Improving graph learning algorithms for datasets containing patterns diffusing in time

Andrés Sánchez Centeno

14th June 2023
MSc thesis

Francisco Manuel Bernal Martínez, Thesis Supervisor

Universidad Carlos III de Madrid (UC3M)

Abstract

In this thesis we cover a graph learning algorithm, described in [31], that retrieves the underlying topology of a graph from signals that have been diffused through its edges over time. After presenting the algorithm in a state-of-the-art context and covering the optimization techniques on which the algorithm is based, we discuss the usage of prior knowledge, the number of edges of the groundtruth graph, to do post-processing of the edges of the graph before evaluating the information retrieval results. We explain how, without those priors, the task of retrieving the binary information—guessing which nodes are connected and which not—is extremely difficult, and we propose two techniques to threshold the number of learned edges in a way in which we make a minimal assumption on the underlying. Then we show how we can easily modify the algorithm to include prior knowledge over the edges and review the results obtained with different percentages of prior knowledge used.

Key words: graph signal processing, graph learning, heat kernel, graph Laplacian, cone programming, alternating direction method of multipliers, sparse signals, sparse representation, matrix exponential, binary classification.

Contents

Agradecimientos

Muchas gracias a mis dos profesores. A Francisco por preocuparse porque aprenda para el futuro y a Juan Acebrón, que ha supervisado también esta tesis (aunque no he podido ponerlo). Gracias por intentar meterme en la cabeza el arte de escribir un paper. He tenido mucha suerte con vosotros.

Gracias a Luca que singlehandedly ha hecho que me aplique con la promesa de invitarme a Botín si sacaba más de un nueve de media. Gracias a la Eigencrew, a Martín y Elías, por hacer todo más divertido. Gracias, en general, a la gente, porque sin gente no merecería la pena. Gracias a mis padres por acogerme de nuevo en casa para sacarme el máster. Gracias a la neurona que decidió que me sacara un máster porque sin ella estaría pudriéndome en un trabajo que no me gusta.

Math is just a game we're all playing together—it's better with more people, and best when we are all friends.

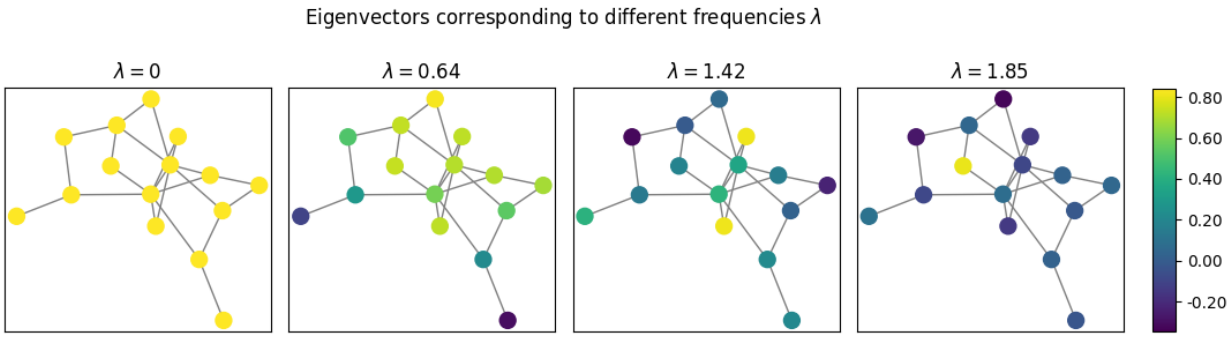
Chapter 1

Graph signal processing: state of the art

1 Motivation

1.1 From classical to graph signal processing

Graph signal processing (GSP) has emerged as a field that uses classical signal processing principles to analyze signals defined in graphs. It has been classified in [32], together with matrix factorization-based methods, random walk-based methods, and deep learning methods, as one of the four main approaches to a wider subject called graph learning. Fundamental operations of classical signal processing such as filtering, translation, modulation, dilation, and downsampling were starting to be applied to signals over graphs around the early 2010s [28]. This was achieved by extending the notion of frequency of a time signal to frequencies of signals over graphs [18]. A signal over a graph is a real-valued function over the set of nodes. For instance, in the case of climate sensors deployed across a geographical area, the signal could represent temperature, humidity, or wind speed measurements. If we manage to define an operator S acting on these signals, then we define their frequencies λ as the eigenvalues of the operator $Sv = \lambda v$ [27]. This significant insight has filled a void in the study of network signals, unlocking the potential to apply powerful existing techniques and enabling more comprehensive investigations into diverse phenomena observed in network structures.



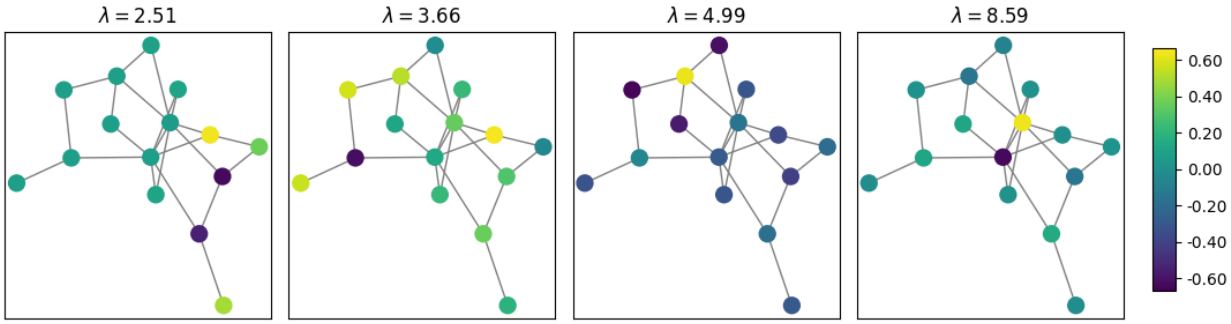


Figure 1: Different eigenvectors associated with different graph frequencies of the graph Laplacian operator L associated with a Barabási–Albert graph. The colorbar represents the value of the eigenvector in each node. Any signal can be decomposed as a linear combination of these eigenvectors, this decomposition is called the graph Fourier transform. Note that the first eigenvector of the Laplacian is always constant, corresponding to the steady state of the heat equation.

GSP does not limit itself to signals defined over preexisting graphs. It might unveil a graph structure from signals in a context where a graph was not to be expected. As an example, in [8], they managed to build graphs from financial market data, where each node represents a stock and the signal represents its value. Using the signals and different models they manage to build the weights that represent how all stocks interact together. Then, with market historical data, they create a sequence of graphs that capture the market's state at each point in time (see Figure 2). Together with a clustering technique, they use this sequence to show how the market division onto economic sectors can change over time. In addition to that, they show that the sequence of graphs can also be used to detect anomalies in the market. Using the second eigenvalue of the graph Laplacians—a linear operator acting on the graph signals—they manage to build an enter-exit market strategy.

The second smallest eigenvalue of the graph Laplacian measures the connectivity of the graph [7]. When it reaches 0, we know that the graph has been divided into two disconnected

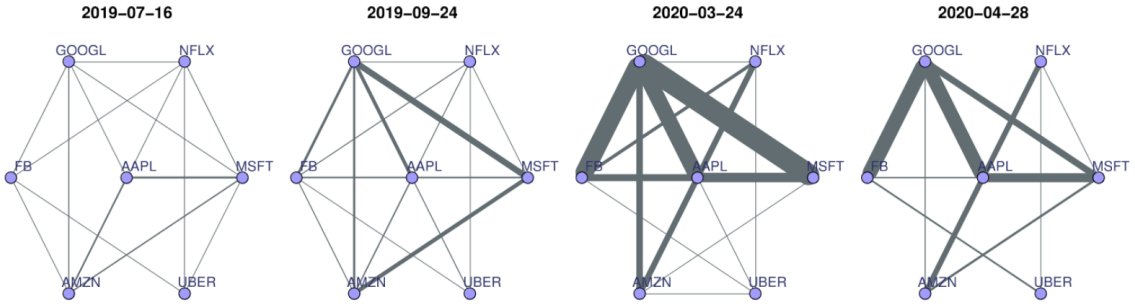


Figure 2: The nodes represent stock symbols (Facebook, Apple, Amazon, Microsoft, Uber, Netflix, Google). Using the stock prize as a signal, a graph is learned at each timestamp. Connectivity—the second eigenvalue—starts to go down, and in March 2020, crosses a threshold. Figure from [8].

