma_N^2 \approx \frac{\sigma_1^2}{N},$$

where σ_1 is the standard deviation over a single sample.[1] We leverage this fact to compare the effective runtimes of all methods *i.e.* the time needed to reach a fixed relative error ϵ. First, we run all methods with $N = 100$. This gives us the average runtime for the computation per sample and the sample variance $\hat{\sigma}_N^2$. Then, we approximate $\hat{\sigma}_1 = \sqrt{N}\hat{\sigma}_N$ for each method. By using this approximation, we solve $\epsilon = \frac{\hat{\sigma}_1}{\mathrm{tr}(K)\sqrt{k}}$ for $\epsilon = 0.002$ to calculate the number of iterations k needed to reach ϵ error. Finally, we calculate the effective runtime per method by multiplying k by the average time for generating a single sample.

The linear solvers we use in Hutchinson's estimator are Algebraic Multigrid (AMG) [2], Conjugate Gradient (CG) [3], CG preconditioned with AMG and sparse Cholesky decomposition using CHOLMOD [CDHR08]. Here we use the block implementation

[1] This is also true for the stratified sampling with the proportional allocation *i.e.* $N_i \propto \mathbb{P}(|\rho'(\Phi_q)| \in C_i)$ for all i

[2] The implementation can be found in https://github.com/JuliaLinearAlgebra/AlgebraicMultigrid.jl

[3] The implementation can be found in https://docs.juliahub.com/KrylovMethods

of CG given in [OLe80]. For the control variate methods, we choose $\alpha = \frac{q}{q+\bar{d}_{avg}}$ for $\bar{s}$ and $\tilde{s}$. In stratified sampling, we divide the sample space into 5 strata $C_1, \ldots, C_5$ verifying $\mathbb{P}(|\rho'(\Phi_q)| \in C_k) \approx 0.2$ for all $k = 1, \ldots, 5$. Here, we empirically choose the number of strata so that $\mathbb{P}(|\rho'(\Phi_q)| \in C_k) \gg 0$. For too many strata, this probability approaches zero and generating samples from each stratum take much more time. For too few, the effect of the stratification diminishes. We use the proportional allocation where we set $N_k = N\mathbb{P}(|\rho'(\Phi_q)| \in C_k)$ per stratum k. The graphs[4] we use in this comparison can be listed as:

- **K-random regular**: A random regular graph with with $n = 10^4$ nodes and $m = 10^5$ edges generated by K-random regular model with the parameter $k = 20$,

- **Barabasi-Albert**: A random graph with $n = 10^4$ nodes and $m = 99900$ edges generated from a Barabasi-Albert model with the parameter $k = 10$.

- **Collab-CM**: A collaboration network of $n = 21363$ nodes and $m = 91342$ edges. Each node is an author in Arxiv on condense matter physics and each edge depicts their collaboration,

- **Citation-HEP**: A citation network with $n = 34401$ nodes and $m = 420828$ edges. Each node is a paper in Arxiv on high energy physics and there is an edge whenever a paper is cited by another,

- **3D-Grid**: A 3 dimensional grid graph with $n = 125000$ nodes and $m = 375000$,

- **Amazon**: A real-life network in Amazon with $n = 262111$ nodes and $m = 899792$ edges. Each node corresponds to a product and an edge between two products indicates that both products are bought by the same clients.

Fig. 4.3 summarizes our comparisons on effective runtimes while we vary q so that $\mathrm{tr}(K)/n$ varies approximately between 0.01 and 0.65.

In relatively small and sparse graphs, such as Collab-CM, the direct method (solving via sparse Cholesky decomposition) gives the best performance which is closely followed by RSF methods. When it comes to larger and denser graphs, the RSF methods often provide the best performance or a comparable performance with the state-of-the-art algorithms.

[4]The real-life data sets can be found in https://snap.stanford.edu/data/

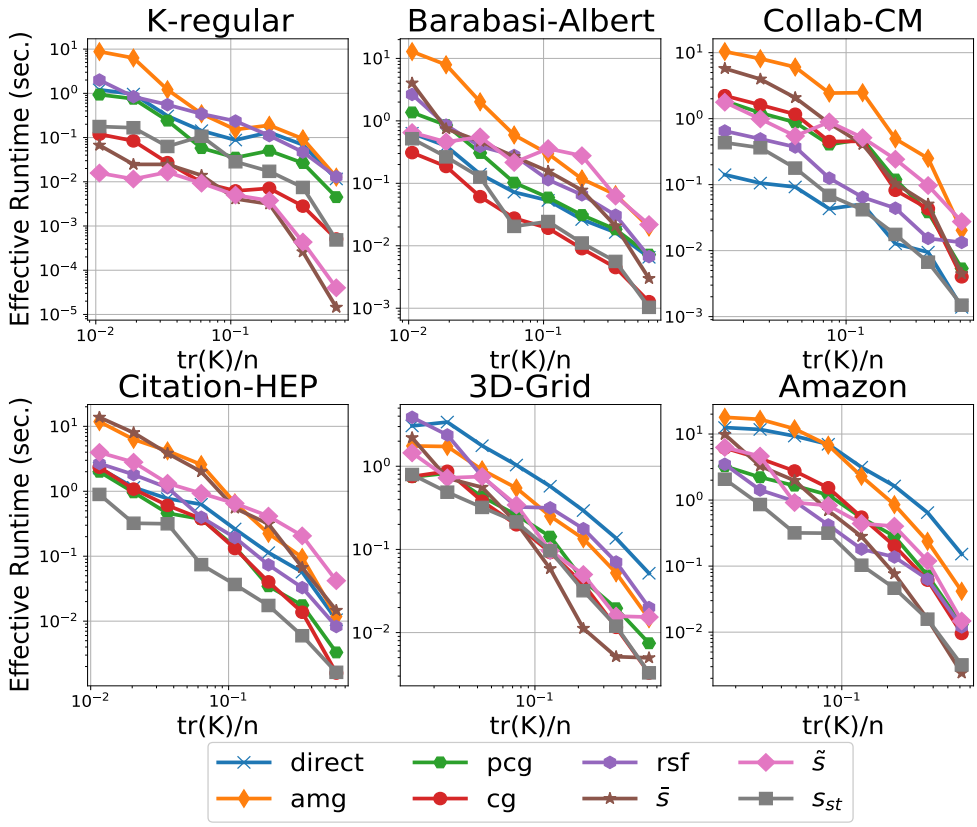


Fig. 4.3.: Effective Runtime vs $\mathrm{tr}(\mathrm{K})/n$.

4.2 Estimators for Effective Resistances

In the second part of this chapter, effective resistances, another significant set of graph quantities, will be the central object. They have important roles in many applications, yet they are not easy to access in the case of large graphs. In the following, after giving their formal definition and, certain properties, we focus on the problem of efficiently computing them. In doing so, we revisit existing algorithms and propose two forest-based estimators. We finish this section with the empirical comparison of these algorithms.

4.2.1 A Crash Course on Effective Resistances

A distance metric, *e.g.* the geodesic distance (the length of the shortest path between a pair of vertices), over the vertices of a graph describes the distance of each pair of vertices on the graph. Such metrics have a vast number of applications in network science. However, distance metrics may be defined in various ways [Sol11;

RR15], and there is no definitive answer as to which one is the best. Depending on the theory and the application that the metric is deployed (mostly the latter), network analysts may favour one over others. The effective resistances are a distance metric over vertices. Initially studied in electrical networks, the effective resistances have many theoretical links with random walks and USTs [LP16]. Therefore, they contain significant information related to the probabilistic and combinatorial properties of the graph. Motivated by these facts and many others, they have been used in many network science applications, including graph clustering [AALG17], sparsification [SS11] and learning [SXGRS20]. In this section, we detail their connections with USTs (and also RSFs) and give RSF-based methods for their numerical computation.

Let us start with the formal definition of effective resistances:

Definition 4.2.1 (Effective Resistances)**.** A resistor network with n nodes (connection points of resistors) can be depicted by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ where each edge is a resistor with a resistance value $1/w(i,j)$. The effective resistance $\mathsf{R}_{i,j}$ is the reciprocal of the current induced when unit potential difference/voltage is applied between i and j. It can be seen as *the resistance measured between node i and j*. One can calculate $\mathsf{R}_{i,j}$ in terms of the entries of $\mathsf{L}^\dagger$:

$$\mathsf{R}_{i,j} = \mathsf{L}^\dagger_{i,i} + \mathsf{L}^\dagger_{j,j} - \mathsf{L}^\dagger_{i,j} - \mathsf{L}^\dagger_{j,i}.$$

The last algebraic identity takes its roots from the combination of Ohm's law and Kirchoff's current law. In order to elaborate, let us denote the voltage at node i by v_i and the current at edge (i,j) by $c_{i,j}$. Applying a unit voltage between i and j means $v_i = 1$ and $v_j = 0$. Then, by the Ohm's law, one has $c_{x,y} = w(x,y)(v_x - v_y)$ for any pair $(x,y) \in \mathcal{E}$. In a matrix form, this constraint can be summarized as:

$$\mathsf{C}\mathsf{B}_u\mathbf{v} = \mathbf{c}. \tag{4.3}$$

where B_u is the unweighted edge incidence matrix and $\mathsf{C} \in \mathbb{R}^{m\times m}$ is the diagonal matrix which contains the edge weights in its diagonal entries. In addition, Kirchoff's current law indicates that the total current flow $f_x = \sum_{y\in\mathcal{N}(x)} c_{x,y}$ at every node $x \neq i,j$ equals to zero. The current flow f_i at node i (or f_j at node j) gives us the effective conductance, which is the reciprocal of effective resistance between i and j. Let us write the flow vector as $\mathbf{f} = \frac{1}{\mathsf{R}_{i,j}}(\delta_i - \delta_j)$ where $\delta_i = [\mathbb{I}(i=j)]_{j\in\mathcal{V}} \in \mathbb{R}^n$. Then one has:

$$\mathbf{f} = \mathsf{B}_u^\top\mathbf{c} = \mathsf{B}_u^\top\mathsf{C}\mathsf{B}_u\mathbf{v} = \mathsf{L}\mathbf{v}$$

Multiplying both sides with $\mathbf{f}^\top \mathsf{L}^\dagger$, one has:

$$\mathbf{f}^\top \mathsf{L}^\dagger \mathbf{f} = \mathbf{f}^\top \left(\mathsf{I} - \frac{1}{n} \mathsf{J} \right) \mathbf{v},$$

where $\mathsf{J} = \mathbf{1}\mathbf{1}^\top \in \mathbb{R}^{n \times n}$. Also by Kirchoff's current law $f_i = -f_j$ as the total current going in and out from the circuit must be the same. Then, one has $\mathbf{f}^\top \mathbf{1} = \mathbf{0}$ and $\mathbf{f}^\top \mathbf{v} = \frac{1}{\mathsf{R}_{i,j}}(v_i - v_j) = \frac{1}{\mathsf{R}_{i,j}}$:

$$\frac{1}{\mathsf{R}_{i,j}^2}(\mathsf{L}_{i,i}^\dagger + \mathsf{L}_{j,j}^\dagger - \mathsf{L}_{i,j}^\dagger - \mathsf{L}_{j,i}^\dagger) = \mathbf{f}^\top \mathbf{v} = \frac{1}{\mathsf{R}_{i,j}}$$

$$\mathsf{R}_{\mathsf{i,j}} = \mathsf{L}_{i,i}^\dagger + \mathsf{L}_{j,j}^\dagger - \mathsf{L}_{i,j}^\dagger - \mathsf{L}_{j,i}^\dagger$$

Along with these algebraic descriptions, effective resistances also have probabilistic properties. One well-known property, closely related to our work, connects effective resistances with RSTs:

Theorem 4.2.1 (Edges in RSTs)**.** *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, the probability of having an edge (i, j) in the RST, T reads:*

$$\mathbb{P}((i, j) \in T) = w(i, j)\mathsf{R}_{i,j}.$$

Proof. The RST T is a projection DPP over $\mathcal{E}$ that verifies for all $S \subset \mathcal{E}$:

$$\mathbb{P}(S \in T) = \det(\mathsf{B}(\mathsf{B}^\top \mathsf{B})^\dagger \mathsf{B})_S$$

Then writing the probability for a single edge $e = (i, j)$ finishes the proof:

$$\mathbb{P}(e \in T) = \det(\mathsf{B}(\mathsf{B}^\top \mathsf{B})^\dagger \mathsf{B})_e = \det(\mathsf{B}\mathsf{L}^\dagger \mathsf{B})_e = w(e)(\mathsf{L}_{i,i}^\dagger + \mathsf{L}_{j,j}^\dagger - \mathsf{L}_{i,j}^\dagger - \mathsf{L}_{j,i}^\dagger) = w(e)\mathsf{R}_{i,j}.$$

□

Due to all of these properties and the others omitted in this thesis, effective resistances have been playing a central role in many applications of network science. On the other hand, direct computation is demanding as it requires computing the entries of $\mathsf{L}^\dagger$ which is expensive when the graph is large. Therefore, the state-of-the-art algorithms are approximate methods that avoid this computation at the cost of precision. In the following sections, we visit some of these algorithms and propose two forest-based estimators for effective resistances.

4.2.2 Computing Effective Resistances

In the case of large graphs, the existing algorithms to compute $\mathsf{R}_{i,j}$'s are randomized approximate approaches. Each algorithm deploys different type of randomization in order to leverage probabilistic properties of effective resistances or to optimize a certain aspect of the algorithm. In the following, we list three algorithms and highlight their strengths and weaknesses:

- Estimation by Spanning Trees [HAY16]: By Theorem 4.2.1, we already know the direct link between RSTs and effective resistances. In [HAY16], the authors use this link to give an unbiased estimator for $\mathsf{R}_{i,j}$'s for all $(i,j) \in \mathcal{E}$. In a nutshell, after sampling N spanning trees $T_1, \ldots, T_N$ by Wilson's algorithm, this estimator computes:

$$\forall (i,j) \in \mathcal{E}, \quad \mathsf{R}^{st}_{i,j} := \frac{1}{N} \sum_{k=1}^{N} \frac{\mathbb{I}((i,j) \in T_k)}{w(i,j)},$$

  where $\mathbb{I}$ is the indicator function. The unbiasedness is easy to check via Theorem 4.2.1. By Hoeffding's inequality, the authors give the number of spanning trees to sample as $N = \frac{\log(2m/\delta)}{2\epsilon^2}$ to guarantee $|\mathsf{R}_{i,j} - \mathsf{R}^{st}_{i,j}| \leq \epsilon$ for each $(i,j) \in \mathcal{E}$ with the probability at least $1 - \delta$. However, this estimator restricted to estimate effective resistances only over the edges.

- Estimation by Random Projections [SS11]: This method suggests estimating $\mathsf{R}_{i,j}$ for all pairs of $(i,j) \in \mathcal{V}$ by using efficient Laplacian solvers [ST04] and Johnson-Lindenstrauss' lemma [Joh84]. The estimator takes three main steps:

  1. Sample a random matrix $\mathsf{A} \in \mathbb{R}^{k \times m}$ where $k = \mathcal{O}(\log n/\epsilon^2)$ for some error parameter $\epsilon > 0$ and each $\mathsf{A}_{i,j} \in \{-1/\sqrt{k}, -1/\sqrt{k}\}$ is a normalized Rademacher random variable.

  2. Compute $\mathsf{Y} = \mathsf{AB}$.

  3. For each row of Y, denoted by $\mathbf{y}_i^\top$, solve the linear system $\mathsf{L}\mathbf{z}_i = \mathbf{y_i}$ to compute $\mathsf{Z} = [\mathbf{z_1}| \ldots |\mathbf{z}_k]$.

  4. Finally, one can calculate the estimate $\mathsf{R}^{rp}_{i,j}$ for any pair $(i,j) \in \mathcal{V}$ as:

$$\mathsf{R}^{rp}_{i,j} := ||\mathsf{Z}(\delta_i - \delta_j)||_2^2.$$

  By the Johnson-Lindenstrauss lemma, this algorithm verifies:

$$\forall (i,j) \in \mathcal{V}, \quad (1-\epsilon)\mathsf{R}_{i,j} \leq \mathsf{R}^{rp}_{i,j} \leq (1+\epsilon)\mathsf{R}_{i,j}$$

In [SS11], the authors propose to approximate the inversion at step 3 with the algorithm proposed in [ST04]. This algorithm computes the estimates $\tilde{\mathbf{z}}_i$'s such that each verifies $||\tilde{\mathbf{z}}_i - \mathsf{L}^\dagger \mathbf{y}_i||_{\mathsf{L}} \leq \delta ||\mathsf{L}^\dagger \mathbf{y}_i||_{\mathsf{L}}$ for some $\delta > 0$ at the time complexity $\mathcal{O}(m \log^c n \log(1/\delta))$ for some constant $c > 1$. Then $\tilde{\mathsf{Z}}$ computed with the error parameter $\delta \leq \frac{\epsilon}{3}\sqrt{\frac{2(1-\epsilon)w_{min}}{(1+\epsilon)n^3 w_{max}}}$ [5] verifies:

$$\forall (i,j) \in \mathcal{V}, \quad (1-\epsilon)^2 \mathsf{R}_{i,j} \leq \tilde{\mathsf{R}}^{rp}_{i,j} \leq (1+\epsilon)^2 \mathsf{R}_{i,j}$$

where $\tilde{\mathsf{R}}^{rp}_{i,j} := ||\tilde{\mathsf{Z}}(\delta_i - \delta_j)||_2^2$. The authors develop on these facts and give the time complexity of computing $\tilde{\mathsf{Z}}$, and so the whole algorithm, as $\tilde{\mathcal{O}}(m \log r/\epsilon^2)$ where $r = w_{max}/w_{min}$ [6].

- Local Algorithms [PLYG21]: Closer to our work, the authors in [PLYG21] leverage the links of effective resistances with random walks, Markov chains and spanning trees. In turn, they propose a family of randomized algorithms for estimating $\mathsf{R}_{i,j}$'s. Given a pair of nodes s and t, these algorithms estimate $\mathsf{R}_{s,t}$ without needing the whole graph as the input. They operate locally *around* the nodes s and t (*e.g.* they use random walks that start from node s and stop at node t). For the detailed analysis of all of these algorithms, we refer the reader to [PLYG21]. In the experiments, we implement two of these algorithms (the most precise and the fastest) in order to compare them with the algorithms proposed in this manuscript.

This finishes our revisit on the SOTA algorithms for estimating $\mathsf{R}_{i,j}$'s. We note that all these algorithms are randomized, but they differ in their pros/cons. For example, the estimator $\mathsf{R}^{st}_{i,j}$ can approximate $\mathsf{R}_{i,j}$ only if $(i,j) \in \mathcal{E}$ whereas one can reach an arbitrary $\mathsf{R}_{i,j}$ with $i,j \in \mathcal{V}$ via random projections and some of the local algorithms[7]. In the following sections, we give the forest-based methods to estimate $\mathsf{R}_{i,j}$'s.

4.2.3 Estimating ER via 2-Rooted Forests

The first algorithm we propose is inspired by the following fact:

[5] w_{max} and w_{min} are maximum and minimum edge weights in the graph.
[6] The soft big-O notation $\tilde{\mathcal{O}}$ hides the term $\log^c n$
[7] In [PLYG21], the authors suggest using the local algorithms for a small number of vertex pairs

Lemma 4.2.2 ($\mathsf{R}_{i,j}$ as a determinantal ratio)**.** *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, one has:*

$$\forall i, j \in \mathcal{V}, \quad \mathsf{R}_{i,j} = \frac{\det \mathsf{L}_{-i,-j|-i,-j}}{\det \mathsf{L}_{-i|-i}} = \frac{\sum_{\phi \in \mathcal{F}_{\{i,j\}}} w(\phi)}{\sum_{\tau \in \mathcal{T}_i} w(\tau)}$$

where $\mathcal{T}_i$ is the set of all spanning trees rooted in i and $\mathcal{F}_{\{i,j\}}$ is the set of all spanning forests with exactly two disjoint trees rooted in i and j.