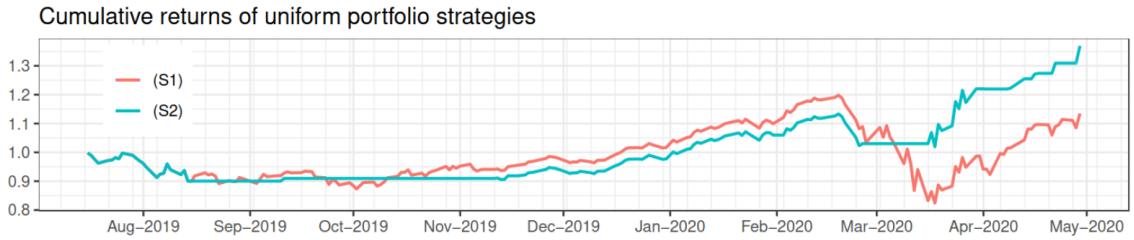


Figure 3: Value of uniform investment onto FAAMUNG companies. (S2), an enter-exit strategy based on the algebraic connectivity of the financial graph, outperforms the continuous investment (S1) and is able to exit before the crash. See how they exit the market two times in August 2019 and March 2020. Figure from [8].

subgraphs, which, heuristically, models a market crash. Building upon that idea, they create an exit-enter strategy in which they withdraw all the money from the market whenever the second eigenvalue goes below a threshold τ and enter again when it goes back up. Then, they simulate the cumulative returns of a uniform portfolio which uses this strategy against one that doesn't (see Figure 3). The exit-enter strategy was able to recognize the economic crash caused by COVID only from the stock data and sell soon enough, outperforming the no-exit clause investment.

GSP for mathematical finance is being actively studied at the moment. For a state-of-the-art review containing finance, we suggest [18]. In other reviews, mainly [10], [20], they opt to cover more scientific topics. According to [20], the hottest GSP topics right now are *sensor networks*, *biological networks*, *image processing*, and *machine learning*.

We traced back our own exploration of the topic—concerning the heat equation over graphs—to an image processing paper from 2008 [34]. The main idea of the paper is to understand images not as functions over a rectangle $[0, a] \times [0, b]$, but as signals over a lattice graph defined over the pixels. Then, using the exponential of the graph Laplacian, they formulate the heat equation over the graph to diffuse the signal through a short period of time and thus perform denoising, see Figure 4. To avoid reducing the sharpness of gradients, the weight between adjacent pixels is reduced according to the difference in value of the image signal. At the present time, there is extensive literature covering image denoising through the graph Laplacian, (see [21], [17], [33]). There has also been an extension of preexisting ideas in image processing to unstructured meshes of 2D points, or 3D cloud points. At the same time, old theoretical ideas from spectral graph theory were used to reformulate some image compression techniques (such as the discrete Fourier transform and the discrete cosine transform) within the new GSP framework [18].

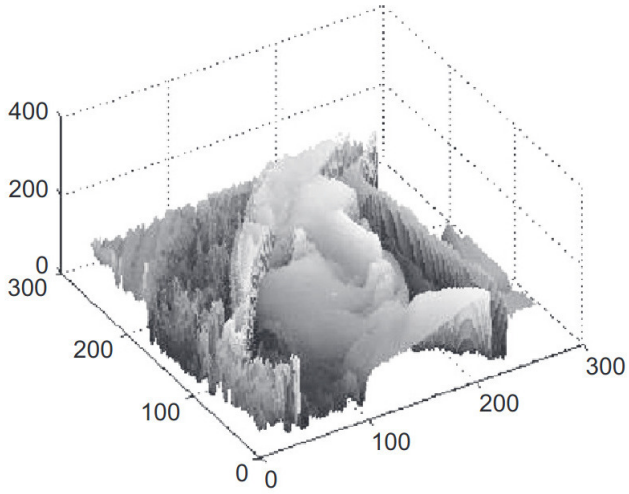


Figure 4: A grey-scale image seen as a signal over the nodes of a lattice graph. We can consider the image as an initial condition to the graph heat equation. We manipulate the weights of the graph to avoid blurring contours and let it diffuse for a brief Δt. Figure found in [34].

Some other new advancements are related to biological networks and the brain. As an example, in [14], an experiment was made in which participants had to learn a motor skill while being MRI scanned. By using a technique found in [30] and statistical analysis, they managed to build a graph for each participant's brain and, by decomposing the signals collected onto their graph frequencies, they found that signals with strong high and low-frequency components were overlapping with the already known regions corresponding to motor learning. As described in [14], before GSP, the way to decompose brain signals was through PCA analysis, which was losing all but a few low-frequency components.

Finally, the most straightforward and simple application of GSP is found in sensor networks. As it was mentioned, a sensor network is a collection of sensors spread out in physical space which measure a quantity, the signal. To build a graph from the nodes, we usually take edges as a decreasing function of the geographical distance between sensors, it might be Euclidean distance, geodesical distance, or difference in geographical elevation [31]. GSP has managed to study these signals without resorting to black box models. We might be interested in not running the whole sensor network all the time, so we might need to reconstruct a signal from a subset of sensor readings, or perhaps we want to deprecate abnormal readings or outliers, for which we need to redesign classical filters to work with signals. These tasks have been thoroughly studied in time and image signals and, thanks to GSP insights, they have been generalized to these cases [20], [36].

When geographical distances are not enough to build the graph, when, for instance, there are geographical obstacles, such as a mountain, or, for unknown reasons, there is simply no connection, then the graph needs to be inferred—learned—from the signals. This will be the main focus of the thesis, algorithms that somehow decide which nodes are connected and with what strength. There is a great effort put onto this subject. All these papers, [9], [15], [27], [31], [35], contain models that try to explain how the creation of the signals relates to the structure of the graph and how can we retrieve the graph from them. In [31], they use their

diffusion-based model to explain the signals and apply the algorithm, called LearnHeat, to the European Tracer Experiment (ETEX) [19], an experiment made in 1993 in which a certain gas, the tracer, was injected into the atmospheric system, measuring its evolution through time during 72 hours at 168 stations along Europe. In Figure 5, we can see the learned connections between sensors that capture the atmospheric dynamics and the geographical barriers between them.

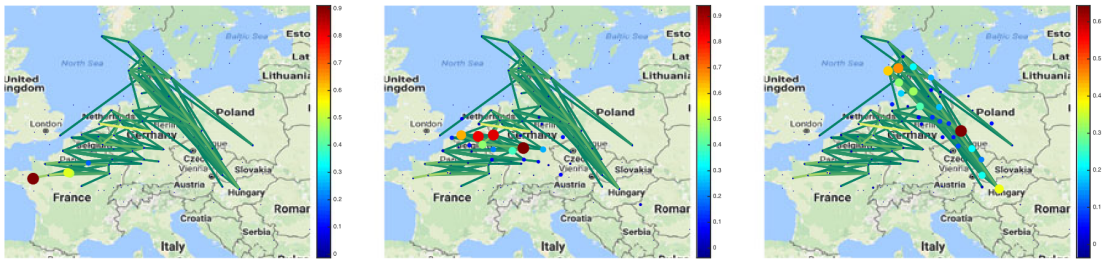


Figure 5: Learned graph and timestamps of the concentration as colored points. To the left, initial condition, gas was injected in Rennes, France, and spreads in the atmosphere from left to right. Green lines depict the heaviest weights of the learned graph. Colored dots are representative measurements of the tracer. Figure found in [31].