Proof. The second part of the identity is straight-forward by adapting Theorems 2.1.2 and 2.1.3. Then we prove this lemma by showing that $\mathsf{R}_{i,j}$ equals the ratio of these two determinants. In order to do so, we go back to electrical networks. Assume we apply the unit voltage between the nodes i and j in order to obtain the effective resistance $\mathsf{R}_{i,j}$. Recall that, by Ohm's law, one has the relation $c_{x,y} = w(x,y)(v_x - v_y)$ between the currents $c_{x,y}$ and voltages v_x, v_y at every edge (x,y). Moreover, by Kirchoff's law, the total current flow f_x at every node $x \in \mathcal{V} \backslash \{i,j\}$ is equal to 0. The current flow f_i gives the effective conductance $\frac{1}{\mathsf{R}_{i,j}}$. A matrix form [Bap10] (alternative to Eq. 4.3) to summarize these constraints is:

$$\mathsf{A}\mathbf{v} = \delta_i \text{ with } \mathsf{A} = \begin{bmatrix} \mathsf{L}_{-i,-j|:} \\ \delta_i^\top \\ \delta_j^\top \end{bmatrix} \in \mathbb{R}^{n \times n}$$

where $\delta_i = [\mathbb{I}(i=j)]_{j \in \mathcal{V}} \in \mathbb{R}^n$. By solving this linear system, one computes the voltages induced at every node via Cramer's rule as:

$$\forall k \in \mathcal{V} \setminus \{i,j\}, \quad v_k = \frac{(-1)^{i+k} \det \mathsf{L}_{-i,-j|-k,-j}}{\det \mathsf{L}_{-i,-j|-i,-j}}.$$

Then rewriting the flow at node i gives us:

$$\begin{aligned} f_i &= \sum_{k \in \mathcal{N}(i)} w(i,j)(v_i - v_k), \\ &= \sum_{k \in \mathcal{N}(i)} w(i,j) \left(1 - \frac{(-1)^{i+k} \det \mathsf{L}_{-i,-j|-k,-j}}{\det \mathsf{L}_{-i,-j|-i,-j}} \right), \\ &= \frac{\det \mathsf{L}_{-i|-i}}{\det \mathsf{L}_{-i,-j|-i,-j}}. \end{aligned}$$

Recalling $f_i = (\mathsf{R}_{i,j})^{-1}$ finishes the proof. □

Corollary 4.2.3 (2-Rooted Forests and ERs). *For any pair $i, j \in \mathcal{V}$, one has:*

$$\mathbb{P}(\rho(\Phi_q) = \{i, j\} || \rho(\Phi_q)| = 2) \propto \mathsf{R}_{i,j}.$$

Proof. The proof is easy by multiplying and dividing the determinantal ratio above by $q^2 \sum_{\phi \in \mathcal{F}_2} w(\phi)$:

$$\mathsf{R}_{i,j} = \frac{\sum_{\phi \in \mathcal{F}_2} w(\phi)}{\sum_{\tau \in \mathcal{T}_i} w(\tau)} \frac{q^2 \sum_{\phi \in \mathcal{F}_{\{i,j\}}} w(\phi)}{q^2 \sum_{\phi \in \mathcal{F}_2} w(\phi)} = \frac{\sum_{\phi \in \mathcal{F}_2} w(\phi)}{\sum_{\tau \in \mathcal{T}_i} w(\tau)} \mathbb{P}(\rho(\Phi_q) = \{i, j\} || \rho(\Phi_q)| = 2).$$

The proportionality follows by realizing that $\frac{\sum_{\phi \in \mathcal{F}_2} w(\phi)}{\sum_{\tau \in \mathcal{T}_i} w(\tau)}$ is invariant w.r.t. i and j[a]. □

[a] Recall that $\forall x, y \in \mathcal{V}, \sum_{\tau \in \mathcal{T}_x} w(\tau) = \sum_{\tau \in \mathcal{T}_y} w(\tau)$

Let us define the RSFs conditioned over $|\rho(\Phi_q)|$ as $\Phi_q^{(k)} := \Phi_q || \rho(\Phi_q)| = k$. and consider a random variable $X_{i,j} = \mathbb{I}(\rho(\Phi_q^{(2)}) = \{i, j\})$. By the corollary above, we have:

$$\mathbb{E}[X_{i,j}] = \mathbb{P}(\rho(\Phi_q^{(2)}) = \{i, j\}) \propto \mathsf{R}_{i,j}.$$

Then, one can estimate $\mathsf{R}_{i,j}$ with $X_{i,j}$ up to a constant. Let $\mathsf{X} = [X_{i,j}]_{i,j \in \mathcal{V}}$ be the matrix form of this estimator. By definition, only non-zero entries of X are $\mathsf{X}_{i,j} = \mathsf{X}_{j,i} = 1$ and the rest is equal to 0. In fact, this estimation can be improved by deploying the conditional Monte Carlo technique used in 3.3.1. Recall Prop. 2.4.6 which states that the distribution of roots is uniform Φ_q when the partition is fixed. Then an improved version of X with the conditional Monte Carlo method is:

$$\forall i, j \in \mathcal{V}, \quad \bar{\mathsf{X}}_{i,j} := \begin{cases} \frac{1}{|\mathcal{V}_{t(\pi(\Phi_q^{(2)}, i)}||\mathcal{V}_{t(\pi(\Phi_q^{(2)}, j)}|} & \text{if } r_{\Phi_q^{(2)}}(i) \neq r_{\Phi_q^{(2)}}(j) \\ 0 & \text{otherwise} \end{cases}. \tag{4.4}$$

The matrix X for a given spanning forest with two trees.

The estimator $\bar{\mathsf{X}}$ admits the same expectation as X by the law of iterated expectation, and it gives a better estimation than X by the law of total variance. Alternatively, for a 2-rooted spanning forests with two parts V_1 and V_2, one can write $\bar{\mathsf{X}}$ a low-rank representation as:

$$\bar{\mathsf{X}} = \frac{1}{|V_1||V_2|} \left(\mathbf{1}_{V_1} \mathbf{1}_{V_2}^\top + \mathbf{1}_{V_2} \mathbf{1}_{V_1}^\top \right), \tag{4.5}$$

where $\mathbf{1}_{V_1} = [\mathbb{I}(i \in V_1)]_i$ and $\mathbf{1}_{V_2} = [\mathbb{I}(i \in V_2)]_i$. On the other hand, $\bar{\mathsf{X}}_{i,j}$ can estimate $\mathsf{R}_{i,j}$ up to a graph related constant c, *i.e.* $\mathbb{E}[\bar{\mathsf{X}}] = c\mathsf{R}$ where,

$$c = \frac{\sum_{\tau \in \mathcal{T}_i} w(\tau)}{\sum_{\phi \in \mathcal{F}_2} w(\phi)}.$$

We already know that this constant is invariant in i and j. In fact, c here can be written in terms of the effective resistances as follows:

$$c = \frac{\sum_{\tau \in \mathcal{T}_i} w(\tau)}{\sum_{\phi \in \mathcal{F}_2} w(\phi)} = \left(\sum_{i,j \in \mathcal{V}} \frac{\sum_{\phi \in \mathcal{F}_{i,j}} w(\phi)}{\sum_{\tau \in \mathcal{T}_i} w(\tau)} \right)^{-1} = \left(\sum_{i,j \in \mathcal{V}} \mathsf{R}_{i,j} \right)^{-1}.$$

Indeed, this quantity is solely related to the graph itself. In fact, it is the reciprocal of the Kirchoff's index, that is often used to measure the stability of a network [EK13]. However, it is not easy to compute directly. In order to circumvent this issue, we propose to use the following identity (Foster's theorem [Fos49]):

$$\sum_{(i,j) \in \mathcal{E}} w(i,j)\mathsf{R}_{i,j} = n - 1.$$

As this identity holds for all graphs, an unbiased estimator for c is:

$$C = \frac{\sum_{(i,j) \in \mathcal{E}} w(i,j)\bar{\mathsf{X}}_{i,j}}{n-1}$$

Then we define a ratio estimator as follows:

$$\mathsf{R}^{2rf}_{i,j} := \frac{\bar{\mathsf{X}}_{i,j}}{C}.$$

This estimator, like many other ratio estimators, is biased in most cases.

Bias analysis. The bias can be approximated by Taylor expansion on $\mathbb{E}[\mathsf{R}^{2rf}_{i,j}]$ (at second order of derivatives):

$$\mathbb{E}[\mathsf{R}^{2rf}_{i,j}] - \mathsf{R}_{i,j} \approx -\frac{\mathrm{Cov}(\bar{\mathsf{X}}_{i,j}, C)}{\mathbb{E}[C]^2} + \frac{\mathbb{E}[\bar{\mathsf{X}}_{i,j}]}{\mathbb{E}[C]^3}\mathrm{Var}(C).$$

This bias usually diminishes at the rate of $\mathcal{O}(N^{-1})$ as N grows [8]. Moreover, one can correct the bias by estimating it from the samples by the sample covariance and variance formulas and subtracting it from the estimate. Many recent works [BCG15; MKJ19] propose various ways of improvement (including unbiased estimators), taking extra computational time.

[8] Here, we compute the sample mean for $\bar{\mathsf{X}}$ and C separately from i.i.d. samples and compute the ratio.

Variance analysis. The variance of the estimator is the other main (usually the most important) element of the approximation error of the proposed estimator. Similar to the bias, it can be approximated by Taylor expansion as follows:

$$\begin{aligned}\operatorname{Var}(\mathsf{R}_{i,j}^{2rf}) &\approx \frac{\operatorname{Var}(\bar{\mathsf{X}}_{i,j})}{\mathbb{E}[C]^2} - 2\frac{\mathbb{E}[\bar{\mathsf{X}}_{i,j}]}{\mathbb{E}[C]^3}\operatorname{Cov}(\bar{\mathsf{X}}_{i,j}, C) + \frac{\mathbb{E}[\bar{\mathsf{X}}_{i,j}]^2}{\mathbb{E}[C]^4}\operatorname{Var}(C) \\ &= \frac{1}{c^2}(\operatorname{Var}(\bar{\mathsf{X}}_{i,j}) - 2\mathsf{R}_{i,j}\operatorname{Cov}(\bar{\mathsf{X}}_{i,j}, C) + \mathsf{R}_{i,j}^2\operatorname{Var}(C))\end{aligned} \tag{4.6}$$

In order to better understand this variance expression, we derive an upper-bound in terms of graph-related constants such as n, m, c or $\mathsf{R}_{i,j}$. In doing so, instead of an individual pair (i,j), we consider the sum of the variance over all pairs $\sum_{i,j\in\mathcal{V}}\operatorname{Var}(\mathsf{R}_{i,j}^{2rf})$ to get a global sense. As implied in Eq. 4.6, $\sum_{i,j\in\mathcal{V}}\operatorname{Var}(\mathsf{R}_{i,j}^{2rf})$ consists of three additive terms. For each, we find the following upper-bounds (not necessarily tight) for unweighted graphs:

- $\sum_{i,j\in\mathcal{V}}\operatorname{Var}(\bar{\mathsf{X}}_{i,j}) \leq \frac{2}{n-1} - c^2\sum_{i,j}\mathsf{R}_{i,j}^2$,
- $-2\sum_{i,j\in\mathcal{V}}\mathsf{R}_{i,j}\operatorname{Cov}(\bar{\mathsf{X}}_{i,j}, C) \leq 2c^2\sum_{i,j}\mathsf{R}_{i,j}^2$,
- $\sum_{i,j\in\mathcal{V}}\mathsf{R}_{i,j}^2\operatorname{Var}(C) \leq \left(\frac{m^2}{(n-1)^4} - c^2\right)\sum_{i,j\in\mathcal{V}}\mathsf{R}_{i,j}^2$.

The derivations can be found in App. A.5. Putting all of these together yields that the total variance is bounded:

$$\sum_{i,j\in\mathcal{V}}\operatorname{Var}(\mathsf{R}_{i,j}^{2rf}) \lessapprox \frac{1}{c^2}\left(\frac{2}{n-1} + \left[\frac{m^2}{(n-1)^4}\right]\sum_{i,j\in\mathcal{V}}\mathsf{R}_{i,j}^2\right).$$

This bound indicates that for larger $c = \frac{1}{\sum_{i,j\in\mathcal{V}}\mathsf{R}_{i,j}}$, the total variance tends to be smaller. This indicates that R^{2rf} tends to perform better when the effective resistances are small all over the graph.

Although R^{2rf} estimates all pairs over a single forest, in practice it yields a poor performance compared to other algorithms. In the following, we lists several reasons and possible solutions to improve the performance by going beyond the work done in this thesis:

- **Recycling forests with 1,3,4 trees.** In order to sample $\bar{\mathsf{X}}$ and C, one needs to sample spanning forests with exactly two trees for a fixed value of q. However, for an arbitrary value of q, sampling a two-rooted forest might be a rare event. In order to find the proper value of q, [ACGM18] suggests using an *annealing process*. This process iteratively decreases the value of q until the desired number of roots is approximately reached. However, even plugging this value,

one might still end up with many forests with 1,3,4 trees which cannot be used in $\bar{\mathsf{X}}$ and C. The up-down sampler given in [RTF18] may be a possible fix for this issue. Originally designed for sampling uniform spanning trees, the up-down sampler for RSFs takes a Markov chain whose states are the spanning forests of the graph (or the spanning trees on the extended graph with Γ), and stationary distribution is the distribution of Φ_q. There are three types of transitions authorized between the states/forests; adding and deleting an edge, adding an edge and deleting a root and finally adding a root and deleting an edge. Setting the transition probabilities accordingly, one can guarantee the stationary distribution is the same with the distribution of Φ_q. An MCMC (Markov Chain Monte Carlo) algorithm to sample from the distribution of Φ_q starts from an arbitrary state in this Markov chain and run a random walk. After taking a sufficient number of steps, the random walk is stopped and the current state is distributed similar to Φ_q. An alternative use of this random walk may allow us to transform the spanning forests with 1,3,4 roots to the spanning forests 2 roots. Instead of an arbitrary state, we start with a spanning forest with 1,3,4 roots sampled via Wilson's algorithm and do the random walk until reaching a spanning forests with two trees. As we sample our initial state from the stationary distribution, the final step we reach by the random walks is also distributed according the stationary distribution. In this way, one recycles the unused samples in computing $\bar{\mathsf{X}}$ and C. One of the main difficulties with this idea is that one might need to take too many steps until reaching a two-rooted forest depending on the graph. Also, one needs to implement complex data structures to maintain transitions efficiently as proposed in [RTF18].

- **Variance Reduction.** High variance is the other main issue with this estimator. One may attempt to solve this issue by variance reduction techniques with the help of rich machinery of statistics in RSFs. A particular example that we come up with is the adaptation of the control variate method. In this case, we use the following relation [GX04] between $\mathsf{L}^\dagger$ and R:

$$\mathsf{L}^\dagger = -\frac{1}{2}\left(\mathsf{R} - \frac{1}{n}(\mathsf{RJ} + \mathsf{JR}) + \frac{1}{n^2}\mathsf{JRJ}\right). \tag{4.7}$$

  Multiplying both sides by L, one has:

$$\left(\mathsf{I} - \frac{1}{n}\mathsf{J}\right) = -\frac{1}{2}\left[\mathsf{LR}\left(\mathsf{I} - \frac{1}{n}\mathsf{J}\right)\right].$$

  By plugging R^{2rf} instead of R, one can show that the expectation $\mathbb{E}\left[\mathsf{LR}^{2rf}\left(\mathsf{I} - \frac{1}{n}\mathsf{J}\right)\right]$ converges to a known matrix $2\left(\frac{1}{n}\mathsf{J} - \mathsf{I}\right)$. Thus one can use $\mathsf{LR}^{2rf}\left(\mathsf{I} - \frac{1}{n}\mathsf{J}\right)$ as

a control variate. Moreover, by the low-rank representation of $\bar{\mathsf{X}}$ in Eq. (4.5), computing the matrix product $\mathsf{L}\mathsf{R}^{2rf}$ boils down to cheap matrix vector products.

At this moment, unfortunately, R^{2rf} yields poor estimator. Besides its simplicity, its main advantage is that given a spanning forest, it provides estimates for all pairs of effective resistances. Nevertheless, its performance may be improved when the suggested fixes above are applied which we keep as the future work. In the next section, we give another estimator for effective resistances. In contrast to R^{2sf}, this one runs locally and gives an estimate for a single $\mathsf{R}_{i,j}$ at each run.

4.2.4 Estimating ER via Local Forests

In the proof of Lemma 4.2.3, we compute the effective conductance $\mathsf{I}_{i,j} = (\mathsf{R}_{i,j})^{-1}$ as follows:

$$\mathsf{I}_{i,j} = \sum_{k \in \mathcal{N}(i)} w(i,k) \left(1 - \frac{(-1)^{i+k} \det \mathsf{L}_{-i,-j|-k,-j}}{\det \mathsf{L}_{-i,-j|-i,-j}}\right). \tag{4.8}$$

A closer look on the determinant ratio gives us with Theorem 2.1.3 and Lemma A.4.1:

$$\begin{aligned}
\frac{(-1)^{i+k} \det \mathsf{L}_{-i,-j|-k,-j}}{\det \mathsf{L}_{-i,-j|-i,-j}} &= \frac{\prod_{\phi \in \mathcal{F}_{k \to i,j}} w(\phi)}{\prod_{\phi \in \mathcal{F}_{i,j}} w(\phi)} \\
&= \frac{q^2 \prod_{\phi \in \mathcal{F}_{k \to i,j}} w(\phi)}{q^2 \prod_{\phi \in \mathcal{F}_{i,j}} w(\phi)} \\
&= \frac{\mathbb{P}(r_{\Phi_q}(k) = i)}{\mathbb{P}(\rho(\Phi_q) = \{i,j\})} \\
&= \mathbb{P}(r_{\Phi_q}(k) = i | \rho(\Phi_q) = \{i,j\}).
\end{aligned}$$

where $\mathcal{F}_{k \to i,j}$ is the set of forests with exactly two roots i and j and the node $k \neq i,j$ is rooted in i. Then, plugging this probability in Eq. 4.8 gives:

$$\begin{aligned}
\mathsf{I}_{i,j} &= \sum_{k \in \mathcal{N}(i)} w(i,k)(1 - \mathbb{P}(r_{\Phi_q}(k) = i | \rho(\Phi_q) = \{i,j\})) \\
&= \sum_{k \in \mathcal{N}(i)} w(i,k) \mathbb{P}(r_{\Phi_q}(k) = j | \rho(\Phi_q) = \{i,j\})
\end{aligned}$$

Let us denote the conditional random forest $\Phi_q | \rho(\Phi_q) = \{i, j\}$ by $\Phi_{\{i,j\}}$. Then, an unbiased estimator for $\mathsf{I}_{i,j}$ is:

$$\tilde{\mathsf{I}}_{i,j} := \sum_{k \in \mathcal{N}(i)} w(i,k) \mathbb{I}(r_{\Phi_{\{i,j\}}}(k) = j).$$

Based on this, we define a reciprocal estimator for $\mathsf{R}_{i,j}$ over N samples as follows:

$$\mathsf{R}^{lf}_{i,j} := \frac{N}{\sum_{k=1}^{N} \tilde{\mathsf{I}}^{(k)}_{i,j}}.$$

The samples from $\Phi_{\{i,j\}}$ can be generated by setting $q_{i,j} \to \infty$ and the rest to zero in Alg. 3. Moreover, one does not need to sample a complete spanning forest to compute $\tilde{\mathsf{I}}_{i,j}$. Whenever the nodes in $\mathcal{N}$ are rooted, one can compute $\tilde{\mathsf{I}}_{i,j}$ without waiting all the nodes are rooted, hence the name. Therefore, in practice, we initiate the LERWs in Alg. 3 from $\mathcal{N}(i)$ and interrupt it whenever all the nodes in $\mathcal{N}(i)$ are rooted. We summarize this implementation for this estimator in Alg. 4.

Algorithm 4 EstimateERviaLocalForests

Inputs:
$\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, $\{i, j\} \in \mathcal{V}$,N
Initialize:
$\tilde{\mathsf{I}}_{i,j} \leftarrow 0$
$\forall k \in \mathcal{V} \setminus \{i, j\}$, $q_k \leftarrow 0$
$q_i = q_j \to \infty$
for $i = 1 : N$ **do**
Run `RandomForest` on $\mathcal{G}$ by launching LERWs from $\mathcal{N}(i)$.
Interrupt the algorithm whenever all the nodes in $\mathcal{N}(i)$ are rooted, yielding a forest ϕ (not necessarily spanning).
$\tilde{\mathsf{I}}_{i,j} \leftarrow \tilde{\mathsf{I}}_{i,j} + \sum_{k \in \mathcal{N}(i)} w(i,k) \mathbb{I}(r_\phi(k) = j)$
end for
return $N / \tilde{\mathsf{I}}_{i,j}$.

Indeed, in this form, R^{lf} is a biased estimator as:

$$\forall i, j \in \mathcal{V}, \quad \mathbb{E}[\mathsf{R}^{lf}_{i,j}] = \mathbb{E}\left[\frac{N}{\sum_{k=1}^{N} \tilde{\mathsf{I}}^{(k)}_{i,j}}\right] \neq \frac{N}{\mathbb{E}[\sum_{k=1}^{N} \tilde{\mathsf{I}}^{(k)}_{i,j}]}.$$

The bias here can be approximated via Taylor expansion on $\mathbb{E}[\mathsf{R}^{lf}_{i,j}]$:

$$\mathbb{E}[\mathsf{R}^{lf}_{i,j}] - \mathsf{R}_{i,j} \approx \frac{1}{\mathbb{E}[\tilde{\mathsf{I}}_{i,j}]^3} \frac{\mathrm{Var}(\tilde{\mathsf{I}}_{i,j})}{N}$$

By bounding $\mathrm{Var}(\tilde{\mathsf{I}}_{i,j}) = \mathbb{E}[\tilde{\mathsf{I}}_{i,j}^2] - \mathbb{E}[\tilde{\mathsf{I}}_{i,j}]^2 \leq d_i^2 - \mathsf{I}_{i,j}^2$, one has a bounded bias:

$$\mathbb{E}[\mathsf{R}_{i,j}^{lf}] - \mathsf{R}_{i,j} \lessapprox \frac{1}{N\mathsf{I}_{i,j}^3}(d_i^2 - \mathsf{I}_{i,j}^2).$$

Moreover, the variance can be approximately bounded as follows: We complete our theoretical analysis on $\mathsf{R}_{i,j}^{lf}$ by noting its variance. The Taylor expansion on the variance gives us:

$$\mathrm{Var}(\mathsf{R}_{i,j}^{lf}) \approx \frac{1}{N}\left(\frac{\mathrm{Var}(\tilde{\mathsf{I}}_{i,j})}{\mathsf{I}_{i,j}^4}\right) \leq \frac{1}{N}\left(\frac{d_i^2 - \mathsf{I}_{i,j}^2}{\mathsf{I}_{i,j}^4}\right).$$

Notice that the bias diminishes at the rate of $\mathcal{O}(N^{-1})$ compared to $\mathcal{O}(N^{-1/2})$ for the standard deviation. Thus, this estimator converges to the true expectation $\mathsf{R}_{i,j}$ for large N. Nevertheless, one can decrease or completely eliminate the bias by making some extra effort. A simple correction is to estimate the bias term from the samples via the sample mean and variance formulas and subtract it. If the time budget is sufficient enough, one can deploy more advanced methods [MKJ19] that can provide an unbiased estimation. Yet, in certain cases, they might increase the variance of the estimator. In the experiments of the next section, we restrict ourselves to simple bias correction for the sake of simplicity. Yet we finish this section by stressing that there is still room for improvement.

4.2.5 Empirical Results on ER Estimation

In this section, we empirically compare the forest-based estimator R^{lf} with the state-of-the-art algorithms in terms of approximation error and run-time. However, a fair comparison is not straight-forward as some algorithms are more favorable for estimating a small number of effective resistances while the others are dedicated to estimate effective resistances in a global sense (over all edges or all pairs). Therefore, we first divide the compared algorithms into two groups called local and global. The local algorithms run on a small portion of the graph and estimate efficiently $\mathsf{R}_{i,j}$ for a given pair (i, j). In this group, we have algorithms 1 and 4 in [PLYG21], namely `EstEff-TranProb` (TP) and `EstEff-MC2` (MC2), and the estimation via local forests (LF) of Section 4.2.4. The global algorithms approximate the effective resistances over all edges or all pairs by operating over the whole graph. The run-time of these algorithms do not depend on the number of effective resistances demanded. Therefore, the number of estimated effective resistances grow, these algorithms become more favorable compared to the local algorithms. In this group, we put

the spanning tree algorithm by [HAY16] (ST) and the random projection algorithm by [SS11] (RP).

Notice that all algorithms in this comparison are Monte Carlo algorithms and each has a parameter that adjusts the number of Monte Carlo trials. For global algorithms ST and RP, these parameters are respectively the number of sampled spanning trees N_{st} and the dimension of the projection space k. We start our experimentation by estimating effective resistances via these algorithms over all edges. In doing so, we vary the parameters N_{st} and k and measure the total run-time and the average relative error $\frac{1}{m}\sum_{(i,j)\in\mathcal{E}} |\mathsf{R}'_{i,j} - \mathsf{R}_{i,j}|/\mathsf{R}_{i,j}$ of the algorithms. This gives us the performance of the global algorithms in estimating a single effective resistance. For the local algorithms TP, MC2 and LF, we run them over 30 randomly selected edges (with replacement) in order to estimate the corresponding effective resistances. Finally, we compute their average relative error and the average run-time over the edges in order to deliver their performance over a single effective resistance. Their parameters are tuned for each edge as follows; we first run LF for 40 local forests, then we tune the parameter l in TP and ϵ in MC2 in order to reach the relative error given by LF. This allows us to compare the run-times of local algorithms for the approximately same relative error. In order to compare them with the global algorithms, we calculate a number called threshold number N_{th}. This is the maximum number of effective resistances that can be estimated by a local algorithm before reaching the run-time of the fastest global algorithm at the same relative error. By looking at this metric, one can answer the following question; *For how many effective resistances, the local algorithms are more useful than the global ones?* As a result, the larger this metric for a local algorithm the better the performance.

In Fig. 4.4, we present our results over 4 benchmark datasets which are listed as:

- Citeseer: A citation network with $n = 2110$ nodes and $m = 3668$ and edges,
- Cora: A citation network with $n = 2485$ nodes and $m = 5069$ edges,
- Pubmed: A citation network with $n = 19717$ nodes and $m = 44324$ edges,
- Collab-CM: A collaboration network with $n = 21363$ and $m = 91342$.

First of all, we observe that over all graphs, ST provides the best performance within global algorithms in estimating effective resistances over all edges. Note that both methods inherently benefit from the sparsity of the graph. In case of RP, the sparse approximate Cholesky decomposition exploits the sparsity to facilitate its intermediate operations. For ST, Wilson's algorithm generates spanning trees by taking much less time in sparse graphs. The difference we observe in these

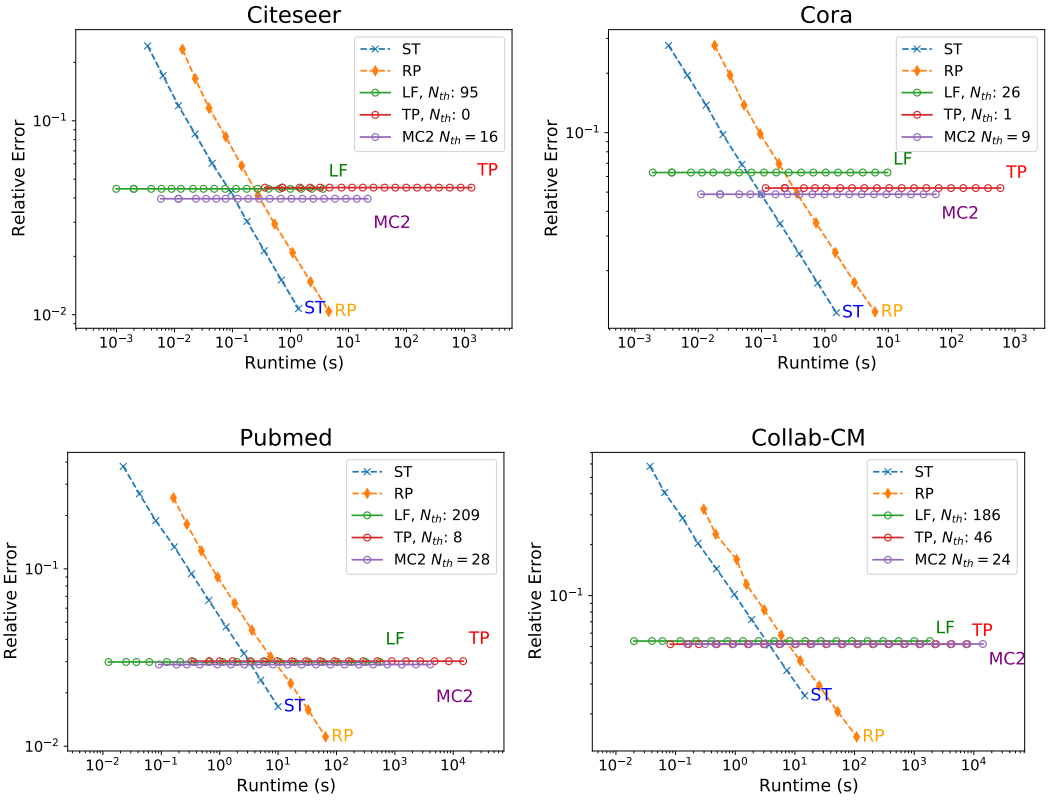


Fig. 4.4.: *Relative error vs. runtime of the global (ST and RP) and local (LF, TP, MC2) algorithms for estimating effective resistances. For the global algorithms, we run them by varying their iteration parameter and measure their average relative error over all edges and total run-time. For the local algorithms, we measure their average relative error and run-time for a single edge which is given in their plots as the leftest marker. We compute the time taken by each local algorithms for estimating k effective resistances by multiplying their run-time for a single estimation by k. By varying k between 1 and m, we generate the points in the plots of the local algorithms.*

simulations is probably due to the low variance of ST for estimating large effective resistances. One can observe this on the extreme case. In this case, we estimate an effective resistance $\mathsf{R}_{i,j}$ where i and j is connected only by a single edge. By definition, we find the true value of $\mathsf{R}_{i,j}$ as 1. Also, the estimation by ST always equals to 1 as this edge has to appear in all spanning trees. Therefore, it has zero variance. On the other hand, this is not necessarily the case for RP as the randomization we use here is independent from the graph. Therefore, the estimation given by RP still might have variance around the true value of $\mathsf{R}_{i,j}$. We see that there are many edges that fit this description on these graphs and thus we observe a significant difference between ST and RP. However, one needs to keep in mind that ST is not defined for the pairs $(i,j) \notin \mathcal{E}$ whereas RP can estimate those effective resistances.

Secondly, all local algorithms are usually more efficient compared to the global ones for estimating a single effective resistances. However, in the global sense, they cannot provide the same efficiency. Thus, we report their threshold numbers N_{th} in order to say up to how many edges they are still useful. By looking at the estimation over a single edge and the threshold numbers, we see that the forest-based algorithm LF often gives the best performance. Compared to other local algorithms TP and MC2, it takes much less time for the same relative error over a single estimation, and it gives a threshold number that is at least 4-5 times that of the closest local algorithm. As a result, LF gives a way to estimate approximately $0.5-2\%$ of the effective resistance over all edges without being slower than the global algorithms.

4.3 Graph Filtering with RSFs

Graph signal filtering is an essential tool in GSP. However, the necessary computations, such as computing the graph Fourier basis U by diagonalizing the graph Laplacian $\mathsf{L} = \mathsf{U}\Lambda\mathsf{U}^\top$, are not scalable with the size of the graph. The RSF estimators $\tilde{\mathbf{x}}$, $\bar{\mathbf{x}}$ and $\bar{\mathbf{z}}$ avoid the heavy computations for approximating a particular graph filtering where the transfer function is:

$$g_q(\lambda) = \frac{q}{q+\lambda} \tag{4.9}$$

where λ is the graph frequency parameter. Indeed filtering with $g_q(\lambda)$ is a very useful operation as it returns a *smooth* signal on the graph. Yet depending on the application, different transfer functions might be desired. For example, the ideal low-pass filters are more favorable if the noise is known to be contained in the high

frequency components. In these filters, the transfer function takes the following form:

$$g_k(\lambda) = \begin{cases} 1, & \text{if } \lambda \leq \lambda_k \\ 0, & \text{otherwise} \end{cases} \tag{4.10}$$

where λ_k is the k-th smallest eigenvalue of L.

When applied, this filter completely removes the components associated with the frequencies $\lambda > \lambda_k$ from the signal. If the noise is known to be contained in those frequency components, one can perfectly reconstruct the original signal. The step function behaviour of $g_k(\lambda)$ is desired in many applications. In our case, we may try to approximate $g_k(\lambda)$ by $g_q(\lambda)$ with the forest estimators by optimizing q such that $g_q(\lambda)$ gives the greatest decrease at λ_k. Even so, $g_q(\lambda)$ is a very poor approximation.

In this section, we focus on the graph filtering other than $g_q(\lambda)$ that can be approximated by the forest-based estimators. In doing so, we first give a forest-based method to solve the Laplacian systems in the form of $\mathsf{L}\mathbf{x} = \mathbf{y}$ which corresponds to low-pass filtering. Then, we switch our focus to another setup where we find a family of graph filters estimated by RSFs. In this setup, we duplicate the graph and add edges between the nodes of the original and copy graph. For certain ways of creating such graphs, the RSF methods yield a set of graph filters different from $g_q(\lambda)$. Instead of a single hyper-parameter as in $g_q(\lambda)$, these filters come with four parameters, yielding wider options for filtering. We illustrate them in a case study where we approximate some of the typical graph filters $g_k(\lambda)$ or,

$$g_{q,p}(\lambda) = \frac{q}{q + \lambda^p}, \tag{4.11}$$

where $p > 1$ is an integer.

4.3.1 Solving $\mathsf{L}\mathbf{x} = \mathbf{y}$ with RSFs

Linear systems of the form $\mathsf{L}\mathbf{x} = \mathbf{y}$ where L is a graph Laplacian are an important class of systems of linear equations. Apart from their direct use in electrical networks, they also appear in the spring networks and maximum flow problems [Vis+13]. It is possible to see the solution $\mathbf{x} = \mathsf{L}^\dagger\mathbf{y}$ as a graph filter where the transfer function is:

$$g^\dagger(\lambda) = \begin{cases} 0 & \lambda = 0 \\ \frac{1}{\lambda} & \text{otherwise} \end{cases}.$$

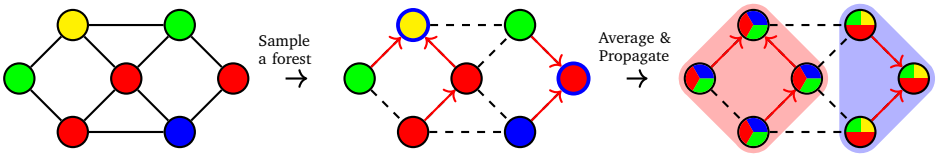


Fig. 4.5.: An illustration of the averaging operation in $\mathbf{x}^{2rf}$. After averaged in one tree, the signal is propagated in the other tree.

Yet again, directly computing $\mathbf{x}$ ($\mathcal{O}(n^3)$) is not practical for very large graphs. In order to avoid this, one can use classical methods for solving linear systems, such as iterative methods. The state-of-the-art algorithms solve the Laplacian systems in quasi-linear time in the number of edges, more specifically $\mathcal{O}(m \log^c n \log \frac{1}{\epsilon})$ where c is a constant and ϵ is the error parameter [ST04]. In this section, we show how RSFs can be leveraged for approximating $\mathsf{L}^\dagger\mathbf{y}$. However, the resulting method at its current state is not competitive with SOTA algorithms.

The forest estimator we find for $\mathsf{L}^\dagger\mathbf{y}$ in fact, takes its root from the following relation between $\mathsf{L}^\dagger$ and the effective resistance matrix R:

$$\mathsf{L}^\dagger = -\frac{1}{2}\left(\mathsf{R} - \frac{1}{n}(\mathsf{RJ} + \mathsf{JR}) + \frac{1}{n^2}(\mathsf{JRJ})\right), \tag{4.12}$$

where $\mathsf{J} = \mathbf{1}\mathbf{1}^\top \in \mathbb{R}^{n\times n}$. Then, whenever one has an estimate $\tilde{\mathsf{R}}$ of R, the product $\mathsf{L}^\dagger\mathbf{y}$ can be approximated by matrix-vector products with $\tilde{\mathsf{R}}$. Interestingly, the forest estimator R^{2rf} may provide such products very cheaply. By definition R^{2rf} is a symmetric matrix with two non-zero blocks in its off-diagonals. Then, the product $\mathbf{x}^{2rf} := (\mathsf{R}^{2rf}\mathbf{y})$ with an arbitrary vector $\mathbf{y}$ reads:

$$\forall i \in \mathcal{V}, \quad \mathbf{x}_i^{2rf} = \begin{cases} \frac{\sum_{j\in\mathcal{V}} \mathbf{y}_j \mathbb{I}(r_{\Phi_q}(i) \neq r_{\Phi_q}(j))}{C|\mathcal{V}_{t(\Phi_q,r_1)}||\mathcal{V}_{t(\Phi_q,r_2)}|} & \text{if } \rho(\Phi_q) = \{r_1, r_2\} \\ 0 & \text{otherwise} \end{cases}$$

where r_1 and r_2 are the random roots of Φ_q. In other words, given a forest ϕ with two roots, it computes a scaled average of $\mathbf{y}$ within the tree that i does not belong to, and propagates it through the tree of i (See Fig. 4.5). Assuming $\mathbf{1}^\top\mathbf{y} = 0$ without loss of generality[9] and rearranging on Eq. (4.12) allows us to define the estimator for $\mathsf{L}^\dagger\mathbf{y}$ as follows:

$$\mathbf{l} := -\frac{1}{2}\left(\mathsf{I} - \frac{1}{n}\mathsf{J}\right)\mathbf{x}^{2rf}.$$

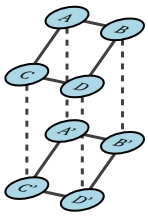
In practice, this estimator computes $\mathbf{x}^{2rf}$, removes its mean and scales the result by $-1/2$. This ensures that the estimates given by $\mathbf{l}$ are zero-mean vectors similar to $\mathsf{L}^\dagger\mathbf{y}$. As defined above, the random vector $\mathbf{x}^{2rf}$ takes a two-tree spanning forest

[9] Note that $\mathsf{L}^\dagger(\mathbf{y} + \mu\mathbf{1}) = \mathsf{L}^\dagger\mathbf{y}$ for any $\mu \in \mathbb{R}$. We choose to assume $\mathbf{y}$ is zero-mean.

and computes the averages within each tree. Unlike the previous estimator $\bar{\mathbf{x}}$, it swaps the scaled averages between trees and then propagates. At first sight, the swapping operation might seem counter-intuitive as we do not have such behaviour in the estimated product $\mathsf{L}^\dagger\mathbf{y}$. In fact, this situation can be explained by examining the overall process. First, notice that for $\mathbf{1}^\top\mathbf{y} = 0$, the scaled averages computed in two trees $\bar{y}_1$ and $\bar{y}_2$ verify $\bar{y}_1 = -\bar{y}_2$. Then, swapping the scaled averages across the trees corresponds to changing only the signs of the values that are propagated. Later on, in calculating $\mathbf{l}$, we multiply $\mathbf{x}^{2rf}$ by $-\frac{1}{2}$, which reverts the sign change that is induced by the swapping.

The intrinsic weakness of this estimator is due to the high variance of R^{2rf}. Due to this fact, we do not expect a good performance in estimating $\mathsf{L}^\dagger\mathbf{y}$. Nevertheless, one may improve this estimator by improving the performance of R^{2rf} by taking our suggestions in Section 4.2.3.

4.3.2 Filters on the Duplicated Graph



We copy the graph and add edges between the nodes of the original and copy graph

Let us start by describing how we duplicate the graph. Given the adjacency matrix W of a graph, the duplicated graph $\mathcal{G}_d$ admits the following adjacency matrix:

$$\mathsf{W}_d = \begin{bmatrix} \mathsf{W} & \mathsf{W}_b \\ \mathsf{W}_b & \mathsf{W} \end{bmatrix},$$

where $\mathsf{W}_b \in \mathbb{R}^{n\times n}$ contains non-negative edge weights. In other words, in order to form the duplicated graph $\mathcal{G}_d$, we duplicate $\mathcal{G}$ and create edges between the nodes of the original and copy graph where the weights of these edges are contained in W_b. Indeed the options for W_b are limitless. Here we restrict ourselves to (entry-wise) non-negative matrices that are matrix functions evaluated at L *i.e.* $\mathsf{W}_b = h(\mathsf{L}) \coloneqq \mathsf{U}h(\Lambda)\mathsf{U}^\top$ for some function $h : \mathbb{R} \to \mathbb{R}$. In turn, we ensure that every block of L_d is diagonalizable by U. This constraint seems a bit technical, but it allows us to keep U as the Fourier basis in the analysis of the RSF filters proposed in the following.