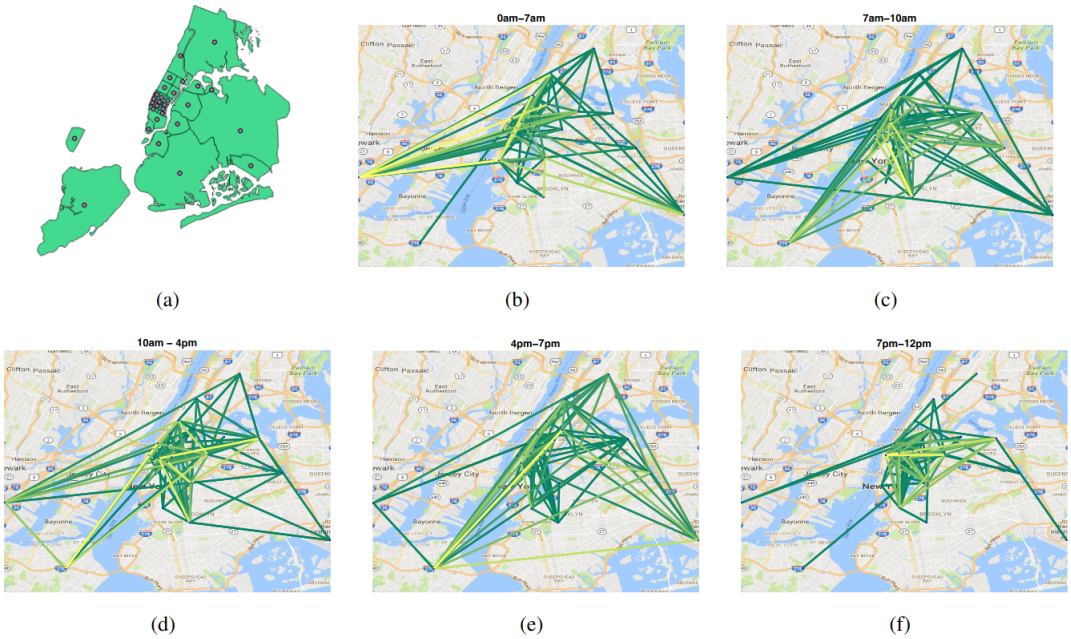


Figure 6: Subfigure (a) is a geographical division of New York City onto $N = 29$ zones. Each zone is a node, signals are the number of pickups at each zone over a time period. At each time period, a graph is learned: (b) 0:00 - 7:00 am, (c) 7:00 - 10:00 am, (d) 10:00 - 4:00 pm, (e) 4:00 - 7:00 pm, and (f) 7:00 pm - 0:00 am. In all subfigures, only the $4N$ most weighted edges. Figure found in [31].

Figure 6 corresponds to a similar experiment as Figure 5 made also in [31]. Assuming a

diffusion process driving the mobility of people throughout New York City and dividing the day into 5 time periods, they use the data from a month's worth of Uber pickups as signals to learn the graph driving the diffusion process, finding mobility patterns for each period of the day by only looking at the Uber pickup. This last model and these two experiments caught our attention, and we decided to explore them in more detail in this thesis.

1.2 Scope of the thesis

We were particularly interested in the diffusion-based model [31]. On the theoretical side, the model has a strong physical interpretation, which aligns with our academic interests. As a more ambitious, close-to-home reason, we thought it would be of great interest to recreate the New York Uber experiment with data from Madrid's underground system. The framework is the same: in Madrid, the ticket is only checked at the entrance, just as in the Uber dataset only the pickup is recorded. So we had in mind to gather entrance data and study the mobility trends in our city.

According to [31], one of the main bottlenecks which were preventing the algorithm to be applied to big networks was that both the computation of the exponential and gradient of the Laplacian—both $N \times N$ symmetric matrices—require $\mathcal{O}(N^3)$ operations via spectral decomposition or scaling and squaring method [13]. In our example, without properly distributing the algorithm, we would need around 2 days to properly learn a mobility trend network in a standard CPU, as the Metro de Madrid network has, at the moment, 328 nodes. This makes the algorithm inapplicable to a lot of interesting networks—social, and biological—which are typically formed by thousands of nodes.

We were committed to distributing the algorithm to be run efficiently in parallel high-performance supercomputers by using Monte Carlo techniques to compute the matrix exponential times vector as in [1], [2]. However, our idea encountered new problems that were challenging its feasibility. The gradient descent performed in the algorithm required an eigendecomposition of the exponential, which couldn't be achieved solely through matrix-vector multiplication by the exponential. Moreover, there was a Lipschitz constant depending on the graph Laplacian which needed to be computed to ensure convergence. We didn't manage to give a closed form to this constant and a backtracking algorithm was required. Thus, any effort made at computing efficiently the exponential and the gradient had now to deal with the number of times we would repeat the computation at each step. When we finally managed to simulate signals and apply the algorithm, we found we were facing an even bigger problem which, in hindsight, was to be expected from the beginning: almost all graphs being learned were fully connected.

We realized that in [31], the results were obtained using the number of edges in the groundtruth graph to filter the edges in the learned graph, so they were, for the moment, avoiding the problem. Even the presented Figures 5 and 6 used an arbitrary cutoff to delete the edges. Given that we did not find any mention to this problem in the existing literature and the papers citing [31], we decided to address the problem ourselves and try to give a solution.

We use Chapter 1, Section 2, to cover the math needed to understand the algorithm and our contributions, as well as place the heat diffusion model in context within the framework of GSP learning models. In Section 3 we explain through examples the clear weaknesses of the algorithm.

In Chapter 2, Section 1, we provide two heuristic algorithms which attempt to give a meaningful threshold to the learned graphs. These are based on solid mathematical ideas: moving windows and persistent homology. Our results demonstrate that these algorithms provide a favorable solution, particularly when considering the lack of existing solutions documented in the current literature. In Chapter 2, Section 2, we address possible criticism of our approach and justify the decisions made throughout the experiments. We show why getting close to the original topology of the graph is not detrimental but beneficial to the predictive capabilities of the learned graph and, in the second subsection, we explore how topological priors can be loaded onto the algorithm, which goes along our original motivation: to apply the algorithm to the underground system, where we know the topology but not the weights.

2 Mathematical background

In this section, we briefly go through the mathematics of GSP and the LearnHeat algorithm. To dive deep into technical details we recommend, in an applied fashion, [15] and [27], and for a more theoretical understanding we refer to [7].

2.1 Spectral graph theory preliminaries

Definition 1 (Undirected simple finite graph) *An **undirected simple finite** graph G is a pair (V, E) where V is a **finite** set of vertices (also called nodes) with cardinality n and E is a subset of **unordered pairs** of vertices $E \subset UConf_2(V)$[1]. We can add real weights $w_{ij} > 0$ to each edges (v_i, v_j) to form a **weighted graph** (V, E, W).*

To study graphs in an algebraic manner, we use the adjacency matrix.

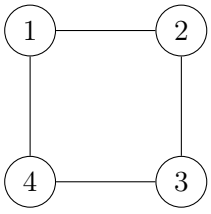
Definition 2 (Adjacency matrix) *Given an ordering of the set of vertices $V = \{v_1, \dots, v_n\}$ the adjacency matrix A of a graph G is defined as*

$$A_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \in E \\ 0 & \text{otherwise} \end{cases}$$

*If the graph is weighted we consider its **weighted** adjacency matrix W*

$$W_{ij} = \begin{cases} w_{ij} & \text{if } \{v_i, v_j\} \in E \\ 0 & \text{otherwise} \end{cases}$$

Note that these matrices are real symmetric, therefore we can apply the spectral theorem. In this thesis, we consider weighted adjacency matrices $W \in \mathcal{W}$, where $\mathcal{W}$ is the set of feasible (weighted)[2] adjacency matrices. Whenever we have a graph G we can construct its adjacency matrix W, and whenever we have a valid $W \in \mathcal{W}$, we can construct a graph whose adjacency matrix is W.



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

Figure 7: Graph and adjacency matrix of an unweighted C_4 (cyclic) graph

Definition 3 (Walk on a graph) *A walk on a graph is a sequence of edges $\{e_i\}_{i \in I}$ joining a sequence of vertices $\{v_j\}_{j \in J}$.*

[1] Proper topological notation.
[2] We omit the weighted word.

Lemma 1 *The ij-th entry of the k-th power of the (weighted) adjacency matrix $(W^k)_{ij}$ of G contains the number of walks of length k between v_i and v_j on G.*

Definition 4 (Degree matrix) *Given an undirected simple graph G the degree matrix D is defined as*

$$D_{ij} = \begin{cases} \delta(v_i) & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

where $\delta(v_i)$ is the degree of the vertex v_i, the cardinality of the set of edges containing v_i.