Let us delve into the filtering properties on $\mathcal{G}_d$. In order to do so, we first write the corresponding graph Laplacian L_d. By observing $h(\mathsf{L})\mathbf{1} = h(\mathbf{0})$, we can simplify L_d to the following block matrix form:

$$\mathsf{L}_d = \mathsf{D}_d - \mathsf{W}_d = \begin{bmatrix} h(0)\mathsf{I} + \mathsf{D} & 0 \\ 0 & h(0)\mathsf{I} + \mathsf{D} \end{bmatrix} - \begin{bmatrix} \mathsf{W} & h(\mathsf{L}) \\ h(\mathsf{L}) & \mathsf{W} \end{bmatrix} = \begin{bmatrix} h(0)\mathsf{I} + \mathsf{L} & -h(\mathsf{L}) \\ -h(\mathsf{L}) & h(0)\mathsf{I} + \mathsf{L} \end{bmatrix}.$$

Given this graph, let us take a look at the filtering induced by the Tikhonov regularization, which computes the smooth signal as:

$$\hat{\mathbf{x}} = q(\mathsf{L} + q\mathsf{I})^{-1}\mathbf{y}.$$

First of all, $\mathcal{G}_d$ contains $2n$ nodes thus $2n$ measurements $y_1, \ldots, y_{2n}$. Therefore, we need to clarify the definition of the measurements vector. A natural choice is to duplicate the original measurements. In order to explore more graph filters on $\mathcal{G}_d$, we define a parametrized duplication as $\mathbf{y}_d = \begin{bmatrix}\alpha\mathbf{y} \\ \beta\mathbf{y}\end{bmatrix}$. Similarly, instead of using the same value of q in both the original and copy graph, we give the flexibility of choosing different values q_1 and q_2 so that we can explore the full potential of the new set of filters. Under this parametrization, the corresponding kernel K_d becomes:

$$\begin{aligned}\mathsf{K}_d = (\mathsf{L}_d + \mathsf{Q}_d)^{-1}\mathsf{Q}_d &= \begin{bmatrix}(h(0)+q_1)\mathsf{I} + \mathsf{L} & -h(\mathsf{L}) \\ -h(\mathsf{L}) & (h(0)+q_2)\mathsf{I} + \mathsf{L}\end{bmatrix}^{-1}\begin{bmatrix}q_1\mathsf{I} & 0 \\ 0 & q_2\mathsf{I}\end{bmatrix} \\ &= \begin{bmatrix}q_1\mathsf{M}_2\mathsf{S} & q_2 h(\mathsf{L})\mathsf{S} \\ q_1 h(\mathsf{L})\mathsf{S} & q_2\mathsf{M}_1\mathsf{S}\end{bmatrix} = \begin{bmatrix}\mathsf{K}_1 & \mathsf{K}_2 \\ \mathsf{K}_3 & \mathsf{K}_4\end{bmatrix}\end{aligned} \tag{4.13}$$

where $\mathsf{M}_{1,2} = (h(0) + q_{1,2})\mathsf{I} + \mathsf{L}$ and $\mathsf{S} = (\mathsf{M}_1\mathsf{M}_2 - h^2(\mathsf{L}))^{-1}$. Then, the product with $\mathbf{y}_d$ gives:

$$\mathsf{K}_d\mathbf{y}_d = \begin{bmatrix}(\alpha\mathsf{K}_1 + \beta\mathsf{K}_2)\mathbf{y} \\ (\alpha\mathsf{K}_3 + \beta\mathsf{K}_4)\mathbf{y}\end{bmatrix}.$$

Notice that all $\mathsf{K}_{1,2,3,4}$ commutes with L thanks to our initial constraint. Then, we can derive the transfer function of the filtering implemented by $(\alpha\mathsf{K}_1 + \beta\mathsf{K}_2)\mathbf{y}$ as follows:

$$f_\theta(\lambda) = \frac{\alpha q_1(\lambda + h(0) + q_2) + \beta q_2(h(\lambda))}{(\lambda + h(0) + q_1)(\lambda + h(0) + q_2) - h(\lambda_i)^2}, \tag{4.14}$$

where we collect the hyper-parameters into $\theta = (q_1, q_2, \alpha, \beta)$. This transfer function with four hyper-parameters gives us a big family of filters, and all of these filters can be approximated by the unbiased estimators $\tilde{\mathbf{x}}$, $\bar{\mathbf{x}}$ and $\bar{\mathbf{z}}$. On top of the hyper-parameters, a significant design choice is the selection of the matrix function h. Unlike θ, we are not completely free on the choice of h as it needs to verify that $h(\mathsf{L})$ is a non-negative matrix. Some simple examples are:

- $h(\mathsf{L}) = d\mathsf{I}$ for some $d > 0$,
- $h(\mathsf{L}) = (d_{max}\mathsf{I} - \mathsf{L})^p$ for some positive integer p,
- $h(\mathsf{L}) = \mathsf{U}e^{-\Lambda}\mathsf{U}^\top$.

The functions that map the graph Laplacian to non-negative matrices have been analyzed in several works by [BCN05; BH08; MW79]. In fact, these works consider a more general class of matrices, called M-matrices, instead of the Laplacian. The M-matrices are defined as the real matrices which contain only non-positive values in the off-diagonal entries. As a result, all graph Laplacians are also M-matrices. Then the following theorem gives us a necessary condition for the function h to build $h(\mathsf{L}) = \mathsf{W}_b$.

Theorem 4.3.1 (Necessary condition for h [BCN05])**.** *Let $h : \mathbb{R}^+ \cup \{0\} \to \mathbb{R}$ be analytic function. The function h maps M-matrices to non-negative matrices only if h verifies for all $i = 1, 2, \dots$ and for $x \in (0, \infty)$:*

$$(-1)^i h^{(i)}(x) \geq 0,$$

in other words, h is completely monotonic.

Proof. We refer the reader to the proof of Theorem 2.1 in [BCN05]. □

Thanks to this necessary condition, we can check the feasibility of a candidate function for building $\mathcal{G}_d$.

Overall, we define here a filtering scheme on an altered version of the graph. The way we alter the graph allows us to keep the same graph Fourier basis with the filtering schemes on the original graph. Further analysis give us a wide set of transfer functions that can be obtained from this new filtering scheme. Most importantly, all of these filters can be approximated by the RSF-based estimators. This fact opens many possible use cases for the forest estimators beyond approximating $g_q(\lambda)$. The following case study is dedicated to illustrate these filters. In this illustration, we empirically show that there exist feasible parameter set θ that allows us to approximate the ideal low-pass filter $g_k(\lambda)$ or the (p-q)-rational filter $g_{p,q}(\lambda) = \frac{q}{q+\lambda^p}$ better than $g_q(\lambda)$.

Low-pass Forest Filters

Filtering a signal by low-pass filters without diagonalizing L is usually desired in GSP [RWS20]. In approximating such filters, one can deploy approximate approaches such as Chebyshev polynomials [SVF11] or graph ARMA filters [ILSL16] yielding state-of-the-art performance. Here we look at a dif-

ferent perspective on approximating low-pass filters. Since the signals filtered by g_q and f_θ can be cheaply estimated by RSF-based estimators without any bias, in this example, we aim to explore the potential of approximating two particular low-pass filters $g_k(\lambda)$ and $g_{p,q}(\lambda)$.

First, let us fix some notation. In a more general sense, we consider a target transfer function $g^\star(\lambda)$ and try to approximate $g^\star(\lambda)$ with a candidate function $c_\Theta(\lambda)$ with a parameter set Θ. As a result, we look for filter parameters $\Theta^\star$ such that:

$$\Theta^\star = \underset{\Theta}{\operatorname{argmin}} \sum_{i=1}^{n} (g^\star(\lambda_i) - c_\Theta(\lambda_i))^2 + \mu ||c_\Theta(0) - 1||_2^2,$$

over the graph frequencies λ_i's[a] and $\mu||c_\Theta(0) - 1||_2^2$ is the regularization term that enforces that the estimated filter equals to 1 at $\lambda = 0$ similar to the target filters. Under this generic setup, we consider $g_k(\lambda)$ or $g_{p,q}(\lambda)$ as the target function, and $g_q(\lambda)$ or $f_\theta(\lambda)$ as the candidate function. In both options of $c_\Theta(\lambda)$, here we have a constrained optimization problem as we need to ensure $q > 0$ for $g_q(\lambda)$ or $q_{1,2} > 0$ for $f_\theta(\lambda)$. On the other hand, this type of approximation for the target functions is quite appealing as we only need to solve an optimization problem with a few parameters. In the case of $g_q(\lambda)$ being the candidate function, one can find a global minimizer solving the corresponding single variable polynomial under the constraint $q > 0$. Indeed a similar analysis should be done for $f_\theta(\lambda)$, which comes in a much more complicated form. In this case study, we prefer to avoid this type of analysis and empirically show that $f_\theta(\lambda)$ may provide a better approximation for the target functions even if the parameter set θ is a local minimizer. In the following, we illustrate this idea on a 3D-Grid graph with 1000 nodes and 3000 edges.

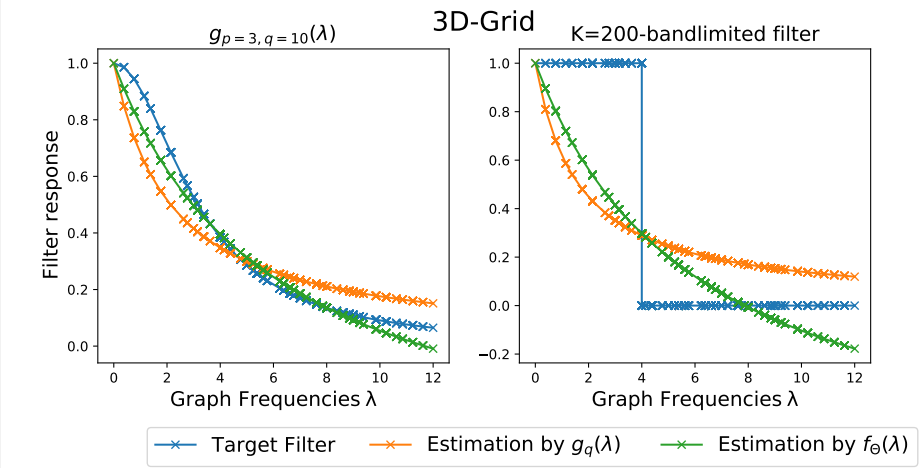


Fig. 4.6.: *We approximate two target filters $g_k(\lambda)$ and $g_{p,q}(\lambda)$ via g_q and f_Θ. For f_Θ, we choose $h(\mathsf{L}) = d_{max}\mathsf{I} - \mathsf{L}$. Then we optimize the parameters of the candidate filters g_q and f_Θ for estimating the filters $g_{k=200}(\lambda)$ and $g_{p=2,q=10}(\lambda)$. In the first figure, we find the parameter $q = 2.14$ for g_q and $\Theta = (0.61, 1.70, 8.02, 18.16)$ for f_Θ. As you can see f_Θ yields much more approximate filter when these parameters plugged compared to that of g_q. The similar results are also observed in the case of approximating $g_{k=200}(\lambda)$. The filter with four parameter f_Θ where $\Theta = (0.25, 2.93, 10.80, 13.88)$ have more flexibility compared to $g_{q=1.63}$, thus it yields a better approximation.*

[a] In practice, this can be replaced by the estimation of the spectrum as it is expensive to calculate all the eigenvalues

4.4 Conclusion

In this section, we present the forest-based algorithms to approximate three important graph quantities, namely the trace of the regularized inverse of the graph Laplacian, the effective resistances and various graph filters. Firstly, the proposed methods for the trace estimation problem in Section 4.1.1 are the improved versions of the forest estimator introduced in [BTGAA19]. Although these improvements take their root from the existing variance reduction methods [Owe13] (control variate method and stratified sampling), it is not evident how to apply them on an arbitrary estimator as they require additional statistics. Our contribution here is to provide such statistics so that these variance reduction techniques can be adapted to improve estimation performance. In turn, we observe that the proposed methods reach the SOTA performance over various graph datasets. In estimating effective resistances, we propose two novel algorithms, which are local and global estimators. Our analysis shows that the global algorithm we propose has interesting properties

but has poor estimation performance. On the other hand, the local algorithm yields simple implementation and comparable performance with the SOTA algorithms as observed in Section 4.1.2. Finally, we show that the graph filters that can be approximated via RSFs are not limited to $\frac{q}{q+\lambda}$. In doing so, we first show that $\mathsf{L}^{\dagger}\mathbf{y}$ is another filtering operation that can be estimated via RSFs, in particular 2-rooted random forests. Then, we slightly modify the graph by extending the filters that can be obtained RSFs. In this modification, we duplicate the graph by keeping the original edges and connecting the nodes in the original and copy graphs. In turn, this gives a family of filters parametrized by four parameters. Finally, we illustrate this filter in approximating certain graph filters such as g_k and $g_{q,p}$.

Conclusion

5

> *The song is ended but the melody lingers on...*
>
> **— Irving Berlin**

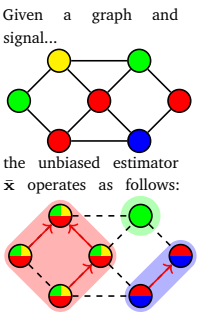
The graph Laplacian is at the heart of many applications involving graph-structured data. Given their extensive use, many numerical algebra tools (deterministic or randomized) have been adapted for dealing with the graph Laplacian. In this thesis, we give a brand new set of techniques for randomized Laplacian-based algebra. These techniques cover a wide variety of problems, including graph Tikhonov regularization, interpolation, node classification, graph ℓ_1-regularization, trace estimation of the regularized inverse of L, estimation of the effective resistances and graph signal filtering. Although these problems seem entirely unrelated, their solutions are closely linked with random spanning forests. Some already known and others uncovered through the thesis, these links have been the supporting pillars of the techniques proposed through this thesis. They take their root in the connections between various concepts, namely graph theory, determinantal point processes and randomized linear algebra. Therefore, we started by giving the necessary background on these concepts and also random spanning forests by emphasizing their connections with RSFs. Then, we presented our randomized methods for the problems in Laplacian-based numerical algebra. For each method, we provided a bias-variance analysis, and we compared them with the existing algorithms over various real datasets whenever possible.

Limitations, Open Questions and Perspectives

In this section, we will discuss each proposed method of this thesis by pointing out its strengths and weaknesses while highlighting associated open questions and future directions.

In Chapter 3, we present a unifying framework in Prob. (3.1.1) for Tikhonov regularization and interpolation on graph signals. While the former draws a regularized unconstrained optimization problem, the latter dictates the value of the signal at

certain nodes. With the help of the node-wise hyperparametrization (defining q_i per node i), we show that both cases can be understood as an instance of Prob. (3.1.1). Besides the wide variety of applications, from graph signal filtering to electrical resistor networks, the explicit solution to the unified problem can be found in a closed-form expression. One of our contributions in this chapter is the forest-based estimator $\tilde{\mathbf{x}}$ for approximating this solution $\hat{\mathbf{x}}$ for all cases. In a nutshell, this estimator samples rooted spanning forests and propagates the signal value at the root within each tree. As simple as it is, the bias-variance analysis shows it has no bias and a tractable variance in terms of the input signal and the Laplacian. Moreover, it is possible to apply variance reduction techniques to reduce its expected error significantly. This brings us to the other contributions of Chapter 3. Thanks to the rich theory of RSFs and existing algorithms, we manage to adapt two variance reduction techniques, namely conditional expectation and the control variate method. We obtain new estimators with no bias and reduced variance in both cases.

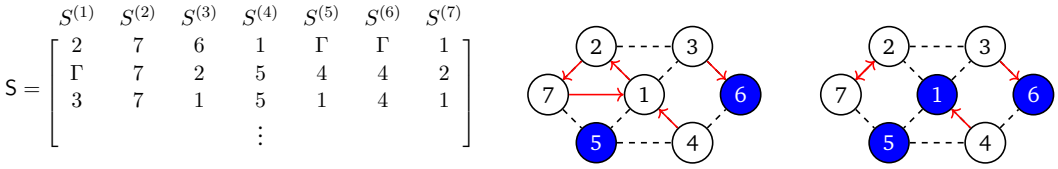


On the other hand, the existing algorithms, namely preconditioned conjugate gradient descent and Chebyshev polynomial approximation, to compute $\hat{\mathbf{x}}$ are deterministic algorithms with a high convergence rate. Specific to inverting Laplacian systems, one can find more sophisticated preconditioning methods for iterative methods based on spanning trees or graph sparsification [Vis+13]. In turn, they speed up the convergence of the algorithm, yielding nearly linear time algorithm with m. In contrast, the proposed methods are subjected to the Monte Carlo convergence rate $\mathcal{O}(N^{-1/2})$ *i.e.* in order to divide the squared error $||\tilde{\mathbf{x}}-\hat{\mathbf{x}}||_2^2$ by 2, one needs to double the sample size. Over various graph datasets in Section 3.3.3, we observe that this rate remains slow compared to the convergence of the deterministic methods. Yet again, in many applications, the exact solution $\hat{\mathbf{x}}$ itself is an inaccurate quantity. For example, in a signal denoising setup, the noise-free signal differs from $\hat{\mathbf{x}}$ by some error. Keeping this in mind, we also measure the denoising performance of our estimators and see that they are comparable with state-of-the-art algorithms.

Indeed, the slower convergence rate is the principal issue of forest-based estimators. On top of this, certain compiler and hardware optimizations give the deterministic methods an advantage, while these are not yet available for the RSF estimators. The deterministic methods access the graph Laplacian L only by the matrix-vector products. These operations are well-optimized in the compilation by considering run-time performance and memory accesses. In the case of forest-based estimators, one needs to run loop-erased-random-walks over the graph, which access the entries of L in arbitrary indices. This brings an additional cost to pull the random parts of L into the computer's cache memory. Given the architecture of today's computers, this cost is inevitable. However, one may still look for speeding up in

Wilson's algorithm itself. In fact, this has been an open research question during the last decades, and many have attempted to propose a faster algorithm than Wilson's for sampling uniform spanning trees. Many attempts in this field yielded approximate algorithms *i.e.* samples a spanning tree from approximately uniform distribution [KM09; MST14; Sch18; ALGVV21]. While some use the links of USTs with the graph Laplacian, some others choose MCMC methods to approximate the distribution of UST with certain theoretical guarantees. Although these methods are theoretically appealing, they require implementing either an algebraic tool or a specific data structure which makes the implementation complicated. On the other hand, Wilson's algorithm is very easy to implement as it is presented in Alg. 2. The most time-consuming step in Wilson's algorithm is usually the first LERW that only terminates at a single node which is the given root (Γ for the forests). As no other path is already sampled, the first LERW might take a lot of time to visit a lot of nodes before its termination. On the other hand, the last LERWs terminate very quickly as they may stop at many nodes. Recently, [NIF22] states that the algorithm by Aldous [Ald90] and Broder [Bro89], which was the fastest algorithm before Wilson's algorithm, has the exact opposite behaviour. Based on this fact, the authors in [NIF22] suggest a hybrid framework that deploys Aldous-Broder's algorithm at the initial steps and launches Wilson's algorithm later on to complete the spanning tree. They show that for certain graphs, this procedure still samples a UST faster than both algorithms. However, at its current stage, it is not generalized to general graphs. Overall, Wilson's algorithm remains the *current* best option for our estimators.

In Chapter 4, we focus on the RSF estimation of other graph quantities, which are the trace of the regularized inverse of Laplacian, effective resistances and various graph filters. Each quantity appears as the solution to a problem involving the graph Laplacian. Firstly, the trace $\mathrm{tr}(q(\mathsf{L}+q\mathsf{I})^{-1})$ is often required in the hyper-parameter selection algorithms in order to tune the hyper-parameter q in GTR. The existing algorithms usually estimate it via Monte Carlo simulations, such as Hutchinson's estimator [Hut89] and the number of roots in RSFs [BTGAA19]. We work on the RSF-based algorithm, the number of roots s, to improve it with variance reduction methods. In doing so, we adapt two variance reduction algorithms, namely control variate and stratified sampling, for reducing the variance of s yielding the best, or at least comparable, results with the state-of-the-art algorithms. In adapting control variate method, we follow a very similar procedure while we transform the estimator $\bar{\mathbf{x}}$ (resp. $\tilde{\mathbf{x}}$) into $\bar{\mathbf{z}}$ (resp. $\tilde{\mathbf{z}}$). In turn, one only needs to track the edges connecting two nodes belong to different trees for computing the control variate. To implement this, we only add a few lines in Wilson's algorithm, and we observe in empirical tests



(a) The stacks

(b) Graph at the first layer

(c) Graph at the second layer

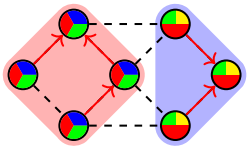
Fig. 5.1.: *An example of the stack representation (left) and the induced graphs from the first and the second layer. Blue nodes are the current root set.*

that this is a cheap price to pay given the variance reduction provided. However, our implementation gives a naive way of tracking edges at the boundaries. An open question is whether there exist faster algorithms or, more probably, data structures that allow us to track these edges without traversing all the edges. A possible direction might be to implement link-cut trees as done in [RTF18] in order to track the edges of interest meanwhile generating the spanning forest. The second variance reduction method we adapt is stratified sampling. We divide the outcome space of $s = |\rho(\Phi_q)|$ by using a random quantity $|\rho_1(\Phi_q)|$ (the number of roots sampled at the first visit of LERWs in Wilson's algorithm) while collecting samples. This quantity has a tractable probability distribution (Poisson-Binomial) and it is easy to sample from. Therefore, we choose to use the stratification variable to decrease the variance of s. In turn, the new estimator competes and even outperforms the existing algorithms. Indeed, one may achieve even better performances by combining these two methods by going beyond this work.

Another interpretation of the random set $\rho_1(\Phi_q)$ comes from the stack representation of Wilson's algorithm (See also Fig. 5.1). We consider infinite stacks at each node. At every layer k of a stack $S_k^{(i)}$ associated with node i, one has a random neighbor j with a probability $\frac{w(i,j)}{q+d_i}$ and with probability of $\frac{q}{q+d_i}$, the node i becomes a root. Then, we form a graph by looking at the top element of each layer. Typically, such a graph contain cycles in it. A cycle-popping algorithm removes the edges of the cycles from the corresponding stacks by updating their top elements. Surprisingly, Wilson's algorithm (for forests) is an implementation of a cycle popping algorithm that runs until there is no cycle remaining. The resulting structure is a spanning forest. When we look at the root set of the spanning forest, we see that it evolves through this process until termination. While some roots are sampled in the first layer of the stacks, some others become a root in the continuation of the cycle popping procedure. In fact, $\rho_1(\Phi_q)$ is the roots sampled at the first layer, and this definition can be generalized to $\rho_k(\Phi_q)$ which are the roots sampled until k-th layer. Interestingly, we see that for $k = 1$, $\rho_1(\Phi_q)$ is a DPP with a diagonal marginal kernel $\mathsf{K}_1 = q(q\mathsf{I} + \mathsf{D})^{-1}$. Similarly as k goes to infinity, one recovers the original root set

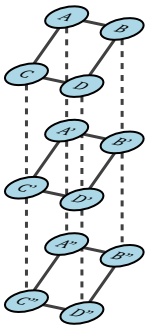
$\rho(\Phi_q)$ with the marginal kernel $\mathsf{K}_{k\to\infty} = \mathsf{K} = q(q\mathsf{I}+\mathsf{L})^{-1}$. Then, the natural question is that *what happens in between*? We wonder whether $\rho_k(\Phi_q)$ is still a DPP for some $k > 1$, or the only cases where the DPPs appear is $k = 1$ and $k \to \infty$. If the former is true, what are their marginal kernels? These questions remain unanswered in this thesis, but one might start to investigate by attempting to write the distributions of these objects by using the distributions of the underlying LERWs.

We also propose two RSF-based estimators for approximating effective resistances without explicitly computing $\mathsf{L}^\dagger$. Both rely on the close connections between two rooted spanning forests and effective resistances. In one, we compute the effective resistances in a global sense (over all pairs of nodes), while the other runs in a much smaller portion of the graph to estimate a single effective resistance. The bias-variance analysis shows that both estimators are biased. Nevertheless, the bias decays much faster than the variance as the sample size increases, leaving the variance as the main source of the approximation error. In initial tests, we observe the global estimator we propose has poor performance. However, it can provide an estimate over all pairs of effective resistances for a single sample of two rooted spanning forests. In its variance analysis, we see that the variance is inversely proportional with c^2 where $c = (\sum_{i,j\in\mathcal{V}} \mathsf{R}_{i,j})^{-1}$. For large graphs, c^2 is typically a small number yielding a large variance in our estimator. Yet again, one might find improvements via recycling unused forests that are not with two roots or control variate suggested in Section 4.2.3. Although it gives poor estimation, it gives cheap matrix-vector products. In turn, one can estimate $\mathsf{L}^\dagger\mathbf{y}$ for an arbitrary vector $\mathbf{y}$ at very cheap cost as described in Section 4.3.1.



The local estimator on the other hand gives a good estimation performance and is comparable with state-of-the-art algorithms. It usually outperforms other existing local algorithms in the maximum number of effective resistances that can be estimated before reaching the best global algorithm. In this algorithm, we set node i and j as roots of estimating $\mathsf{R}_{i,j}$, and we launch LERWs at the neighbours of i. We stop the algorithm when all neighbours of i are rooted. In this way, the algorithm produces an estimate without generating a complete spanning forest of the graph. As aforementioned, the first LERW again take the most of the run-time. An interesting modification might be an early stop strategy in which we stop the first LERW after reaching a certain number of steps, which introduces an additional bias in our estimation. Indeed, an open question is if this version of the algorithm yields a substantial run-time gain without significantly increasing the expected error of the estimator. One might start to analyze the bias by deriving the distribution of the interrupted LERWs, which might not necessarily appear in determinantal ratios as in the distributions of the original laws.

Later, we give a set of graph filters that RSFs can approximate. In this set, we have three types of graph filters whose transfer functions are $g_q(\lambda)$ (the graph filter in GTR), $g^\dagger(\lambda)$ (the filter by the pseudo-inverse of Laplacian) and $f_\Theta(\lambda)$ (the filter obtained by duplicating the graph). As mentioned before, $g^\dagger(\lambda)$ or solving Laplacian linear systems is at the reach of the RSF-based estimators thanks to our global forest-based estimator for effective resistances. The implementation is simple and efficient, but the high variance of the global estimator is also the biggest issue for estimating $g^\dagger(\lambda)$. Therefore, we suggest using this estimator after applying the improvements in Section 4.2.3. This leaves us with the parametric filters $g_q(\lambda)$ and $f_\theta(\lambda)$. By duplicating the graph and solving a version of graph Tikhonov regularization on the duplicated graph, we obtain a new set filter $f_\theta(\lambda)$. Indeed, this filter is also within reach of the RSF-based estimators. Moreover, it is parametrized by four parameters instead of a single one q, which provides a much wider range of filters. We illustrate this extended range in approximating popular graph filters such as the ideal low-pass filters or $g_{q,p>1}(\lambda) = \frac{q}{q+\lambda^p}$ and observe that the duplicated graph filter $f_\theta(\lambda)$ may provide better approximations than $g_q(\lambda)$ with the help of more parameters. Given this increased capability, one might ask; *What happens if we copy the graph more than twice?* In this case, one would obtain more degree of freedom in the filters while the parameters to select increase by 2 per added layer. One can increase the versatility of the filters by choosing the wiring function $h(\lambda)$ differently between different layers. All of these lead us to a complicated function analysis to discover the full potential of the filters in duplicated graphs, which we leave as future work.



As a final remark, we would like to stress that all the methods proposed on Laplacian-based numerical algebra can be adapted for symmetric diagonally dominant (SDD) matrices, which is a larger class of matrices than the graph Laplacian. The trick by [Gre96] allows us to transform the linear systems with SDD matrices into larger linear systems with graph Laplacian. Using this connection, one can replace L with an SDD matrix in GTR, and yet the solution can still be approximated via the RSF estimators. A similar adaptation is already given in [BTGAA19] for the trace estimation problem. Given this extension, one might hope to extend larger classes of matrices than SDD matrices. In this vein, a promising work [Ökt05] gives a random walk-based solution to solve linear systems in the form of $\mathbf{Ax} = \mathbf{b}$ where A is only constrained to be an entry-wise real matrix. Here, an interesting perspective would be replacing LERWs, and so RSFs, in the place of simple random walks for solving $\mathbf{Ax} = \mathbf{b}$. So far, this adaptation is not obvious to our eyes, but we find investigating it promising direction to extend the scope of our methods further.

Epilogue

At first sight, this thesis's main achievements can be considered the RSF-based algorithms for several graph-based problems. Especially, the forest-based algorithms for trace estimation and effective resistances show ensuring performance when compared to existing algorithms. Similarly, the estimators for GTR and interpolation are shown to be useful when one does not need to compute the solution at very high accuracy. These methods are very easy to implement and flexible for parallelization or distributed implementations. In the bigger picture, in this thesis, we reconsider RSFs as tools to *sample a small portion of the graph* and solve the problem at hand on this small portion to avoid heavy computations. Compared to the existing set of tools for SDD matrices [Vis+13; ST04] and DPP-based approaches [DM21], the RSF-based tools bring a brand new and elegant perspective to the literature.

Up to our work, a similar perspective is usually presented with simple random walks by emphasizing the connections between the Markov chains and the graph-related algebra [BOP19]. In this context, for example, the works by [Wu16; WLSWC12] provide a random-walk insight to several practical machine learning algorithms solving GTR, interpolation or clustering. Here, taking a step forward from random walks to loop-erased random walks carries us to the prosperous realm of DPPs and RSFs. In turn, we give practical algorithms to solve very central problems. In addition, we observe that the direction we take has inspired recent research on randomized graph-related algebra. For example, in their recent work [FB22], the authors look at the problems of graph sparsification and solving Laplacian-based linear systems with multi-type spanning forests, which are certain types of DPPs and closely related to RSFs. In the future, we believe that RSFs and other types of DPPs will have more impact on applications of SDD-based linear algebra and hopefully extend to the other classes of matrices.

Bibliography

[AALG17] Vedat Levi Alev, Nima Anari, Lap Chi Lau, and Shayan Oveis Gharan. “Graph clustering using effective resistance”. In: *arXiv preprint arXiv:1711.06530* (2017).

[AB02] Réka Albert and Albert-László Barabási. “Statistical mechanics of complex networks”. In: *Reviews of modern physics* 74.1 (2002), p. 47.

[ACGM18] Luca Avena, Fabienne Castell, Alexandre Gaudillière, and Clothilde Mélot. “Random forests and networks analysis”. In: *Journal of Statistical Physics* 173.3 (2018), pp. 985–1027.

[ACGM20] Luca Avena, Fabienne Castell, Alexandre Gaudillière, and Clothilde Mélot. “Intertwining wavelets or multiresolution analysis on graphs through random forests”. In: *Applied and Computational Harmonic Analysis* 48.3 (2020), pp. 949–992.

[Ald90] David J Aldous. “The random walk construction of uniform spanning trees and uniform labelled trees”. In: *SIAM Journal on Discrete Mathematics* 3.4 (1990), pp. 450–465.

[ALGVV21] Nima Anari, Kuikui Liu, Shayan Oveis Gharan, Cynthia Vinzant, and Thuy-Duong Vuong. “Log-concave polynomials IV: approximate exchange, tight mixing times, and near-optimal sampling of forests”. In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 2021, pp. 408–420.

[AMGS12] Konstantin Avrachenkov, Alexey Mishenin, Paulo Gonçalves, and Marina Sokol. “Generalized optimization framework for graph-based semi-supervised learning”. In: *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM. 2012, pp. 966–974.

[Bap10] Ravindra B Bapat. *Graphs and matrices*. Vol. 27. Springer, 2010.

[BAT19] Simon Barthelmé, Pierre-Olivier Amblard, and Nicolas Tremblay. “Asymptotic equivalence of fixed-size and varying-size determinantal point processes”. In: *Bernoulli* 25.4B (2019), pp. 3555–3589.

[BBV04] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[BCG15] Jose H Blanchet, Nan Chen, and Peter W Glynn. “Unbiased Monte Carlo computation of smooth functions of expectations via taylor expansions”. In: *2015 Winter Simulation Conference (WSC)*. IEEE. 2015, pp. 360–367.

[BCN05] RB Bapat, M Catral, and Michael Neumann. “On functions that preserve M-matrices and inverse M-matrices”. In: *Linear and Multilinear Algebra* 53.3 (2005), pp. 193–201.

[BH08] Gautam Bharali and Olga Holtz. “Functions preserving nonnegativity of matrices”. In: *SIAM Journal on Matrix Analysis and Applications* 30.1 (2008), pp. 84–101.

[BL06] Kurt Bryan and Tanya Leise. “The $25,000,000,000 eigenvector: The linear algebra behind Google”. In: *SIAM review* 48.3 (2006), pp. 569–581.

[Bla47] David Blackwell. “Conditional expectation and unbiased sequential estimation”. In: *The Annals of Mathematical Statistics* (1947), pp. 105–110.

[BLMV17] Rémi Bardenet, Frédéric Lavancier, Xavier Mary, and Aurélien Vasseur. “On a few statistical applications of determinantal point processes”. In: *ESAIM: Proceedings and Surveys* 60 (2017), pp. 180–202.

[BLPS01] Itai Benjamini, Russell Lyons, Yuval Peres, and Oded Schramm. “Special invited paper: uniform spanning forests”. In: *Annals of probability* (2001), pp. 1–65.

[BOP19] Anna Ben-Hamou, Roberto I Oliveira, and Yuval Peres. “Estimating graph parameters with random walks”. In: *Mathematical Statistics and Learning* 1.3 (2019), pp. 375–399.

[Bot12] Léon Bottou. “Stochastic gradient descent tricks”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[BP93] Robert Burton and Robin Pemantle. “Local characteristics, entropy and limit theorems for spanning trees and domino tilings via transfer-impedances”. In: *The Annals of Probability* (1993), pp. 1329–1371.

[BPCPE+11] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. “Distributed optimization and statistical learning via the alternating direction method of multipliers”. In: *Foundations and Trends® in Machine learning* 3.1 (2011), pp. 1–122.

[Bro89] Andrei Z Broder. “Generating random spanning trees”. In: *FOCS*. Vol. 89. 1989, pp. 442–447.

[BTGAA19] Simon Barthelme, Nicolas Tremblay, Alexandre Gaudilliere, Luca Avena, and Pierre-Olivier Amblard. “Estimating the inverse trace using random forests on graphs”. In: *GRETSI 2019 - XXVIIème Colloque francophone de traitement du signal et des images*. Lille, France, Aug. 2019. URL:

[CA02] Pavel Chebotarev and Rafig Agaev. “Forest matrices around the Laplacian matrix”. In: *Linear algebra and its applications* 356.1-3 (2002), pp. 253–274.

[CDHR08] Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam. “Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate”. In: *ACM Transactions on Mathematical Software (TOMS)* 35.3 (2008), pp. 1–14.

[CGT91] Andrew R Conn, Nicholas IM Gould, and Ph L Toint. “Convergence of quasi-Newton matrices generated by the symmetric rank one update”. In: *Mathematical programming* 50.1 (1991), pp. 177–195.

[Chu67] Kai Lai Chung. “Markov chains”. In: *Springer-Verlag, New York* (1967).

[CK78] Seth Chaiken and Daniel J Kleitman. “Matrix tree theorems”. In: *Journal of combinatorial theory, Series A* 24.3 (1978), pp. 377–381.

[CPFSC21] Stefania Colonnese, Manuela Petti, Lorenzo Farina, Gaetano Scarano, and Francesca Cuomo. “Protein-protein interaction prediction via graph signal processing”. In: *IEEE Access* 9 (2021), pp. 142681–142692.

[Dav91] William C Davidon. “Variable metric method for minimization”. In: *SIAM Journal on Optimization* 1.1 (1991), pp. 1–17.

[DCV19] Michal Derezinski, Daniele Calandriello, and Michal Valko. “Exact sampling of determinantal point processes with sublinear time preprocessing”. In: *Advances in neural information processing systems* 32 (2019).

[DKM06] Petros Drineas, Ravi Kannan, and Michael W Mahoney. “Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix”. In: *SIAM Journal on computing* 36.1 (2006), pp. 158–183.

[DM16] Petros Drineas and Michael W Mahoney. “RandNLA: randomized numerical linear algebra”. In: *Communications of the ACM* 59.6 (2016), pp. 80–90.

[DM21] Michał Derezinski and Michael W Mahoney. “Determinantal point processes in randomized numerical linear algebra”. In: *Notices of the American Mathematical Society* 68.1 (2021), pp. 34–45.

[DMMS11] Petros Drineas, Michael W Mahoney, Shan Muthukrishnan, and Tamás Sarlós. “Faster least squares approximation”. In: *Numerische mathematik* 117.2 (2011), pp. 219–249.

[DWM04] Jennifer A Dunne, Richard J Williams, and Neo D Martinez. “Network structure and robustness of marine food webs”. In: *Marine Ecology Progress Series* 273 (2004), pp. 291–302.

[EK13] Wendy Ellens and Robert E Kooij. “Graph measures and network robustness”. In: *arXiv preprint arXiv:1311.5064* (2013).

[ER05] Alan Edelman and N Raj Rao. “Random matrix theory”. In: *Acta numerica* 14 (2005), pp. 233–297.

[FB22] Michaël Fanuel and Rémi Bardenet. “Sparsification of the regularized magnetic Laplacian with multi-type spanning forests”. In: *arXiv preprint arXiv:2208.14797* (2022).

[FFFFM12] Georg Frobenius, Ferdinand Georg Frobenius, Ferdinand Georg Frobenius, Ferdinand Georg Frobenius, and Germany Mathematician. “Über Matrizen aus nicht negativen Elementen”. In: (1912).

[FKL19] Farzad V Farahani, Waldemar Karwowski, and Nichole R Lighthall. “Application of graph theory for identifying connectivity patterns in human brain networks: a systematic review”. In: *frontiers in Neuroscience* 13 (2019), p. 585.

[FL50] George E Forsythe and Richard A Leibler. “Matrix inversion by a Monte Carlo method”. In: *Mathematics of Computation* 4.31 (1950), pp. 127–129.

[Fle70] Roger Fletcher. “A new approach to variable metric algorithms”. In: *The computer journal* 13.3 (1970), pp. 317–322.

[Fos49] Ronald M Foster. “The average impedance of an electrical network”. In: *Contributions to Applied Mechanics (Reissner Anniversary Volume)* 333 (1949).

[GDJSRLHVRT+21] Thomas Gaudelet, Ben Day, Arian R Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy BR Hayter, Richard Vickers, Charles Roberts, Jian Tang, et al. “Utilizing graph machine learning within drug discovery and development”. In: *Briefings in bioinformatics* 22.6 (2021), bbab159.

[Gre96] Keith D Gremban. “Combinatorial preconditioners for sparse, symmetric, diagonally dominant linear systems”. PhD thesis. Carnegie Mellon University, 1996.

[Gri04] Geoffrey Grimmett. “The random-cluster model”. In: *Probability on discrete structures*. Springer, 2004, pp. 73–123.

[GX04] Ivan Gutman and W Xiao. “Generalized inverse of the Laplacian matrix and some applications”. In: *Bulletin (Académie serbe des sciences et des arts. Classe des sciences mathématiques et naturelles. Sciences mathématiques)* (2004), pp. 15–23.

[HAY16] Takanori Hayashi, Takuya Akiba, and Yuichi Yoshida. “Efficient Algorithms for Spanning Tree Centrality.” In: *IJCAI*. Vol. 16. 2016, pp. 3733–3739.

[HBMBRV17] Weiyu Huang, Thomas AW Bolton, John D Medaglia, Danielle S Bassett, Alejandro Ribeiro, and Dimitri Van De Ville. “A graph signal processing view on functional brain imaging”. In: *arXiv preprint arXiv:1710.01135* (2017).

[HKLW12] Sebastian Haeseler, Matthias Keller, H Daniel Lenz, and Radosław K Wojciechowski. “Laplacians on infinite graphs: Dirichlet and Neumann boundary conditions”. In: *Journal of Spectral theory* 2.4 (2012), pp. 397–432.

[HKPV06] J Ben Hough, Manjunath Krishnapur, Yuval Peres, and Bálint Virág. “Determinantal processes and independence”. In: *Probability surveys* 3 (2006), pp. 206–229.

[HLDN19] Arman Hasanzadeh, Xi Liu, Nick Duffield, and Krishna R Narayanan. “Piecewise stationary modeling of random processes over graphs with an application to traffic prediction”. In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE. 2019, pp. 3779–3788.

[HLDNC17] Arman Hasanzadeh, Xi Liu, Nick Duffield, Krishna R Narayanan, and Byron Chigoy. “A graph signal processing approach for real-time traffic prediction in transportation networks”. In: *arXiv preprint arXiv:1711.06954* (2017).

[Hon13] Yili Hong. “On computing the distribution function for the Poisson binomial distribution”. In: *Computational Statistics & Data Analysis* 59 (2013), pp. 41–51.

[HTFF09] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[Hur] Hurriyet. *Shooting at sun won’t cool down earth: NASA scientist to Turkish girl*. URL:

[Hut89] Michael F Hutchinson. “A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines”. In: *Communications in Statistics-Simulation and Computation* 18.3 (1989), pp. 1059–1076.

[HVG11] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. “Wavelets on graphs via spectral graph theory”. In: *Applied and Computational Harmonic Analysis* 30.2 (2011), pp. 129–150.

[ILSL16] Elvin Isufi, Andreas Loukas, Andrea Simonetto, and Geert Leus. “Autoregressive moving average graph filtering”. In: *IEEE Transactions on Signal Processing* 65.2 (2016), pp. 274–288.

[Joh05] Kurt Johansson. “Random matrices and determinantal processes”. In: *arXiv preprint math-ph/0510038* (2005).

[Joh84] William B Johnson. “Extensions of Lipschitz mappings into a Hilbert space”. In: *Contemp. Math.* 26 (1984), pp. 189–206.

[JWHCLWSCWH21] Dejun Jiang, Zhenxing Wu, Chang-Yu Hsieh, Guangyong Chen, Ben Liao, Zhe Wang, Chao Shen, Dongsheng Cao, Jian Wu, and Tingjun Hou. "Could graph neural networks learn better molecular representation for drug discovery? A comparison study of descriptor-based and graph-based models". In: *Journal of cheminformatics* 13.1 (2021), pp. 1–23.

[KCbH04] Tomer Kalisky, Reuven Cohen, Daniel ben-Avraham, and Shlomo Havlin. "Tomography and stability of complex networks". In: *Complex networks*. Springer, 2004, pp. 3–34.

[Kei22] Ryan Keisler. "Forecasting Global Weather with Graph Neural Networks". In: *arXiv preprint arXiv:2202.07575* (2022).

[Kir47] Gustav Kirchhoff. "Ueber die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird". In: *Annalen der Physik* 148.12 (1847), pp. 497–508.