Definition 5 (Weighted combinatorial Laplacian matrix) *Given a weighted undirected simple graph G the combinatorial Laplacian matrix L is defined as*

$$(L_c)_{ij} = \begin{cases} \delta(v_i) & \text{if } i = j \\ -w_{ij} & \text{if } \{v_i, v_j\} \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

which is a positive semidefinite matrix.

Lemma 2 *For any graph G*

$$(L_c) = D - W$$

where D is the degree matrix and W is the adjacency matrix.

In our previous example of a C_4 graph, the identity would read as

$$L_c = \begin{bmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix} = D - W = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Definition 6 (Weighted standard Laplacian matrix) *Given a connected graph G with weights W, the weighted standard Laplacian is defined as*

$$L_{ij} = \begin{cases} 1 & \text{if } i = j \\ \frac{-w_{ij}}{\sqrt{\delta(v_i)\delta(v_j)}} & \text{if } \{v_i, v_j\} \in E \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

which is a positive semidefinite matrix. If the graph G has $\{G_k\}$ connected components we define L_G as the direct sum of Laplacians $\bigoplus_k L_{G_k}$.

Lemma 3 *If G is connected then the identity*

$$L = D^{-1/2} L_c D^{-1/2}$$

holds

Remark. In graph learning problems, the Laplacian used is implicit in the feasible set that is given for the optimization problem.

Lemma 4 *Given a graph G with k connected components and its weighted standard (and combinatorial) Laplacian matrix L then the 0-th eigenvalue has exactly multiplicity k*

$$\sigma(L) = \{\lambda_1 = \lambda_2 = \cdots = \lambda_k = 0 < \lambda_{k+1} \leq \cdots \leq \lambda_n\}$$

and $\operatorname{tr} L = n - k + 1$. *In standard Laplacians, $\lambda_n \leq 2$, in combinatorial Laplacians, $\lambda_n \leq 2\rho$, where ρ is the maximal degree of the graph [7].*

Definition 7 (Signal over a graph) *Given a graph $G = (V, E, W)$, a signal x is a square integrable function defined at each node $x \in \ell^2(V)$. This formulation is used to generalize to infinite graphs, in the finite real-valued context, in a graph of n nodes, a signal is just a vector $x \in \mathbb{R}^n$.*

Remark. This definition of signal leads to a formulation of the Laplacian as a self-adjoint operator acting on $\ell^2(V)$. In the finite context, we recover the symmetric matrix formulation. A matrix $L \in \mathbb{R}^{n\times n}$ acts on a vector $x \in \mathbb{R}^n$ by matrix-vector multiplication Lx.

Definition 8 (Heat kernel of L) *Given a graph $G = (V, E, W)$ and its Laplacian $L = D - W$ acting on $\ell^2(V)$, the heat kernel H_t is the solution to the system of ODEs*

$$\frac{\partial H_t}{\partial t} = -LH_t \tag{3}$$

which, for each t, is a symmetric positive definite matrix acting over—diffusing—the set of initial conditions $\ell^2(V)$.

Equation (3) is the analog in graphs to the heat equation over Riemannian manifolds or, in particular, Euclidean space. The heat kernel H_t contains information concerning the path length distribution on the graph. In combinatorial Laplacians, the entry $(L^k)_{ij}$ contains the number of paths from v_i to v_j of length k, just as in Markov chains [7].

Lemma 5 *The solution to (3) is given by*

$$H_t = e^{-tL} = \mathbf{I} - tL + \frac{t^2}{2!}L^2 - \frac{t^3}{3!}L^3 \dots$$

As

$$L = \chi\Lambda\chi^T$$

that enables us to rewrite the heat kernel as

$$H_t = \chi e^{-t\Lambda}\chi^T$$

Graph-shift operators and graph filters

GSP revolves around finding matrix representations of the graph, the so-called *graph-shift operators*. The following math concerning graph-shift operators might be of interest to the reader, as it is contained in the state-of-the-art GSP problem formulation.

Definition 9 (Graph-shift operator (GSO)) *A graph-shift operator (GSO) over a graph G is defined as any real matrix $S \in \mathbb{R}^{N\times N}$ that captures the sparsity pattern of the graph [25]. That means that the ij-th entry of the matrix will be 0 if and only if there is no edge connecting vertex v_i to v_j*[3].

Lemma 6 *For undirected graphs, the GSO is always a square real symmetric matrix, therefore it admits an orthogonal eigendecomposition which we will denote as*

$$S = \chi\Lambda\chi^T$$

Examples of commonly used GSO are the adjacency matrix W and any kind of Laplacian L_c, L.

Definition 10 (Graph Fourier transform (GFT)) *Given a GSO S acting over signals $\ell^2(V)$ the graph Fourier transform is defined as $F = \chi^{-1} = \chi^T$ acting over $x \in \ell^2(V)$.*

GSP interpretation of the heat kernel. If $S = L = \chi\Lambda\chi^T$, given a signal $x \in \mathbb{R}^n$, we can apply the GFT to the signal $\chi^T x$. Each entry $(\chi^T x)_i$ is the coefficient of the signal x written in the basis of eigenvectors corresponding to graph frequencies λ_i. Applying the heat kernel corresponds to multiplying each entry by $e^{-t\lambda_i}$. All frequencies $\lambda_i > 0$ will make the coefficient of its eigenvector go to zero exponentially, and $\lambda_i = 0$, $e^{-t\lambda_i} = 1 \forall t$, each for each connected component of the graph, will eventually dominate, which corresponds to the steady state: the average of x, $\bar{x}$, over the connected component.

[3] This doesn't apply to the diagonal S_{ii}.

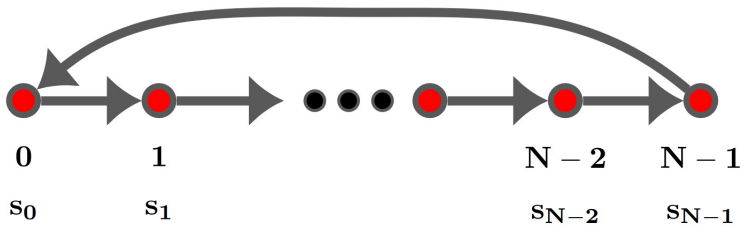


Figure 8: The eigenvector matrix of the **directed**—we use undirected graphs in this thesis—cyclic graph's adjacency matrix is just the well known DFT_n matrix. This is one of multiple examples of mathematical concepts getting simpler when generalized. (Figure from [20]).

Definition 11 (Graph filter) *Given a GSO S, a filter is $H = h(S)$ where h is any matrix function in the sense of [13], then the filter acts over signals as*

$$y = h(S)x = \sum_i \chi_i h(\lambda_i)\chi_i^T x$$

this is the definition found in [15], other restrictive definitions might be found in the literature.

Remark. For a fixed t, the filter $h_t(x) = e^{-tx}$ corresponds to the heat kernel. Filters are seen to be functions that selectively amplify frequencies in the Fourier space of the graph while removing or reducing the non-important ones [32], in the case of the heat kernel, it exponentially reduces all frequencies that do not correspond to steady states.

Definition 12 (Stationary signal) *If we are given a white noise signal w with $\mathbb{E}[w] = 0$ and $\mathbb{E}[ww^T] = \mathbf{I}$, then we say that x is stationary with respect to S if there exist real coefficients $\{\alpha_i\}_{i=0}^{\infty}$ such that the following identity holds [27]*

$$x = \alpha_0 \prod_{i=1}^{\infty}(\mathbf{I} - \alpha_i S)w = \sum_{i=0}^{\infty} \beta_i S^i w \tag{4}$$

Lemma 7 *[27] Given a set of stationary signals $X \in \mathbb{R}^{n\times m}$ with respect to a GSO $S = \chi\Lambda\chi^T$ then*

$$\mathbb{E}\left(\frac{1}{n}XX^T\right) = \chi\hat{\Lambda}\chi^T$$

This implies that we can approximate the eigenvectors of the GSO with enough stationary signals by performing an eigendecomposition of the covariance matrix.