[KM09] Jonathan A Kelner and Aleksander Madry. "Faster generation of random spanning trees". In: *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. IEEE. 2009, pp. 13–21.

[Koz07] Gady Kozma. "The scaling limit of loop-erased random walk in three dimensions". In: *Acta mathematica* 199.1 (2007), pp. 29–152.

[KT+12] Alex Kulesza, Ben Taskar, et al. "Determinantal point processes for machine learning". In: *Foundations and Trends® in Machine Learning* 5.2–3 (2012), pp. 123–286.

[Law79] Gregory Francis Lawler. "A SELF-AVOIDING RANDOM WALK." PhD thesis. Princeton University, 1979.

[LBU10] Florian Luisier, Thierry Blu, and Michael Unser. "Image denoising in mixed Poisson–Gaussian noise". In: *IEEE Transactions on image processing* 20.3 (2010), pp. 696–708.

[LP16] Russell Lyons and Yuval Peres. *Probability on Trees and Networks*. Vol. 42. Cambridge Series in Statistical and Probabilistic Mathematics. Available at . Cambridge University Press, New York, 2016, pp. xv+699. ISBN: 978-1-107-16015-6. DOI: URL:

[Mac75] Odile Macchi. “The coincidence approach to stochastic point processes”. In: *Advances in Applied Probability* 7.1 (1975), pp. 83–122.

[Mar00] Philippe Marchal. “Loop-erased random walks, spanning trees and Hamiltonian cycles”. In: *Electronic Communications in Probability* 5 (2000), pp. 39–50.

[Mar15] Pierre Martinetti. “Designing the sound of a cut-off drum”. In: *arXiv preprint arXiv:1502.02720* (2015).

[MFEMB19] Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein. “Fake news detection on social media using geometric deep learning”. In: *arXiv preprint arXiv:1902.06673* (2019).

[MKGS14] Sisi Ma, Patrick Kemmeren, David Gresham, and Alexander Statnikov. “De-novo learning of genome-scale regulatory networks in S. cerevisiae”. In: *Plos one* 9.9 (2014), e106479.

[MKJ19] Sarat Babu Moka, Dirk P Kroese, and Sandeep Juneja. “Unbiased estimation of the reciprocal mean for non-negative random variables”. In: *2019 Winter Simulation Conference (WSC)*. IEEE. 2019, pp. 404–415.

[MLWFM16] Shaoshuai Mou, Zhiyun Lin, Lili Wang, Daniel Fullmer, and A Stephen Morse. “A distributed algorithm for efficiently solving linear equations and its applications (special issue jcw)”. In: *Systems & Control Letters* 91 (2016), pp. 21–27.

[MP20] José Vinıécius de Miranda Cardoso and Daniel P Palomar. “Learning undirected graphs in financial markets”. In: *2020 54th Asilomar Conference on Signals, Systems, and Computers*. IEEE. 2020, pp. 741–745.

[MST14] Aleksander Madry, Damian Straszak, and Jakub Tarnawski. “Fast generation of random spanning trees and the effective resistance metric”. In: *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2014, pp. 2019–2036.

[MT20] Per-Gunnar Martinsson and Joel A Tropp. “Randomized numerical linear algebra: Foundations and algorithms”. In: *Acta Numerica* 29 (2020), pp. 403–572.

[MW79] Charles A Micchelli and Ralph A Willoughby. “On functions which preserve the class of Stieltjes matrices”. In: *Linear Algebra and its Applications* 23 (1979), pp. 141–156.

[NIF22] Igor Nunes, Giulio Iacobelli, and Daniel Ratton Figueiredo. “A transient equivalence between Aldous-Broder and Wilson’s algorithms and a two-stage framework for generating uniform spanning trees”. In: *arXiv preprint arXiv:2206.12378* (2022).

[NMR15] Vivi Nastase, Rada Mihalcea, and Dragomir R Radev. “A survey of graphs in natural language processing”. In: *Natural Language Engineering* 21.5 (2015), pp. 665–698.

[NO12] Sunil K Narang and Antonio Ortega. “Perfect reconstruction two-channel wavelet filter banks for graph structured data”. In: *IEEE Transactions on Signal Processing* 60.6 (2012), pp. 2786–2799.

[OBS01] Alan V Oppenheim, John R Buck, and Ronald W Schafer. *Discrete-time signal processing. Vol. 2*. Upper Saddle River, NJ: Prentice Hall, 2001.

[OFKMV18] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José MF Moura, and Pierre Vandergheynst. “Graph signal processing: Overview, challenges, and applications”. In: *Proceedings of the IEEE* 106.5 (2018), pp. 808–828.

[Ökt05] Giray Ökten. “Solving linear equations by Monte Carlo simulation”. In: *SIAM Journal on Scientific Computing* 27.2 (2005), pp. 511–531.

[OLe80] Dianne P O’Leary. “The block conjugate gradient algorithm and related methods”. In: *Linear algebra and its applications* 29 (1980), pp. 293–322.

[OR02] Evelien Otte and Ronald Rousseau. “Social network analysis: a powerful strategy, also for the information sciences”. In: *Journal of information Science* 28.6 (2002), pp. 441–453.

[OTLFPG22] Yassine El Ouahidi, Hugo Tessier, Giulia Lioi, Nicolas Farrugia, Bastien Pasdeloup, and Vincent Gripon. “Pruning Graph Convolutional Networks to select meaningful graph frequencies for fMRI decoding”. In: *arXiv preprint arXiv:2203.04455* (2022).

[Owe13] Art B Owen. “Monte Carlo theory, methods and examples”. In: (2013).

[PA13] Giuliano Andrea Pagani and Marco Aiello. “The power grid as a complex network: a survey”. In: *Physica A: Statistical Mechanics and its Applications* 392.11 (2013), pp. 2688–2700.

[PABT20] Yusuf Y Pilavci, Pierre-Olivier Amblard, Simon Barthelme, and Nicolas Tremblay. “Smoothing graph signals via random spanning forests”. In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 5630–5634.

[PABT21a] Yusuf Pilavcı, Pierre-Olivier Amblard, Simon Barthelmé, and Nicolas Tremblay. “Variance reduction in stochastic methods for large-scale regularised least-squares problems”. In: *arXiv preprint arXiv:2110.07894* (2021).

[PABT21b] Yusuf Yiğit Pilavcı, Pierre-Olivier Amblard, Simon Barthelme, and Nicolas Tremblay. “Graph tikhonov regularization and interpolation via random spanning forests”. In: *IEEE transactions on Signal and Information Processing over Networks* 7 (2021), pp. 359–374.

[PABT22] Yusuf Yigit Pilavci, Pierre-Olivier Amblard, Simon Barthelme, and Nicolas Tremblay. “Variance Reduction for Inverse Trace Estimation via Random Spanning Forests”. In: *GRETSI 2022 - XXVIIIème Colloque Francophone de Traitement du Signal et des Images*. Nancy, France, Sept. 2022.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. *The PageRank citation ranking: Bringing order to the web.* Tech. rep. Stanford InfoLab, 1999.

[Per07] Oskar Perron. “Zur theorie der matrices”. In: *Mathematische Annalen* 64.2 (1907), pp. 248–263.

[PIL19] YUSUF YIGIT PILAVCI. “Random spanning forests: theory and applications”. In: (2019).

[PLYG21] Pan Peng, Daniel Lopatta, Yuichi Yoshida, and Gramoz Goranci. “Local Algorithms for Estimating Effective Resistance”. In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2021, pp. 1329–1338.

[PNV21] George Panagopoulos, Giannis Nikolentzos, and Michalis Vazirgiannis. “Transfer graph neural networks for pandemic forecasting”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 6. 2021, pp. 4838–4845.

[PTGV18] Gilles Puy, Nicolas Tremblay, Rémi Gribonval, and Pierre Vandergheynst. “Random sampling of bandlimited signals on graphs”. In: *Applied and Computational Harmonic Analysis* 44.2 (2018), pp. 446–475.

[PV17] Nathanaël Perraudin and Pierre Vandergheynst. “Stationary signal processing on graphs”. In: *IEEE Transactions on Signal Processing* 65.13 (2017), pp. 3462–3477.

[Rao45] CR Rao. *Information and accuracy attainable in the estimation of statistical parameters. Kotz S & Johnson NL (eds.), Breakthroughs in Statistics Volume I: Foundations and Basic Theory, 235–248*. 1945.

[RBTGV19] Benjamin Ricaud, Pierre Borgnat, Nicolas Tremblay, Paulo Gonçalves, and Pierre Vandergheynst. “Fourier could be a data scientist: From graph Fourier transform to signal processing on graphs”. In: *Comptes Rendus Physique* 20.5 (2019), pp. 474–488.

[RR15] Ahmad Rawashdeh and Anca L Ralescu. “Similarity Measure for Social Networks-A Brief Survey.” In: *Maics* (2015), pp. 153–159.

[RTF18] Luıés MS Russo, Andreia Sofia Teixeira, and Alexandre P Francisco. “Linking and cutting spanning trees”. In: *Algorithms* 11.4 (2018), p. 53.

[RWS20] Raksha Ramakrishna, Hoi-To Wai, and Anna Scaglione. “A user guide to low-pass graph signal processing and its applications: Tools and applications”. In: *IEEE Signal Processing Magazine* 37.6 (2020), pp. 74–85.

[Saa03] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[Sch18] Aaron 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.

[SNFOV13] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains”. In: *IEEE signal processing magazine* 30.3 (2013), pp. 83–98.

[Sol11] Gleb B Sologub. “On measuring of similarity between tree nodes”. In: *Young scientists conference in information retrieval*. 2011, pp. 63–71.

[Spi12] Daniel Spielman. “Spectral graph theory”. In: *Combinatorial scientific computing* 18 (2012).

[SS11] Daniel A Spielman and Nikhil Srivastava. “Graph sparsification by effective resistances”. In: *SIAM Journal on Computing* 40.6 (2011), pp. 1913–1926.

[ST04] Daniel A Spielman and Shang-Hua Teng. “Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems”. In: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. 2004, pp. 81–90.

[SVF11] David I Shuman, Pierre Vandergheynst, and Pascal 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.

[SXGRS20] Rakshith S Srinivasa, Cao Xiao, Lucas Glass, Justin Romberg, and Jimeng Sun. “Fast graph attention networks using effective resistance based graph sparsification”. In: *arXiv preprint arXiv:2006.08796* (2020).

[TAB17] Nicolas Tremblay, Pierre-Olivier Amblard, and Simon Barthelmé. “Graph sampling with determinantal processes”. In: *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE. 2017, pp. 1674–1678.

[TBD16] Mikhail Tsitsvero, Sergio Barbarossa, and Paolo Di Lorenzo. “Signals on graphs: Uncertainty principle and sampling”. In: *IEEE Transactions on Signal Processing* 64.18 (2016), pp. 4845–4860.

[TGB18] Nicolas Tremblay, Paulo Gonçalves, and Pierre Borgnat. “Design of graph filters and filterbanks”. In: *Cooperative and Graph Signal Processing*. Elsevier, 2018, pp. 299–324.

[TPGV16] Nicolas Tremblay, Gilles Puy, Rémi Gribonval, and Pierre Vandergheynst. “Compressive spectral clustering”. In: *International conference on machine learning*. PMLR. 2016, pp. 1002–1011.

[Van10] Piet Van Mieghem. *Graph spectra for complex networks*. Cambridge University Press, 2010.

[Vis+13] Nisheeth K Vishnoi et al. “Lx= b”. In: *Foundations and Trends® in Theoretical Computer Science* 8.1–2 (2013), pp. 1–141.

[VMS21] Javier Villalba-Diez, Martin Molina, and Daniel Schmidt. “Geometric Deep Lean Learning: Evaluation Using a Twitter Social Network”. In: *Applied Sciences* 11.15 (2021), p. 6777.

[Von07] Ulrike Von Luxburg. “A tutorial on spectral clustering”. In: *Statistics and computing* 17.4 (2007), pp. 395–416.

[Wan93] Yuan H Wang. “On the number of successes in independent trials”. In: *Statistica Sinica* (1993), pp. 295–312.

[Wil96] David Bruce Wilson. “Generating random spanning trees more quickly than the cover time”. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 296–303.

[WLSWC12] Xiao-Ming Wu, Zhenguo Li, Anthony So, John Wright, and Shih-Fu Chang. “Learning with partially absorbing random walks”. In: *Advances in neural information processing systems* 25 (2012).

[WMOKBB] Carlos Canudas de Wit, Fabio Morbidi, Luis León Ojeda, Alain Y Kibangou, Iker Bellicot, and Pascal Bellemain. “Grenoble Traffic Lab”. In: ().

[WPCLZP20] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. “A comprehensive survey on graph neural networks”. In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[WPKV14] Xiangrong Wang, Evangelos Pournaras, Robert E Kooij, and Piet Van Mieghem. “Improving robustness of complex networks via the effective graph resistance”. In: *The European Physical Journal B* 87.9 (2014), pp. 1–12.

[Wu16] Xiaoming Wu. *Learning on graphs with partially absorbing random walks: Theory and practice*. Columbia University, 2016.

[Xu21] Mengjia Xu. “Understanding graph embedding methods and their applications”. In: *SIAM Review* 63.4 (2021), pp. 825–853.

[YB20] Ye Yuan and Ziv Bar-Joseph. “GCNG: graph convolutional networks for inferring gene interaction from spatial transcriptomics data”. In: *Genome biology* 21.1 (2020), pp. 1–16.

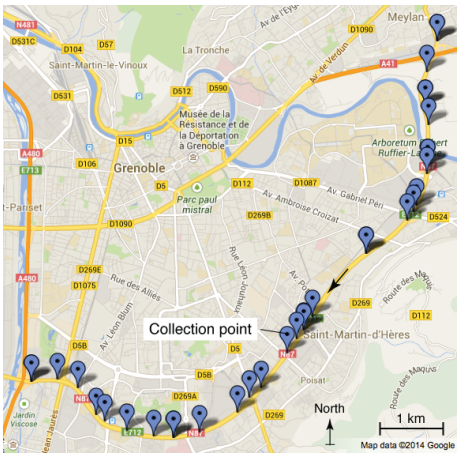
[Zhu05] Xiaojin Zhu. *Semi-supervised learning with graphs*. Carnegie Mellon University, 2005.

Resumé Substantiel

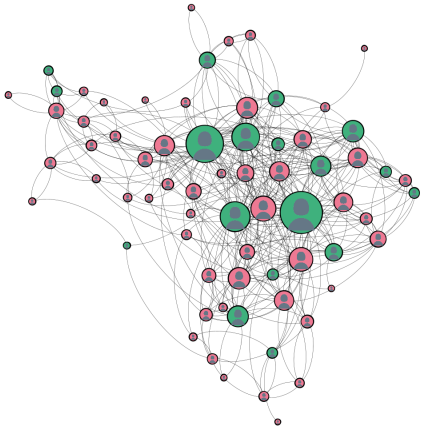
Depuis leur première apparition pour modeliser les ponts de Königsberg, les graphes, c'est-à-dire un ensemble de sommets et d'arêtes, ont été les objets de modélisation et de résolution de nombreux problèmes en physique, chimie, biologie, informatique, sciences sociales et bien d'autres encore. Ils sont si *ubiquitaires* car ils sont les modèles naturels à utiliser lorsque les données représentent des relations mutuelles entre des éléments individuels. Aujourd'hui, le traitement des données structurées en graphes est l'une des principales questions de recherche en traitement du signal et en apprentissage automatique. De nombreuses recherches se concentrent sur l'exploitation de la structure des graphes pour accomplir des tâches avancées, comme la gestion et l'analyse d'Internet (un réseau informatique colossal), la compréhension du cerveau humain [FKL19], la découverte de médicaments [JWHCLWSCWH21], l'analyse des réseaux sociaux [OR02] ou la prévision/planification du trafic à grande échelle [HLDNC17]. D'autres exemples apparaissent dans le traitement du langage naturel [NMR15], la finance [MP20], les réseaux alimentaires [DWM04], les réseaux électriques [PA13] et bien d'autres encore. Dans bon nombre de ces exemples et d'autres, les graphes de la vie réelle sont accompagnés de certaines caractéristiques/signaux, également appelés signaux sur graphe, sur l'ensemble des sommets. En voici quelques exemples :

- Réseaux de transport [HLDN19] : Les réseaux routiers en sont un exemple frappant. Dans une configuration typique, chaque emplacement correspond à un nœud, et deux nœuds sont reliés par une arête lorsqu'une route les relie. Les signaux sur ces réseaux sont généralement des caractéristiques du trafic, telles que la vitesse ou le volume du trafic.

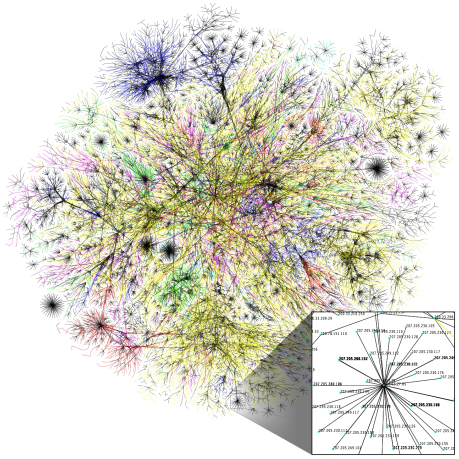
- Réseaux sociaux [VMS21] : Dans de nombreuses études sociales, outre les informations individuelles sur les sujets, la structure de leurs interactions est également d'une importance capitale. Les réseaux de médias sociaux tels que Facebook et Twitter sont probablement les exemples les plus connus. Dans ces cas, chaque utilisateur est considéré comme un nœud et deux nœuds sont connectés s'il existe une interaction entre les utilisateurs correspondants, comme des amis sur Facebook ou des followers sur Twitter. Compte tenu de cette structure de graphe/réseau, toute caractéristique d'un utilisateur, telle que ses goûts, peut être analysée comme un signal de graphe.



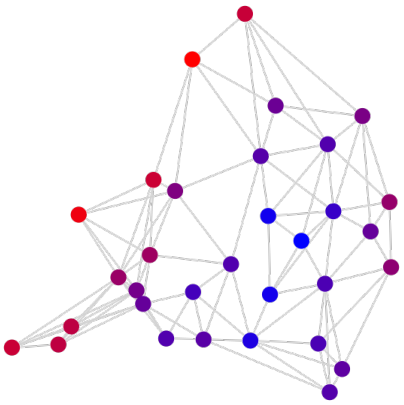
(a) Données de trafic recueillies par un réseau de capteurs à Grenoble [WMOKBB]



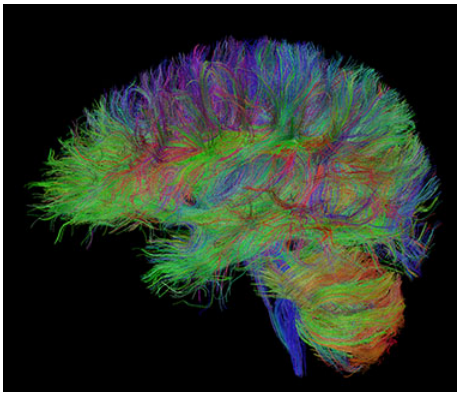
(b) Réseau d'amitié Twitter des auteurs des tweets que l'utilisateur @PilavciYigit a aimé jusqu'au 29/06/2022.



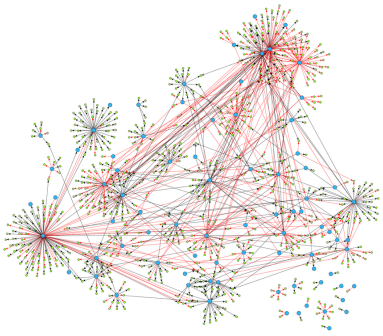
(c) Une carte partielle d'Internet



(d) Réseau de capteurs de température sur la Bretagne/France [PV17]. La couleur rouge correspond à des températures moyennes plus élevées.



(e) Connectivité du cerveau humain [HBM-BRV17]



(f) Gene regulatory network [MKGS14]

Fig. 5.1.: *Exemples de graphes dans la vie réelle*

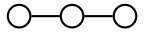
- Réseaux de température [PV17] : Un réseau de température est un réseau de capteurs de température sur une région dans laquelle chaque capteur est un nœud. Ici, les arêtes peuvent être établies selon différents critères. La pratique courante est de construire le graphe des plus proches voisins à partir des proximités des capteurs. Le signal, dans ce cas, est naturellement les mesures de température recueillies par les capteurs.

- Réseaux neurologiques [HBMBRV17] : Compte tenu des nombreuses questions restées sans réponse, l'analyse du cerveau est un sujet passionnant. L'étude basée sur les graphes, en fait, donne une façon naturelle d'analyser les signaux du cerveau. Dans ce type d'analyse, on considère souvent les différentes régions du cerveau comme des nœuds et les arêtes sont établies en fonction de leur proximité structurelle ou de propriétés fonctionnelles telles que la corrélation dans leurs activités. L'analyse des signaux neurologiques sur un tel réseau permet de déchiffrer des informations intéressantes sur le cerveau.