2.2 State-of-the-art graph learning models

All the GSP-based graph learning papers tend to adopt the following structure:

- Propose a mathematical model for the signals using the GSO.
- Given the signals, formulate an optimization problem whose solution is the GSO.
- Create synthetic signals, solve the optimization problem, and evaluate the results against other graph learning models.

The most commonly used GSO is the standard Laplacian, as its 2-norm is uniformly bounded for all n (see Lemma 4). The all-ones-diagonal condition is relaxed in all practical applications to $\mathrm{tr}(L) = n$, which is computationally more manageable. The following is one of the first GSP-based learning algorithms [9]. It is based on a global smoothness assumption. The heat diffusion model will be a case in which global smoothness is substituted by local piecewise smoothness.

Optimization problem 1 (Dong's smooth learning model) *Given a set of signals* $X \in \mathbb{R}^{N\times M}$*, alternate between solving this two optimization problems. In the first step solve*

$$\begin{aligned} \min_{L} \quad & \alpha \operatorname{tr}(Y^T L Y) + \beta \|L\|_F^2 \\ s.t. \quad & (\forall i \neq j),\ L_{ij} = L_{ji} \leq 0, L_{ii} = -\sum_{i\neq j} L_{ij}, \quad \operatorname{tr}(L) = n \end{aligned}$$

then, in the second step, solve

$$\min_{Y} \|X - Y\|_F^2 + \alpha \operatorname{tr}(Y^T L Y)$$

This algorithm is the precursor of plenty of now state-of-the-art algorithms in the field. It is minimizing some quadratic form over a denoised version of the original signal X. That quadratic form measures the total variation of the signal along the graph

$$f^T L f = \frac{1}{2} \sum_{i\sim j} w_{ij} (f(i) - f(j))^2$$

and by minimizing it we make the observed signals smooth under the learned graph ($i \sim j$ means $(v_i, v_j) \in E$ and $\frac{1}{2}$ avoids summing twice the same edge.) Problem 1 is non-jointly convex. This is ubiquitous in all GSP algorithms, they are mathematically complex and computationally intensive but, as a trade-off, they are resilient to noise. In [9] they measure their results against the statistical-based—not GSP-based—graph learning model described in [16]. As the GSP model manages to perform in equal terms with the other well-established model [9], the paper catches the graph learning community's attention and soon, a second paper based on smooth signals [15] is written, proposing the following change in formulation:

Optimization problem 2 (Kalofolias smooth learning model) *Given a set of signals* $X \in \mathbb{R}^{N\times M}$ *with* $X = [x_1, x_2, \ldots, x_N]$ *and it's pairwise distance matrix* Z *defined by*

$$Z_{ij} = \|x_i - x_j\|^2$$

solve the following optimization problem

$$\begin{aligned} \min_{W\in\mathcal{W}} \quad & \|W \circ Z\|_{1,1} + \alpha \|W\mathbf{1}\|_2^2 + \alpha \|W\|_F^2 \\ s.t. \quad & \|W\|_{1,1} = n \end{aligned}$$

where $\|W\|_{1,1} = \sum_{i,j} |w_{ij}|$, $\circ$ *stands for the Hadamard product, and* $\mathcal{W}$ *is the set of feasible weighted undirected adjacency matrices.*

They claim to solve Dong's algorithm's two main weaknesses: lack of interpretability of the regularizers and difficult numerical interpretation. Problems 1 and 2 are the two commanding state-of-the-art smooth signal models at the present date. In the late 2010s and early 2020s, there has been a great effort to further develop new models that do not pursue smoothness, but rather some physical explanation to the signals, check review [10], written by, again, three out of four authors of [31]. We will focus now on spectral filtering models, which are the ones concerning this thesis, [27], [31], [35] and in particular, the heat diffusion model [31].

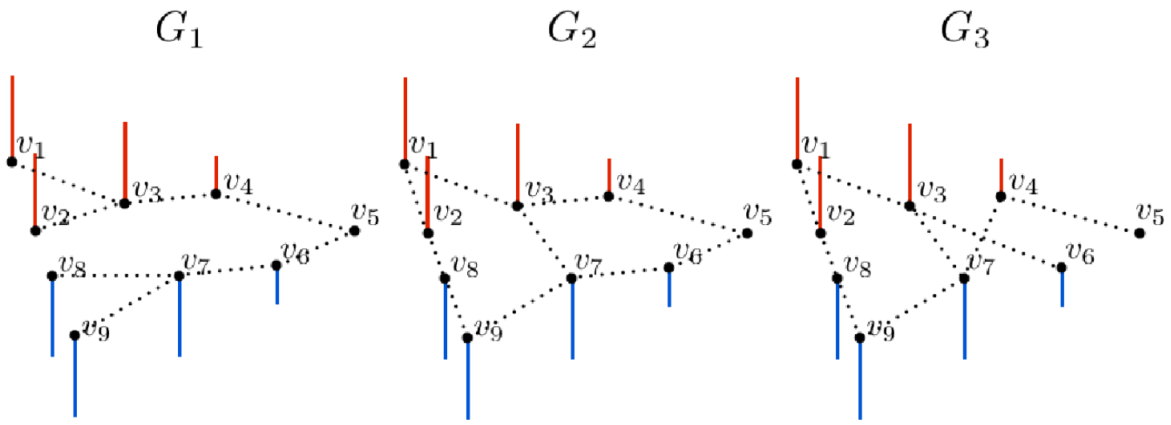


Figure 9: Three possible Laplacians for a given signal. G_1 minimizes the total variation of the signal along the edges, therefore the signal is smoother in the G_1 topology than in the rest of the graphs. Figure from [9].

2.3 The heat diffusion dictionary

To model a signal, the authors in [31] consider the factor analysis model [3], related to what is known as dictionary learning [35]. Its objective is to explain a signal from a potentially smaller number of unobserved latent variables. They consider the following model

$$x = \mathcal{D}h + u_x + \epsilon \tag{5}$$

where $x \in \mathbb{R}^N$ is the observed graph signal, $h \in \mathbb{R}^K$ is the latent variable that attempts to explain the signal and $\mathcal{D} \in \mathbb{R}^{N \times K}, N \leq K$ is a dictionary. The vector $u_x \in \mathbb{R}^N$ is the mean of x, which can be set to 0 and then be added up after the learning process, and ϵ is Gaussian noise coming from a $\mathcal{N}(0, \sigma_\epsilon^2 \mathbf{I})$. Recall that we call these signals stationary and algorithms such as [27] can be used. Columns of the dictionary $\mathcal{D}$ are called atoms. We usually take dictionaries of the form

$$\mathcal{D} = [h_1(L)\ h_2(L)\ \ldots\ h_S(L)]$$

where $\{h_s(\cdot)\}_s$ are graph filters that commute with the shift, in this case the Laplacian L. In the paper, they impose a Laplace (sparse) prior on the latent h variable

$$p(h) = \prod_i \alpha \exp\left(-\alpha |h(i)|\right)$$

where α will be a tuning parameter. Then taking $u_x = 0$ without loss of generality we obtain

$$p(x|h) \sim \mathcal{N}(\mathcal{D}h, \sigma_\epsilon^2 \mathbf{I})$$

So we can sketch our first spectral-based graph learning problem.

Optimization problem 3 (Maximum a posteriori estimate of h) *Given a signal $x \in \mathbb{R}^N$, the maximum a posteriori estimate of h is given by*

$$\begin{aligned} h_{MAP}(x) &= \arg\max_h p(h|x) = \arg\max_h p(x|h)p(h) \\ &= \arg\min_h(-\log p_E(x - \mathcal{D}h) - \log p_H(h)) \\ &= \arg\min_h \|x - \mathcal{D}h\|_2^2 + \alpha\|h\|_1 \end{aligned}$$

Optimization problem (3) will be twisted outside a statistical viewpoint, so stationary signals will only be produced to compare results with [27]. When we have multiple signals, the MAP estimate then becomes

$$\arg\min_H \ \|X - \mathcal{D}H\|_F^2 + \alpha \sum_{m=1}^{M} \|h_m\|_1 \tag{6}$$

for $H = [h_1, h_2, \ldots, h_M]$. The left part of the loss function, which attempts to undo the generation of the signal

$$X = \mathcal{D}H$$

is overdetermined, so under full-rank assumptions it will have an infinite subspace of solutions, therefore we regularize through

$$\alpha \sum_{m=1}^{M} \|h_m\|_1 \qquad \text{(sometimes written as } \alpha\|H\|_{1,1}\text{)}$$

in an attempt to impose sparsity and reduce the dimensionality of the set of solutions. To define a dictionary $\mathcal{D}$ we will use the heat kernel.

Definition 13 (Diffusion dictionary) *Given a vector $\tau = [\tau_1, \tau_2, \ldots, \tau_S] \in \mathbb{R}^S$ of diffusion rates and a graph Laplacian L the diffusion dictionary $\mathcal{D} \in \mathbb{R}^{N \times NS}$ is defined as the horizontal concatenation of heat kernels associated to each τ_s evaluated at time $t = 1$*

$$\mathcal{D} = [e^{-\tau_1 L}, e^{-\tau_2 L}, \ldots, e^{-\tau_S L}] \tag{7}$$

This means that we have a prior belief that there are S independent diffusion processes driving the signals through different timescales. Keep in mind that X is not a time series, but a snapshot of the diffusion process, there is no notion of time contained in the signal, it is contained in the dictionary as S different timescale options: the algorithm will be in charge of learning which.

Optimization problem 4 (Heat diffusion model) *Given a set of signals $X \in \mathbb{R}^{N \times M}$, solve*

$$\underset{L,H,\tau}{\text{minimize}} \quad \|X - \mathcal{D}H\|_F^2 + \alpha \sum_{m=1}^{M} \|h_m\|_1 + \beta\|L\|_F^2 + \delta(L|\mathcal{C})$$

where $\delta(L|\mathcal{C})$ is the indicator function for the convex set $\mathcal{C} = \{\text{tr}(L) = N, L_{ij} = L_{ji} \leq 0, i \neq j, L\mathbf{1} = \mathbf{0}\}$, defined as

$$\delta(L|\mathcal{C}) = \begin{cases} 1, & \text{if } L \in \mathcal{C} \\ +\infty, & \text{otherwise} \end{cases}$$

and the dictionary $\mathcal{D} \in \mathbb{R}^{N \times NS}$ *is given by* S *different heat kernels* $[e^{-\tau_1 L}, e^{-\tau_2 L}, \ldots, e^{-\tau_S L}]$.

Lemma 8 *The objective function in optimization problem* (4) *is nonsmooth and nonconvex with respect to* L, H, *and* τ *simultaneously, therefore the problem will almost always have many local minima [31].*

The trace constraint serves as a regulator to the volume (the sum of degrees) of the graph and emulates, somehow, that the trace of the standard Laplacian would also be N. This prevents the Laplacian from blowing up. The $\beta\|L\|_F^2$ term is meant to control the values of the off-diagonal entries of the Laplacian, even if they are also controlled by the trace and eigenvector constraints.

We want to state clearly that the groundtruth L, H, τ are not the minima of their own optimization problems due to the regularizers. Nevertheless, it has been shown experimentally that we need them in order to obtain good results. As we said earlier, the β-regularizer has found its critique in [15] but we will still use it to be faithful to the paper's results.

2.4 Solving the heat learning problem with the PALM algorithm

For a detailed introduction to the PALM algorithm, we refer to [5].

Let's rewrite the heat learning objective function in the form of the PALM algorithm. Consider the following three functions. The smooth function involving all terms

$$Z(L, H, \tau) = \|X - \mathcal{D}H\|_F^2$$

and the regularizers for each term

$$f(H) = \sum_{m=1}^{M} \|h_m\|_1, \quad g(L) = \|L\|_F^2 + \delta(L|\mathcal{C})$$

Then we can rewrite the objective function as

$$\underset{L,H,\tau}{\text{minimize}} \quad \alpha f(H) + \beta g(L) + Z(L, H, \tau)$$

The PALM algorithm consists in, at each step, **alternate** around optimizing H, L, and τ. At each substep, say the L-step in the t-th iteration, we will **linearize** the function through a first-order Taylor expansion

$$Z(L^t + \Delta L, H^{t+1}, \tau^t) = Z(L^t, H^{t+1}, \tau^t) + \langle \Delta L, \nabla_L Z(L^t, H^{t+1}, \tau^t) \rangle + \mathcal{O}((\Delta L)^2)$$

and input the linearized version in the direction of steepest descent (multiplied by $d_t = \gamma_2 C_2(H^{t+1}, \tau^t)$ where $C_2(H^{t+1}, \tau^t)$ is a global Lipschitz constant of the gradient) onto the **proximal** map given the regularizer $g(L)$.

$$L^{t+1} = \text{prox}_{d_t}^g \left(L^t - \frac{1}{d_t} \nabla_L Z(L^t, H^{t+1}, \tau^t) \right) \tag{8}$$

To read a general definition of the proximal operator, go to the references [5]. In this particular case, L^{t+1} is the solution to the following quadratic program

$$\arg\min_{L} \langle L - L^t, \nabla_L Z(L^t, H^{t+1}, \tau^t)\rangle + \frac{d_t}{2}\|L - L^t\|_F^2 + \beta\|L\|_F^2 + \delta(L|\mathcal{C}) \tag{9}$$

which we believe can be solved through the vectorization $\text{vec}(L) \in \mathbb{R}^{n^2}$ and half-vectorization $\text{vech}(L) \in \mathbb{R}^{\frac{n(n-1)}{2}}$ related through the duplication matrix

$$\mathcal{M}_{dup}\text{vech}(L) = \text{vec}(L)$$

and rewriting all Frobenius norms

$$\|L\|_F^2 = \text{tr}(LL^t) = \text{vec}(L)^T\text{vec}(L) = \text{vech}(L)^T\mathcal{M}_{dup}^T\mathcal{M}_{dup}\text{vech}(L)$$

which makes the Frobenius norm a quadratic form over the half-vectorized matrix. This has been done in [9], rewriting as well the matrix constraint onto vectorized form, in this last paper [31] the optimization problem has been solved with the SCS library[4], which uses the ADMM (Alternating Direction Method of Multipliers) method together with cone programming. Dealing with all the nuances of the method is not our main focus, so we used the Python implementation `cvxpy`. Here is the algorithm greatly simplified.

Algorithm 1 Learning heat kernel graphs **(LearnHeat)**

Input: Signal set X, number of iterations `iter`, gamma vector γ, regularization parameters α and β, precondition L^0, τ^0, H^0 (optional)
Output: Sparse signal representations H, graph Laplacian L, diffusion parameter τ
Initialization: If not preconditioned, initialize $L = L^0$, $\tau = \tau^0$, $H = H^0$, $\mathcal{D}^0 = [e^{-\tau_1 L^0}, e^{-\tau_2 L^0}, \ldots, e^{-\tau_S L^0}]$

1: **for** $t = 1, 2, \ldots,$ `iter` **do**
2: Choose $c_t = \gamma_1 C_1(L^t, \tau^t)$
3: Update H^{t+1}
4: Choose $d_t = \gamma_2 C_2(H^{t+1}, \tau^t)$ by a backtracking algorithm
5: Update L^{t+1} and dictionary $\mathcal{D}$
6: Choose $e_t = \gamma_3 C_3(H^{t+1}, L^{t+1})$
7: Update τ^{t+1} and dictionary $\mathcal{D}$
8: **end for**
9: **return** $L = L^{iter}$, $H = H^{iter}$, $\tau = \tau^{iter}$

The PALM algorithm, at each substep, that is, when we go from H^t to H^{t+1} and from τ^t to τ^{t+1} requires the solution of two additional problems (in each substep). Just like with the Laplacian L, we need to solve the forward-backward update of H

$$\begin{aligned} H^{t+1} &= \arg\min_{H} \quad \langle H - H^t, \nabla Z_H(L^t, H^t, \tau^t)\rangle + \frac{c_t}{2}\|H - H^t\|^2 + f(H) \\ &= \text{prox}_{c_t}^f(H^t - \frac{1}{c_t}\nabla_H Z(L^t, H^t, \tau^t)) \end{aligned}$$

and for τ

$$\begin{aligned} \tau^{t+1} = \arg\min_{\tau} \quad & \langle \tau - \tau^t, \nabla_\tau Z(L^{t+1}, H^{t+1}, \tau^t)\rangle + \frac{e_t}{2}\|\tau - \tau^t\|^2 \\ \text{s.t.} \quad & \tau \geq 0 \end{aligned}$$