- Autres réseaux biologiques [GDJSRLHVRT+21; CPFSC21; YB20] : Le cerveau n'est pas seulement le sujet biologique de l'analyse signal sur graphes. Dans la littérature, nous trouvons de nombreuses études liées à la découverte de médicaments, à l'analyse des structures protéiques ou des interactions génétiques.

Compte tenu du grand volume et de la diversité de ces ensembles de données, le développement d'outils appropriés pour les traiter et les analyser est devenu problématique. En fait, la recherche en apprentissage automatique et en traitement du signal pour les graphes et les signaux de graphes a émergé au cours des dernières décennies [OFKMV18; RBTGV19]. À leur tour, ces outils ont été utilisés pour résoudre des problèmes de la vie réelle allant de la détection de fausses nouvelles dans les médias sociaux [MFEMB19] à l'analyse de la propagation pour COVID-19 [PNV21], du décodage des signaux du cerveau [OTLFPG22] aux prévisions météorologiques [Kei22], etc. Dans la plupart de ces outils et méthodes, les représentations matricielles des graphes, notamment le laplacien des graphes, revêtent une importance significative.

Un graphe path



Son Laplacian L

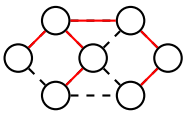
$$\begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

Laplacien de Graphe De nombreux outils d'analyse extraient des informations des représentations matricielles des graphes, en particulier du Laplacien de graphe. Le laplacien est une matrice carrée symétrique de la taille du nombre de sommets. Il

peut être considéré comme l'opérateur laplacien discret sur les graphes lorsqu'il est appliqué à une fonction sur les sommets :

$$(\mathsf{L}f)_i = \sum_{j \in \mathcal{N}(i)} (f(i) - f(j)),$$

où $\mathcal{N}(i)$ désigne les nœuds qui partagent un bord avec le nœud i (la définition formelle se trouve section 2.1.1). Le laplacien est une représentation matricielle simple mais utile des graphes. L'analyse algébrique le relie à de nombreuses propriétés utiles des graphes. Par exemple, de nombreuses tâches dans l'analyse de la connectivité du graphe peuvent être accomplies en observant la décomposition de la matrice du Laplacien du graphe. En regardant simplement la multiplicité des valeurs propres nulles, on peut compter le nombre de composantes connectées. De plus, la deuxième plus petite valeur propre, également appelée écart spectral, donne une mesure de la connectivité du graphe, qui est principalement utilisée dans les applications de stabilité et de robustesse des réseaux dynamiques. De même, le vecteur propre correspondant, également connu sous le nom de vecteur de Fiedler, donne la plus petite partition du graphe, celle qui laisse le plus petit nombre d'arêtes entre les parties. Le dessin spectral de graphes est un autre problème qui utilise la décomposition des vecteurs propres du laplacien. Dans ce problème, on cherche un dessin (une correspondance des sommets aux coordonnées euclidiennes) d'un graphe qui minimise la distance entre les sommets qui sont proches les uns des autres dans le graphe. Il s'avère que certains vecteurs propres du Laplacien sont les solutions analytiques de ce problème. Plus proche des thèmes principaux de cette thèse, nous trouvons un lien intéressant entre les valeurs propres de L et l'énumération d'un certain type de sous-graphes. En particulier, le célèbre théorème de l'arbre matriciel de Kirchoff stipule que le produit des valeurs propres non nulles de L est égal au nombre de tous les arbres couvrants, *i.e.* sous-graphes connectés qui ne contiennent pas de cycles, sur un graphe.



Un arbre couvrant. Ses arêtes sont indiquées en rouge, et les lignes pointillées indiquent les arêtes du graphe.

En dehors de ces exemples, le laplacien joue un rôle important dans l'analyse des réseaux de résistances électriques. Dans ces réseaux, nous modélisons chaque résistance par une arête où le poids de l'arête représente la conductance de la résistance, et chaque nœud correspond à ses points de connexion. Ces modèles sont largement utilisés dans la théorie des circuits afin de calculer les courants circulant à travers les résistances/arêtes ou les tensions induites aux nœuds lorsqu'une différence de potentiel fixe (tension) est appliquée entre deux nœuds (ou plus). Étonnamment, ces calculs se résument à la résolution de systèmes linéaires impliquant le laplacien du graphe (voir le chapitre 4).

Au cours des dernières décennies, des exemples encore plus nombreux sont apparus dans le traitement du signal et l'apprentissage automatique impliquant des graphes.

Laplacien dans le traitement du signal des graphes. Le GSP est le sous-domaine du traitement du signal qui traite des signaux définis sur les sommets du graphe. Au cours de la dernière décennie, de nombreux outils classiques de traitement du signal ont été adaptés afin de traiter de tels signaux. Citons par exemple le filtrage [SVF11; SNFOV13], l'échantillonnage [TAB17; PTGV18], l'opération de translation [SNFOV13], l'analyse en ondelettes [HVG11; NO12; ACGM20] ou le principe d'incertitude [TBD16]. Dans beaucoup de ces adaptations, la *transformation de Fourier de graphe* joue un rôle important. Par analogie, elle nous permet de représenter les signaux du graphe dans le domaine de fréquence du graphe. À son tour, on peut définir des schémas de traduction et de filtrage des graphes en utilisant un analogue du célèbre théorème de convolution [OBS01]. Cependant, en raison de l'irrégularité du domaine du signal, la définition de la transformée de Fourier n'est pas unique. Une définition populaire de la PSG repose sur le fait que les fonctions propres de l'opérateur Laplacien sont les fonctions de base de la transformée de Fourier. En s'appuyant sur ce principe, les auteurs de l'article [SNFOV13] suggèrent d'utiliser les vecteurs propres du laplacien du graphe comme base de Fourier et les valeurs propres comme fréquences du graphe. De cette façon, des valeurs propres plus élevées correspondent à une composante de fréquence plus élevée dans l'analyse de Fourier. En projetant le signal sur ces bases, on peut calculer la réponse en fréquence du graphique du signal à chaque composante.

Laplacien en l'apprentissage automatique. Une des premières utilisations du laplacien se produit dans l'apprentissage semi-supervisé sur des graphes [Zhu05; AMGS12]. Dans ce problème, on nous donne quelques étiquettes sur les sommets et le but est de déduire les étiquettes des autres sommets en utilisant les étiquettes données et le graphe sous-jacent. Une solution de base donnée par [Zhu05] suggère d'utiliser le Laplacien du graphe pour formuler ce problème. Leur principale hypothèse est que la fonction d'étiquetage f est fixée à un sous-ensemble $V \subseteq \mathcal{V}$ et lisse sur le reste $U = \mathcal{V} \setminus V$, *i.e.* f ne varie pas trop à travers les bords. Pour deux classes d'étiquettes (pour simplifier), cette formulation cherche une fonction de classification f qui n'est connue que sur un petit ensemble de sommets $V \subseteq \mathcal{V}$.

Désignons les étiquettes connues par $\mathbf{y} \in \mathbb{R}^{|V|}$, puis, [Zhu05] suggérons de résoudre le problème contraint suivant :

$$f^\star = \operatorname*{argmin}_f \sum_{i \in U} \sum_{j \sim i} (f(i) - f(j))^2$$
$$\text{s. t. } \forall i \in \mathcal{V}, f(i) = y_i,$$

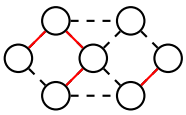
En fait, le terme minimisé peut être écrit en termes de laplacien du graphe comme $\mathbf{f}^\top \mathsf{L}_U \mathbf{f}$ où $\mathbf{f} = [f(i)]_{i \in U}$. De plus, la solution sous forme fermée implique ce qui suit dans le Laplacien du graphe :

$$f^\star = (\mathsf{L}_U)^{-1} \mathsf{L}_{U|V} \mathbf{f}.$$

Plus tard, cette formulation est considérée dans un cadre non contraint et généralisée dans [AMGS12]. Dans ces formes également, la solution prend la forme de l'inverse matriciel du laplacien du graphe. Parmi les exemples plus récents en apprentissage automatique qui utilisent largement le laplacien, citons le clustering spectral [Von07; TPGV16], l'intégration de graphes [Xu21], la sparsification [SS11] et l'apprentissage profond sur les graphes [WPCLZP20]. Dans toutes ces applications et bien d'autres, l'analyse de L est de la plus haute importance.

Algèbre numérique basée sur le Laplacien. Comme indiqué ci-dessus, le nombre d'applications impliquant le Laplacien des graphes est grand. Dans beaucoup de ces applications, la solution nécessite le calcul d'une décomposition inverse (régularisée ou pseudo) ou d'une décomposition de l'indice du Laplacien. Cependant, le calcul direct de ces quantités ne s'adapte pas bien à la taille du graphe et devient même impossible pour les très grands graphes. En conséquence, de nombreux chercheurs se sont attachés à développer des outils d'algèbre numérique spécifiques au laplacien. Les plus importants sont étroitement liés à la théorie spectrale des graphes [Spi12] et à la résolution de systèmes linéaires laplaciens [Vis+13]. Ces études donnent une riche collection d'outils algébriques qui évitent le calcul direct coûteux et, à la place, approchent la solution requise dans certaines plages d'erreur. Ce faisant, elles tirent parti de certaines propriétés communes du laplacien des graphes, telles que leurs propriétés algébriques, leurs liens étroits avec la théorie des graphes ou la rareté des graphes réels. Par rapport à ces travaux, nous prenons dans cette thèse une voie alternative en utilisant la randomisation afin de développer des outils algébriques spécifiques au Laplacien. Nous approchons la solution requise dans certains de ces problèmes en exploitant les riches connexions théoriques entre les Laplaciens et un processus aléatoire sur les graphes.

Algèbre Linéaire Aléatoire (ALA). En bref, l'ALA est un ensemble d'outils qui évitent le calcul direct d'une opération algébrique par le biais de la randomisation. Le principal avantage de ces outils est qu'ils permettent d'approximer des opérations très coûteuses, telles que l'inversion de matrice et la décomposition de la valeur singulière, de manière très économique en prenant uniquement des échantillons aléatoires de la matrice d'entrée. Plus de détails et d'exemples peuvent être trouvés dans [DM16]. Plus proche des thèmes de cette thèse, [DM21] montrent que les processus ponctuel déterminantaux (PPD), *i.e.* un processus ponctuel aléatoires avec de nombreuses propriétés tractables, peuvent être facilement adaptés comme algorithmes ALA afin de résoudre des systèmes linéaires impliquant un large ensemble de matrices d'entrée couvrant le Laplacien du graphe. Inspirés par ces exemples et bien d'autres, nous nous concentrons sur un processus ponctuel déterminantal particulier qui nous permet de développer des outils randomisés pour l'algèbre numérique basée sur le Laplacien.



Une forêt d'envergure avec trois arbres. Les arêtes de la forêt sont en rouge, et le graphe est représenté par des lignes pointillées.

Les Forêts Couvrantes Aléatoire (FCA) Une forêt dans un graphe est un sous-graphe qui ne contient aucun cycle. Elle est dite couvrante si elle contient tous les sommets du graphe. Enfin, les forêts couvrantes aléatoires sont les processus aléatoires dans lesquels nous choisissons une forêt étendue au hasard sur un graphe donné. On peut imaginer une infinité d'options pour la distribution d'un tel processus. Dans cette thèse, nous nous concentrons sur la distribution introduite dans [ACGM18]. Dans cette distribution, la probabilité d'avoir une forêt couvrante est principalement régulée par les poids de ses arêtes et le nombre de ses composants connectés (arbres). Nous nous limitons à cette distribution pour deux raisons principales :

- Les FCAs échantillonnés à partir de cette distribution ont des connexions théoriques riches avec le Laplacien des graphes via certains PPDs,
- Il existe un algorithme efficace pour échantillonner à partir de cette distribution, appelé algorithme de Wilson.

Grâce à ces faits, nous savons que les échantillons de FCAs peuvent être obtenus de manière peu coûteuse, et que leurs propriétés algébriques sont étroitement liées aux propriétés algébriques du graphe original. L'objectif principal de cette thèse est d'exploiter ces faits afin de développer des algorithmes ALA pour l'algèbre numérique basée sur le Laplacien.

Les principales contributions de cette thèse peuvent être énumérées comme suit :

- Nous mettons en lumière les riches connexions des FCAs avec les processus ponctuels déterminant aux et le Laplacien des graphes ; Nous passons en

revue l'élégante théorie et les propriétés utiles des FCA en tant que PPD. Tout au long de notre tour, alors que pour certaines des propriétés, nous reproduisons les preuves existantes, pour d'autres, nous donnons surtout des preuves algébriques qui sont plus accessibles pour les chercheurs de différents domaines (Voir Théorèmes 2.4.4, 2.4.5).

- En retour, nous développons et analysons des algorithmes ALA basés sur les FCAs pour approximer les solutions d'une grande variété de problèmes impliquant le Laplacien des graphes. Ces problèmes sont :

  – **Régularisation et interpolation de Tikhonov pour les signaux de graphes** : Les problèmes de débruitage, d'élimination des parties bruyantes d'un signal et de complétion des parties manquantes d'un signal sont des problèmes de longue date en traitement du signal. Nous donnons d'abord un cadre d'optimisation unifié pour ces problèmes pour les signaux de graphes *i.e.* signaux définis sur des sommets. Ensuite, nous donnons de nouveaux algorithmes aléatoires qui approchent la solution de ce cadre.

  – **Estimation de la trace de l'inverse régularisé des matrices symétriques diagonalement dominantes (SDD)** : La trace (somme des entrées diagonales d'une matrice) est une opération centrale en algèbre linéaire. Cependant, le calcul de la trace d'une matrice est un problème difficile lorsque la matrice d'entrée n'est pas directement accessible eg multiplication de grandes matrices ou inverse d'une grande matrice. Nous donnons de nouveaux algorithmes basés sur les FCA pour estimer l'inverse régularisé des matrices SDD (une classe de matrices qui contiennent le Laplacien du graphe). En retour, nos algorithmes sont au moins comparables, et ils surpassent généralement les méthodes de l'état de l'art.

  – **Estimation des résistances effectives dans les réseaux électriques** : La résistance effective est une métrique qui prend sa source dans les réseaux de résistances électriques[1] et ils sont utilisés pour mesurer la similarité d'une paire de nœuds. Ils jouent un rôle central dans de nombreuses applications liées aux graphes, telles que la sparsification, le regroupement ou l'apprentissage des graphes. Cependant, le calcul de cette quantité utile ne s'adapte pas bien à l'augmentation de la taille des graphes, car il faut inverser un laplacien de graphe. Pour éviter ce calcul coûteux, nous donnons des algorithmes basés sur le FCA pour estimer

[1] La représentation graphique d'un réseau de résistances électriques prend chaque bord comme une résistance avec une conductance unitaire. La conductance est égale au poids de l'arête pour les graphes pondérés.

les résistances effectives. Les méthodes proposées sont comparables et même meilleures que les algorithmes de l'état de l'art lorsque le nombre de résistances effectives souhaitées est faible.

  - **Estimation de divers filtres graphiques:** Cet aspect de notre travail établit des liens intéressants entre les FCAs et le filtrage sur graphes *i.e.* spectral signal filtering for graph signals [TGB18]. Le filtrage sur graphes est un outil essentiel pour traiter ce type de signaux et a été utilisé dans diverses applications. Pourtant, les calculs associés nécessitent souvent de diagonaliser L, ce qui est coûteux. Dans ce travail, nous donnons un ensemble de filtres graphiques qui peuvent être approximés via des estimateurs basés sur le FCAs en évitant la diagonalisation de L. Ces filtres couvrent ou peuvent imiter en optimisant quelques hyperparamètres certains des filtres fréquemment utilisés, tels que le filtre passe-bas idéal.

- Nous fournissons une analyse biais-variance des algorithmes proposés afin d'énoncer leurs performances théoriques.

- Nous illustrons ces algorithmes dans des applications et des ensembles de données réels tout en comparant ces algorithmes avec les algorithmes existants.

La structure principale de la thèse est la suivante ; nous commençons par donner les outils techniques nécessaires dans le Chapitre 2 sur la théorie des graphes, l'algèbre linéaire aléatoire et les forêts couvrantes aléatoire. Ensuite, au Chapitre 3, nous introduisons les méthodes basées sur les FCAs pour résoudre la régularisation de Tikhonov sur les graphes. Nous comparons ces méthodes aux méthodes existantes et montrons leurs utilisations dans d'autres problèmes liés aux graphes tout en illustrant des applications réelles. Dans le Chapitre 4, nous étendons la gamme des méthodes basées sur les FCAs à un nouvel ensemble de problèmes, notamment l'estimation de la trace de l'inverse régularisée des matrices SDD, les résistances effectives et certains filtres de graphes. Enfin, nous terminons dans le chapitre 5 avec une conclusion générale, une discussion sur les questions ouvertes et les travaux futurs.

Les publications évaluées par les pairs associées à cette thèse peuvent être trouvées dans ce qui suit :

- Yusuf Yigit Pilavci et al. "Variance Reduction for Inverse Trace Estimation via Random Spanning Forests". In: *GRETSI 2022 - XXVIIIème Colloque Francophone de Traitement du Signal et des Images*. Nancy, France, Sept. 2022

- Yusuf Pilavcı et al. “Variance reduction in stochastic methods for large-scale regularised least-squares problems”. In: *arXiv preprint arXiv:2110.07894* (2021)
- Yusuf Yiğit Pilavcı et al. “Graph tikhonov regularization and interpolation via random spanning forests”. In: *IEEE transactions on Signal and Information Processing over Networks* 7 (2021), pp. 359–374
- Yusuf Y Pilavci et al. “Smoothing graph signals via random spanning forests”. In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 5630–5634

De plus, certains matériaux sont présentés dans ma thèse de maîtrise [PIL19].

List of Figures

Appendix A

A.1 Upper-bound on the Complexity of Wilson's algorithm for Forests

Here we give an upper-bound on the trace $\mathrm{tr}((\mathsf{Q}+\mathsf{L})^{-1}(\mathsf{Q}+\mathsf{D}))$ which gives the expected number of steps taken by Wilson's algorithm. First, we write this trace explicitly as follows:

$$\mathrm{tr}((\mathsf{Q}+\mathsf{L})^{-1}(\mathsf{Q}+\mathsf{D})) = \sum_{i=1}^{n}(q_i+d_i)(\mathsf{Q}+\mathsf{L})^{-1}_{i,i} = \sum_{i=1}^{n}(q_i+d_i)\frac{\mathsf{K}_{i,i}}{q_i}.$$

Then, by Theorem 2.4.4, we know that $\mathsf{K}_{i,i} = \mathbb{P}(i \in \rho(\Phi_Q)) \leq 1$. By plugging this upper bound on $\mathsf{K}_{i,i}$'s, one obtains:

$$\mathrm{tr}((\mathsf{Q}+\mathsf{L})^{-1}(\mathsf{Q}+\mathsf{D})) \leq \sum_{i=1}^{n}\frac{q_i+d_i}{q_i} = n + \sum_{i=1}^{n}\frac{d_i}{q_i} \leq n + \frac{2m}{q_{min}}.$$

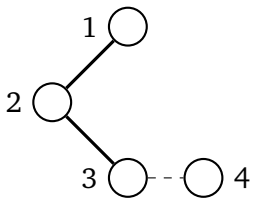
A.2 Proof of Proposition 2.1.2

The proof of this theorem is two-fold:

- In the first part, we show $|\det \mathsf{B}_{S|-m}| = \left[\prod_{(i,j)\in S} w(i,j)\right]^{1/2}$, $\forall m \in \mathcal{V}$ in case that S forms a spanning tree.
- In the second, we prove that $|\det \mathsf{B}_{S|-m}| = 0$ if S forms a subgraph includes a cycle.