[4] `https://www.cvxgrp.org/scs/`

As there is no regularizer, this admits a closed solution of the form

$$\tau^{t+1} = \max\left(-\frac{\nabla_\tau Z(L^{t+1}, H^{t+1}, \tau^t) - e_t\tau^t}{e_t}, 0\right)$$

just as H admits also a closed solution [31] Section B. The hard part is that, at each iteration, we need to find three Lipschitz constants, $C_1(L^t, \tau^t)$, $C_2(H^{t+1}, \tau^t)$ and $C_3(H^{t+1}, L^{t+1})$ that will control the length of the gradient step so as to guarantee convergence. Their computation is long and tedious and can be found in the Appendix of [31]. There is still work to be done with $C_2(H^{t+1}, \tau^t)$ and $C_3(H^{t+1}, L^{t+1})$, their implementation right now is far from optimal, nevertheless, we will keep using their techniques. We will not spend more time covering the mathematics behind the algorithm. A public implementation at the moment is `https://github.com/semink/learnHeat`. We used this implementation as a basis for our implementation, with minor modifications.

3 Limitations found through experiments

3.1 Setting up the experiments

Let's explain rigorously the algorithm setup, after the explanation we will briefly recap with bullet points. First, we create a random graph G_{ground}. This graph will be either weighted or unweighted. The unweighted graphs are the Barabási-Albert (BA) graph, governed by the parameters m and m_0 ($m \leq m_0 \leq N$, $m, m_0 \in \mathbb{N}$), and the Erdös-Rényi graph (ER), determined by a parameter $p \in [0, 1]$. In the Barabási-Albert graph, we start from a fully connected graph K_{m_0} and then sequentially connect the rest $N - m_0$ nodes by m edges following a preferential attachment rule. These graphs are said to model scale-free networks present in nature. We will take unless said otherwise, $m = m_0 = 1$, which creates trees: acyclic connected graphs with N nodes and $N - 1$ edges. In the Erdös-Rényi graph, every node has a probability p of being connected with every other node. Finally, the weighted graph that we will use is the inverted radial basis function graph (RBF). It is created by uniformly sampling N points $\{x_i\}_{i=1}^N$ in $[0, 1]^2$ and computing the adjacency matrix by the following rule

$$W_{ij} = \begin{cases} \exp(\frac{-\|x_i - x_j\|_2^2}{2\sigma^2}), & \text{if } \|x_i - x_j\|_2 > \kappa \\ 0, & \text{otherwise} \end{cases}$$

We call it inverted because in [31] the cutoff is taken to be $\|x_i - x_j\|_2 < \kappa$. This definition enables us to include in the experiments a graph that has both a small lower and upper bound for the weights. Once we have created G_{ground}, we extract its Laplacian, L_{ground}. Using some predetermined diffusion coefficients, say $\tau = [0.5, 1, 2.5, 4]$, which can capture different time scales, we create the dictionary $\mathcal{D} = [e^{-\tau_i L_{ground}},\ \tau_i \in \tau]$ and a sparse matrix $H \in \mathbb{R}^{NS \times M}$ where M is the number of signals we want. Finally, we obtain the signals $X = \mathcal{D}H + \epsilon$ where ϵ is the Gaussian noise we want to experiment on, X will be fed onto the algorithm `LearnHeat` along parameters $\alpha, \beta, \gamma_1, \gamma_2, \gamma_3$ ($\gamma_i = 1.1$ in our experiments) and iterate `max_iter` times, usually 50. The result will be somehow post-processed, we will see why later, and the final learned graph will be compared against the graph used to generate the signals. Let's recap:

- Create a random graph from a set of parameters (number of nodes, type of graph...) and extract its Laplacian matrix L_{ground}.
- Using the Laplacian matrix and some diffusion parameters $\tau_1, \ldots, \tau_S$, create the heat dictionary $\mathcal{D}$: this will act over the initial conditions and diffuse them through the graph.
- Create random sparse initial conditions H, enough of them so that we can recover the dictionary (more on that in next subsection), and diffuse them through the graph via matrix multiplication with the dictionary $X = \mathcal{D}H + \epsilon$, where ϵ is noise (set to 0 if we don't want to test with noise).
- Feed the signals X to the algorithm together with regularizers α,β. Get $L_{learned}$ as an output.
- Delete weights of $L_{learned}$ until we get a reasonable graph $L_{threshold}$.
- Compare the edges present in $L_{threshold}$ with those in L_{ground}. Record how well we captured the groundtruth graphs topology.

The following diagram might be of help.

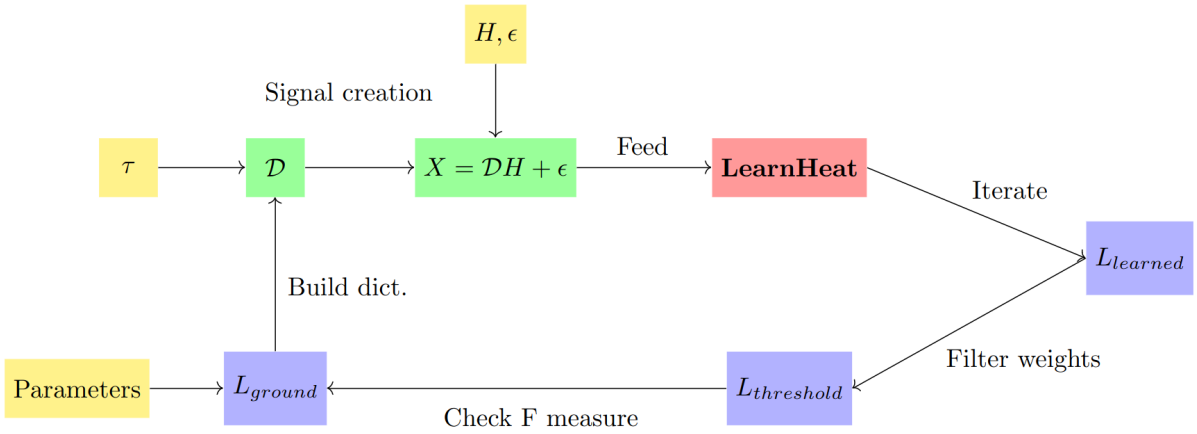


Figure 10: Schematic representation of the experiments.

The main highlight of [31] is to show that, under certain aprioristic weight filters, $N = 20$ nodes, and a considerable number of noisy signals $M = 20000$ with standard deviation $\sigma = 0.02$, the F-measure scores, which will be defined in Section 3.4, are almost perfect (see Figure 15).

Critiques to the applicability of results

We found two critiques of the applicability of this result.

The first one is that in real-life diffusion processes, there will never be 20000 signals available. To have multiple signals of a diffusion process, the diffusion process needs to be periodically reset. An example would be how people diffuse in the Metro during the day and then go back to their homes at night. Then, for each τ_s, we only get one signal each day. If for example, Metro opens at 6 a.m., and $\tau_1 = 5$, then we only get one measurement of that diffusion process at 11 a.m., and we need to wait the next day to obtain another one. Then, in a noisy framework, if we want to get similar results as in Figure 15, we need to wait 20000 days, that is 55 years, just in a Metro system of 20 stops. In a Metro system like Metro de Madrid, which has circa 300 stops, the number of signals M should be meaningfully rescaled, making the data collection last various lifetimes. Moreover, the initial conditions —where people take the Metro each morning— might not be random at all, which makes signals, in some sense, redundant, as we will see when we comment on the number of signals needed to learn a graph. We argue that the results shown in Figure 15 are caused by an overdetermination of signals which spits back unrealistic results not representative of the results we would obtain in experiments with real datasets.