The first part is proven by induction in which we will show that the statement holds for a base case and an inductive step. We consider two nodes connected with an edge e as the base case. The statement is true for the base case since $|\det \mathsf{B}_{e|-1}| = w(e)$ where $m = 1$ without loss of generality. This leaves us to examine the inductive step. At each inductive step, we add a node and connect it to one of the others (See Fig. A.1). Note that this inductive step can build any possible tree and it only

constructs trees. Without loss of generality, each newly added node is connected to the one added in the previous step. We denote the edge set at k-step as $\mathcal{E}_k$. To show that the statement is true for an inductive step, at first, we assume that it holds for k-th step. Under this assumption, proving that it also holds for $k+1$-th step confirms that it holds for the induction step and finishes the first part.



$$\mathsf{B}_{\mathcal{E}_3|-1} = \begin{bmatrix} \sqrt{w(1,2)} & 0 & 0 \\ -\sqrt{w(2,3)} & \sqrt{w(2,3)} & 0 \\ 0 & -\sqrt{w(3,4)} & \sqrt{w(3,4)} \end{bmatrix}$$

Fig. A.1.: *A weighted graph and its reduced edge incidence matrix*

At $k+1$-th step, the reduced edge incidence matrix $\mathsf{B}_{\mathcal{E}_{k+1}|-1}$ reads:

$$\mathsf{B}_{\mathcal{E}_{k+1}|-1} = \begin{bmatrix} & & 0 \\ & \mathsf{B}_{\mathcal{E}_k|-1} & 0 \\ & & \vdots \\ & & 0 \\ 0 \quad \ldots & -\sqrt{w(k+1,k+2)} & \sqrt{w(k+1,k+2)} \end{bmatrix} \tag{A.1}$$

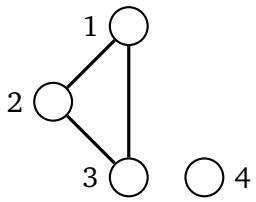
By applying the block matrix determinant formula, we have:

$$|\det(\mathsf{B}_{\mathcal{E}_{k+1}|-1})| = \sqrt{w(k+1,k+2)}|\det \mathsf{B}_{\mathcal{E}_k|-1}| \tag{A.2}$$

Given $|\det \mathsf{B}_{\mathcal{E}_k|-1}| = \left[\prod_{(i,j)\in\mathcal{E}_k} w(i,j)\right]^{1/2}$, one easily obtains:

$$\begin{aligned} |\det \mathsf{B}_{\mathcal{E}_{k+1}|-1}| &= \sqrt{w(k+1,k+2)}\left[\prod_{(i,j)\in\mathcal{E}_k} w(i,j)\right]^{1/2} \\ &= \left[\prod_{(i,j)\in\mathcal{E}_{k+1}} w(i,j)\right]^{1/2} \end{aligned}$$

which proves the statement for the induction step and completes the first part.



$$\mathsf{B}_{\{\mathcal{E}_3|-1\}} = \begin{bmatrix} -w(1,2) & 0 & 0 \\ w(2,1) & -w(2,1) & 0 \\ 0 & w(3,1) & 0 \end{bmatrix}$$

Fig. A.2.: *A cycle and its reduced edge incidence matrix*

In the second part, we examine the case that S includes a cycle. Imagine that we build a circuit in a similar fashion. Then, whenever we construct a cycle, the corresponding reduced edge incidence matrix contains a singular principal block (See Fig. A.2). Thus, the resultant determinant $|\det \mathsf{B}_{\mathcal{E}_k|-1}|$ is equal to zero.

A.3 Intermediate steps in the proof of Theorem 2.4.4

Let us rewrite as $\mathsf{S} = (\alpha^{-1}\mathsf{I} + \mathsf{A})^{-1}$ where $\mathsf{A} = \mathsf{Q} - \mathsf{Q}^{1/2}\mathsf{B}^\top(\mathsf{B}\mathsf{B}^\top + \alpha^{-1}\mathsf{I})^{-1}\mathsf{B}\mathsf{Q}^{1/2}$. Then, the matrix M reads:

$$\mathsf{M} = \lim_{\alpha\to\infty} \mathsf{A}(\mathsf{A} + \alpha^{-1}\mathsf{I})^{-1}.$$

In order to prove the equality in Eq. (2.22), we re-arrange the terms in the matrix A by using some matrix inverse identities. The first identity we apply is $(\mathsf{X}+\mathsf{Y}\mathsf{Y}^\top)^{-1}\mathsf{Y} = \mathsf{X}^{-1}\mathsf{Y}(\mathsf{I} + \mathsf{Y}^\top\mathsf{X}^{-1}\mathsf{Y})^{-1}$ where X is a invertible square matrix. By plugging $\mathsf{X} = \alpha^{-1}\mathsf{I}$ and $\mathsf{Y} = \mathsf{B}$, we write A as follows:

$$\mathsf{A} = \mathsf{Q} - \alpha\mathsf{Q}^{1/2}\mathsf{B}^\top\mathsf{B}(\mathsf{I} + \alpha\mathsf{B}^\top\mathsf{B})^{-1}\mathsf{Q}^{1/2} = \mathsf{Q}^{1/2}(\mathsf{I} - \mathsf{L}(\alpha^{-1}\mathsf{I} + \mathsf{L})^{-1})\mathsf{Q}^{1/2}.$$

Noticing $\mathsf{I} - \mathsf{L}(\alpha^{-1}\mathsf{I} + \mathsf{L})^{-1} = (\mathsf{I} + \alpha\mathsf{L})^{-1}$ even more simplifies:

$$\mathsf{A} = \mathsf{Q}^{1/2}(\mathsf{I} + \alpha\mathsf{L})^{-1}\mathsf{Q}^{1/2}.$$

Let us go back to the original limit. Another matrix trick allows to write:

$$\mathsf{M} = \lim_{\alpha\to\infty} \mathsf{I} - (\mathsf{I} + \alpha\mathsf{A})^{-1}.$$

Plugging A as found above gives:

$$\mathsf{M} = \lim_{\alpha\to\infty} \mathsf{I} - (\mathsf{I} + \mathsf{Q}^{1/2}(\alpha^{-1}\mathsf{I} + \mathsf{L})^{-1}\mathsf{Q}^{1/2})^{-1}.$$

The final manipulation we do is due to the identity $(\mathsf{I}+\mathsf{XY})^{-1} = \mathsf{I} - \mathsf{X}(\mathsf{I}+\mathsf{YX})^{-1}\mathsf{Y}$. We re-write the final matrix inverse by setting $\mathsf{X} = \mathsf{Q}^{1/2}$ and $\mathsf{Y} = (\alpha^{-1}\mathsf{I}+\mathsf{L})^{-1}\mathsf{Q}^{1/2}$. By doing so, we finish this proof as follows:

$$\begin{aligned}\mathsf{M} &= \lim_{\alpha\to\infty} \mathsf{I} - \mathsf{I} + \mathsf{Q}^{1/2}(\mathsf{I}+(\alpha^{-1}\mathsf{I}+\mathsf{L})^{-1}\mathsf{Q})^{-1}(\alpha^{-1}\mathsf{I}+\mathsf{L})^{-1}\mathsf{Q}^{1/2}. \\ &= \lim_{\alpha\to\infty} \mathsf{Q}^{1/2}(\alpha^{-1}\mathsf{I}+\mathsf{L}+\mathsf{Q})^{-1}\mathsf{Q}^{1/2} = \mathsf{Q}^{1/2}(\mathsf{L}+\mathsf{Q})^{-1}\mathsf{Q}^{1/2}.\end{aligned}$$

A.4 Proof of Lemma A.4.1

Lemma A.4.1. *Given the extended graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', w)$, for all edge subsets $\mathcal{S} \subseteq \mathcal{E}'$ with $|\mathcal{S}| = |\mathcal{V}'| - 2 = n - 1$, the following inequality holds:*

$$(-1)^{i+j} \det \mathsf{B}'_{\{\mathcal{S}|-i,-\Gamma\}} \det \mathsf{B}'_{\{\mathcal{S}|-j,-\Gamma\}} \geq 0 \quad \forall i,j \in \mathcal{V}$$

.

Proof. Consider the columns of matrix $\mathsf{B}'_{\{\mathcal{S}|-i,-\Gamma\}} = (\mathbf{b}_1|\dots|\mathbf{b}_{i-1}|\mathbf{b}_{i+1}|\dots|\mathbf{b}_j|\dots|\mathbf{b}_{|\mathcal{V}|})$ where $i<j$ assumed without loss of generality. Then, with a certain column permutation, one can find:

$$\operatorname{perm}(\mathsf{B}'_{\{\mathcal{S}|-i,-\Gamma\}}) = \mathsf{P}_i \mathsf{B}'_{\{\mathcal{S}|-i,-\Gamma\}} = \begin{bmatrix}\mathbf{b}_j & \mathsf{X}\end{bmatrix}$$

where P_i is the associated permutation matrix. A similar can be done for $\mathsf{B}'_{\{\mathcal{S}|-j,-\Gamma\}}$ as $\operatorname{perm}(\mathsf{B}'_{\{\mathcal{S}|-j,-\Gamma\}}) = \mathsf{P}_j \mathsf{B}'_{\{\mathcal{S}|-j,-\Gamma\}} = [\mathbf{b}_i \quad \mathsf{X}]$. Note that the determinants of these permutation matrices read $\det \mathsf{P}_i = (-1)^{j-1}$ and $\det \mathsf{P}_j = (-1)^i$ for $i<j$. Then, one can rewrite the initial determinant product as:

$$\begin{aligned}\det \mathsf{B}'_{\{\mathcal{S}|-i,-\Gamma\}} \det \mathsf{B}'_{\{\mathcal{S}|-j,-\Gamma\}} &= (-1)^{j-1} \det \operatorname{perm}(\mathsf{B}'_{\{\mathcal{S}|-i,-\Gamma\}})(-1)^i \det \operatorname{perm}(\mathsf{B}'_{\{\mathcal{S}|-j,-\Gamma\}}) \\ &= (-1)^{i+j-1} \det \begin{bmatrix}\mathbf{b}_j & \mathsf{X}\end{bmatrix}^T \begin{bmatrix}\mathbf{b}_i & \mathsf{X}\end{bmatrix} \\ &= (-1)^{i+j-1} \det \begin{bmatrix}\mathbf{b}_j^T\mathbf{b}_i & \mathbf{b}_j^T\mathsf{X} \\ \mathsf{X}^T\mathbf{b}_i & \mathsf{X}^T\mathsf{X}\end{bmatrix}\end{aligned}$$

Due to this, showing $\det \begin{bmatrix} \mathbf{b}_j^T \mathbf{b}_i & \mathbf{b}_j^T \mathsf{X} \\ \mathsf{X}^T \mathbf{b}_i & \mathsf{X}^T \mathsf{X} \end{bmatrix} \leq 0$ implies that the sign of the overall product is $(-1)^{i+j}$ and finishes the proof:

$$\begin{aligned} \det \begin{bmatrix} \mathbf{b}_j^T \mathbf{b}_i & \mathbf{b}_j^T \mathsf{X} \\ \mathsf{X}^T \mathbf{b}_i & \mathsf{X}^T \mathsf{X} \end{bmatrix} &= \det \mathsf{X}^T \mathsf{X} \det(\mathbf{b}_j^T \mathbf{b}_i - \mathbf{b}_j^T \mathsf{X} (\mathsf{X}^T \mathsf{X})^{-1} \mathsf{X}^T \mathbf{b}_i) \\ &= \det \mathsf{X}^T \mathsf{X} \left[\mathbf{b}_j^T \mathbf{b}_i - \mathbf{b}_j^T \mathsf{X} (\mathsf{X}^T \mathsf{X})^{-1} \mathsf{X}^T \mathbf{b}_i \right] \quad \text{(Determinant of a scalar)} \end{aligned} \tag{A.3}$$

$\mathsf{X}^T \mathsf{X}$ is a semi-positive definite matrix, thus $\det \mathsf{X}^T \mathsf{X} \geq 0$. By definition, the vector product $\mathbf{b}_k^T \mathbf{b}_l$ writes $\forall k, l \in \mathcal{V}$:

$$\mathbf{b}_k^T \mathbf{b}_l = \begin{cases} \sum_{n \in \mathcal{V}} w(k,m) & \text{if, } k = l \\ -w(k,l) & \text{if, } k \neq l \text{ and } (k,l) \in \mathcal{E} \cup \Gamma \\ 0 & \text{otherwise} \end{cases}$$

Due to this, $\mathbf{b}_j^T \mathbf{b}_i$ and all entries of both $\mathbf{b}_j^T \mathsf{X}$ and $\mathsf{X}^T \mathbf{b}_i$ are non-positive. Moreover, $\mathsf{X}^T \mathsf{X}$ writes a M-matrix whose inverse is entry-wise non-negative. Summing these up gives $\mathbf{b}_j^T \mathbf{b}_i \leq 0$, $(\mathbf{b}_j^T \mathsf{X} (\mathsf{X}^T \mathsf{X})^{-1} \mathsf{X}^T \mathbf{b}_i) \geq 0 \; \forall i, j \in \mathcal{V}$ and

$$\det \begin{bmatrix} \mathbf{b}_j^T \mathbf{b}_i & \mathbf{b}_j^T \mathsf{X} \\ \mathsf{X}^T \mathbf{b}_i & \mathsf{X}^T \mathsf{X} \end{bmatrix} = \det \mathsf{X}^T \mathsf{X} \left[\mathbf{b}_j^T \mathbf{b}_i - \mathbf{b}_j^T \mathsf{X} (\mathsf{X}^T \mathsf{X})^{-1} \mathsf{X}^T \mathbf{b}_i \right] \leq 0$$

thus,

$$\begin{aligned} (-1)^{i+j} \det \mathsf{B}'_{\{\mathcal{S}|-i,-\Gamma\}} \det \mathsf{B}'_{\{\mathcal{S}|-j,-\Gamma\}} &= (-1)^{2i+2j-1+1} \left| \det \begin{bmatrix} \mathbf{b}_j^T \mathbf{b}_i & \mathbf{b}_j^T \mathsf{X} \\ \mathsf{X}^T \mathbf{b}_i & \mathsf{X}^T \mathsf{X} \end{bmatrix} \right| \\ &= \left| \det \begin{bmatrix} \mathbf{b}_j^T \mathbf{b}_i & \mathbf{b}_j^T \mathsf{X} \\ \mathsf{X}^T \mathbf{b}_i & \mathsf{X}^T \mathsf{X} \end{bmatrix} \right| \geq 0 \end{aligned}$$

□

A.5 Upper bounds on the terms in Eq. (4.6)

Upper bound on $\sum_{i,j \in \mathcal{V}} \mathrm{Var}(\bar{\mathsf{X}}_{\mathsf{i,j}})$. Let us start by plugging the definition of the variance:

$$\sum_{i,j \in \mathcal{V}} \mathrm{Var}(\bar{\mathsf{X}}_{\mathsf{i,j}}) = \sum_{i,j \in \mathcal{V}} \mathbb{E} \left[\frac{\mathbb{I}(r_{\Phi_q}(i) \neq r_{\Phi_q}(j), |\rho(\Phi_q)| = 2)}{|\mathcal{V}_{t(\Phi_q,i)}|^2 |\mathcal{V}_{t(\Phi_q,j)}|^2} \right] - c^2 \mathsf{R}_{i,j}^2.$$

Then rearranging the summation yields:

$$\sum_{i,j\in\mathcal{V}} \mathrm{Var}(\bar{\mathsf{X}}_{\mathsf{i,j}}) = \mathbb{E}\left[\sum_{i,j\in\mathcal{V}} \frac{\mathbb{I}(r_{\Phi_q}(i) \neq r_{\Phi_q}(j), |\rho(\Phi_q)| = 2)}{|\mathcal{V}_{t(\Phi_q,i)}|^2 |\mathcal{V}_{t(\Phi_q,j)}|^2}\right] - \sum_{i,j\in\mathcal{V}} c^2 \mathsf{R}_{i,j}^2$$

Notice that, the sum $\sum_{i,j\in\mathcal{V}} \frac{\mathbb{I}(r_{\Phi_q}(i)\neq r_{\Phi_q}(j), |\rho(\Phi_q)|=2)}{|\mathcal{V}_{t(\Phi_q,i)}|^2 |\mathcal{V}_{t(\Phi_q,j)}|^2}$ is only non-zero over the spanning forests with exactly two trees. Moreover, given an arbitrary forest ϕ with two roots v_1, v_2, it always sums up to:

$$\sum_{i,j\in\mathcal{V}} \frac{\mathbb{I}(r_\phi(i) \neq r_{\Phi_q}(j))}{|\mathcal{V}_{t(\phi,i)}|^2 |\mathcal{V}_{t(\phi,j)}|^2} = \sum_{i,j\in\mathcal{V}} \frac{\mathbb{I}(r_\phi(i) \neq r_\phi(j))}{|\mathcal{V}_{t(\phi,v_1)}|^2 |\mathcal{V}_{t(\phi,v_2)}|^2} = \frac{2}{|\mathcal{V}_{t(\phi,v_1)}||\mathcal{V}_{t(\phi,v_2)}|} \leq \frac{2}{n-1}.$$

Recall that $|\mathcal{V}_{t(\phi,v_1)}| + |\mathcal{V}_{t(\phi,v_2)}| = n$ for any graphs and both terms are positive integers. Thus, the minimum of $|\mathcal{V}_{t(\phi,v_1)}||\mathcal{V}_{t(\phi,v_2)}|$ equals to $n-1$. As a result, we obtain the following upper-bound:

$$\sum_{i,j\in\mathcal{V}} \mathrm{Var}(\bar{\mathsf{X}}_{\mathsf{i,j}}) \leq \frac{2}{n-1} - c^2 \sum_{i,j\in\mathcal{V}} \mathsf{R}_{i,j}^2.$$

Upper bound on $-\sum_{i,j\in\mathcal{V}} \mathsf{R}_{i,j} \,\mathrm{Cov}(\bar{\mathsf{X}}_{\mathsf{i,j}}, C)$. Notice that both $\bar{\mathsf{X}}_{i,j}$ and C are non-negative random variables. Then the following inequality holds:

$$\begin{aligned} -\sum_{i,j\in\mathcal{V}} \mathsf{R}_{i,j} \,\mathrm{Cov}(\bar{\mathsf{X}}_{\mathsf{i,j}}, C) &= -\sum_{i,j\in\mathcal{V}} \mathsf{R}_{i,j} (\mathbb{E}[C\bar{\mathsf{X}}_{i,j}] - \mathbb{E}[C]\mathbb{E}[\bar{\mathsf{X}}_{i,j}]) \\ &\leq \sum_{i,j\in\mathcal{V}} \mathsf{R}_{i,j} \mathbb{E}[C]\mathbb{E}[\bar{\mathsf{X}}_{i,j}] = c^2 \sum_{i,j\in\mathcal{V}} \mathsf{R}_{i,j}^2. \end{aligned}$$

Upper bound on $\sum_{i,j\in\mathcal{V}} \mathsf{R}_{i,j}^2 \,\mathrm{Var}(C)$. We again plug the definition of the variance and it yields:

$$\begin{aligned} \sum_{i,j\in\mathcal{V}} \mathsf{R}_{i,j}^2 \,\mathrm{Var}(C) &= \sum_{i,j\in\mathcal{V}} \mathsf{R}_{i,j}^2 \left[\mathbb{E}[C^2] \; - c^2\right] \\ &= \sum_{i,j\in\mathcal{V}} \mathsf{R}_{i,j}^2 \left(\frac{1}{(n-1)^2} \mathbb{E}[\sum_{(x,y)\in\mathcal{E}} \sum_{(i,j)\in\mathcal{E}} \bar{\mathsf{X}}_{x,y}, \bar{\mathsf{X}}_{a,b}] \; - c^2\right). \end{aligned} \tag{A.4}$$

By developing on the term within the expectation, one obtains:

$$\sum_{(x,y)\in\mathcal{E}} \sum_{(i,j)\in\mathcal{E}} \bar{\mathsf{X}}_{x,y}, \bar{\mathsf{X}}_{a,b} = \sum_{(x,y)\in\mathcal{E}} \sum_{(a,b)\in\mathcal{E}} \frac{\mathbb{I}(r_{\Phi_q}(a) \neq r_{\Phi_q}(b), r_{\Phi_q}(x) \neq r_{\Phi_q}(y), |\rho(\Phi_q)| = 2)}{|\mathcal{V}_{t(\Phi_q,a)}||\mathcal{V}_{t(\Phi_q,b)}||\mathcal{V}_{t(\Phi_q,x)}||\mathcal{V}_{t(\Phi_q,y)}|}.$$

The condition on the numerator is satisfied only if Φ_q has exactly two roots and the pairs (a, b) and (x, y) must be in different trees. Then for an arbitrary forest ϕ with exactly two trees rooted in v_1 and v_2, this sum boils down to:

$$\sum_{(x,y)\in\mathcal{E}} \sum_{(a,b)\in\mathcal{E}} \bar{\mathsf{X}}_{x,y}\bar{\mathsf{X}}_{a,b} = \sum_{(x,y)\in\mathcal{E}} \sum_{(a,b)\in\mathcal{E}} \frac{\mathbb{I}(r_\phi(a) \neq r_\phi(b), r_{\Phi_q}(x) \neq r_\phi(y))}{|\mathcal{V}_{t(\phi,v_1)}|^2 |\mathcal{V}_{t(\phi,v_2)}|^2}.$$

Assuming the condition is verified by all edge pairs[1], one reaches to the maximum value of this sum as follows:

$$\sum_{(x,y)\in\mathcal{E}} \sum_{(i,j)\in\mathcal{E}} \bar{\mathsf{X}}_{x,y}\bar{\mathsf{X}}_{a,b} \leq \frac{m^2}{|\mathcal{V}_{t(\phi,v_1)}|^2 |\mathcal{V}_{t(\phi,v_2)}|^2} \leq \frac{m^2}{(n-1)^2}.$$

Applying this upper bound on the expectation term in Eq. (A.4) concludes this section.

[1] In fact, this is only possible on a graph with two nodes and one edge. Otherwise, there must be at least one edge that connects nodes with the same root.