The second critique is to the unjustified usage of prior knowledge when thresholding. Their results are obtained by deleting small weights in the learned graph until they are left with a thresholded graph with the same number of edges as the groundtruth graph. They can do this because they have generated the graphs to create the signals (so they know the number of edges), but, in real applications, there is no information whatsoever about the graph: the graph is hidden, and it is our task to recover the graph from the signals. Our constructive criticism of

this fact will appear in Chapter 2. We will argue that, if the prior information is really needed, then we might as well use it while running the algorithm, not after it has finished running.

3.2 Understanding the limitations through examples

Let's build the intuition to this topic from a simple example. Suppose we are in a Metro system and we don't have access to the map. We are informed that the distribution of people over the Metro system evolves in time through a graph diffusion process. We have access, only at some point in time, to the distribution of people through the Metro system, but we don't know how much time has passed since the Metro opened, which corresponds to how much time people have been diffusing, (τ), we don't know what was the distribution of people at the time it opened up (H), and we don't know which stations are connected to which (L).

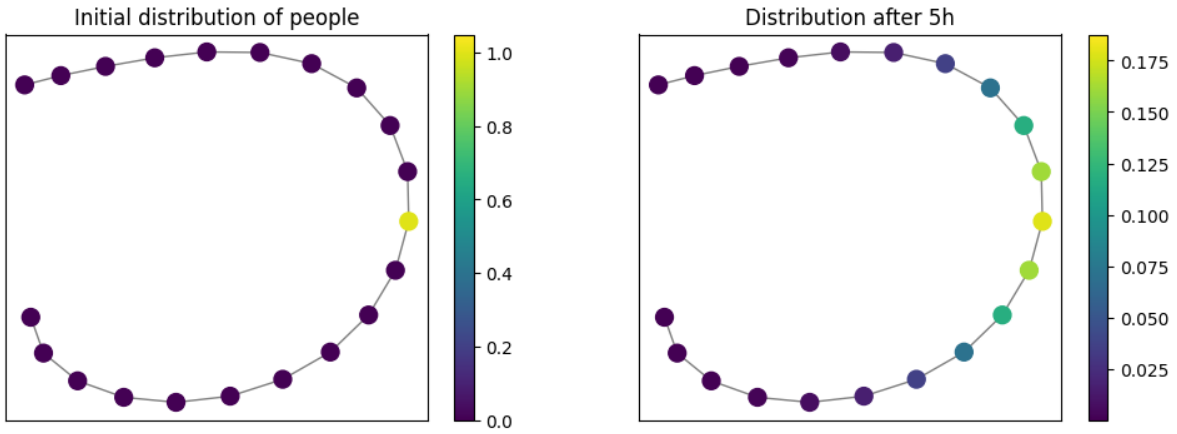


Figure 11: A depiction of the real physical process. The day starts with all people gathered at the 11-th station, which corresponds to the yellow dot in the left subfigure. We renormalize the population to 1, so the signal is the proportion of people present at each station at a given point in time. The initial condition is a delta over the 11-th station: a vector in $\mathbb{R}^{21}$ of all 0 except a 1 in the 11-th entry. We let the equation diffuse the people through the graph 5 units of time, we call time units hours for intuition purposes. The diffused people are seen in the right subfigure. We observe that there is still a significant number of people in the 11-th station, but the rest have spread through the graph.

We are tasked by our informant to guess which is the map of the Metro —which stations are connected to which— just from the distribution of the people at $t = 5$. The information available to us is depicted in Figure 12.

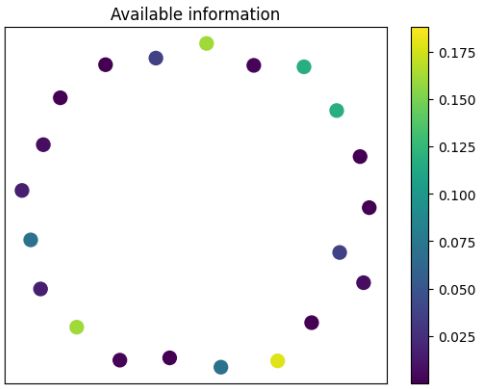


Figure 12: One signal x, we don't know the topology (the map) of the Metro, it is our task to infer it from this signal.

From only one signal $X \in \mathbb{R}^{21\times 1}$, the vector in Figure 12, we run the algorithm, and get the left subfigure in Figure 13: a K_{21}, a complete graph. We show the result to our informant, and he tells us that it is totally wrong: there are only 20 connections in the Metro system. This corresponds to our first critique: without knowing the groundtruth number of edges, we wouldn't know how many edges to delete. With this information available, we delete all but the 20 most weighted edges, and our result, after thresholding, is the subfigure on the right: a chaotic nonsensical Metro. This corresponds to our second critique: without enough signals, we won't be able to recover the groundtruth graph.

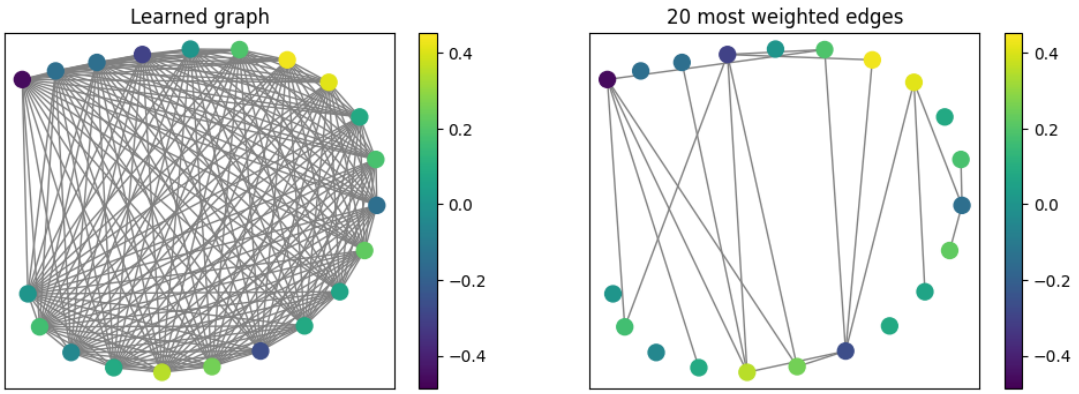


Figure 13: To the left, raw output of the algorithm $L_{learned}$ together with learned initial condition $H_{learned}$. As there is no meaningful way to enforce sparsity, all weights are bigger than 0, even though a fair number of weights will be really small, around 10^{-n}, $n > 3$. To the right, 20 most weighted edges, threshold as in [31]: there are plenty of initial conditions and graphs that can lead to the same diffused vector, the problem is completely underdetermined.

As the reader can observe, the depicted learned initial condition takes negative values, even though the concentration of the population is always non-negative. This is a topic that we did not cover. As the results show in Figure 15, the Laplacians can be correctly learned with enough information and thresholds, but the recovery of the learned initial condition, which in [31] is

addressed as source recovery, is completely ill-posed. The algorithm returns H and τ because it has used them and has iterated over them in order to learn L, but it is not its purpose to return meaningful H and τ. τ is restricted to be positive in the optimization problem, but not H, the user of the algorithm can modify it to include $H_{ij} \geq 0$ as a constraint.

3.3 Required number of signals to learn a graph

In [31], to obtain each signal X_i for the experiments, they pick a sparse H_i, in particular, they randomly choose three entries of the column and fill it with i.i.d. samples of a $\mathcal{N}(0,1)$. Then they construct each column of the signal matrix X as

$$X_i = \mathcal{D}H_i + \epsilon$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is gaussian noise. They will set $\sigma^2 = 0.02$ in their experiments to test the resilience of the algorithm to noise. The signal creation has this form to make it stationary, as they want to compare the results to [27]. One of the drifts from the original paper lies in the answer to this question

How many H_i do we need to retrieve the dictionary $\mathcal{D}$ from X without noise?

To retrieve $\mathcal{D}$ we need at least $M = NS$ full-rank initial conditions. The intuition behind this is that, in order to know a graph through a diffused process, we need to let information diffuse at each v_i at each diffusion rate, when we see how they have diffused, we can infer how the node v_i is connected with the rest of the graph. Mathematically, we need to invert the initial condition matrix to get to the matrices acting on it

$$\mathcal{D} = XH^{-1}$$

then to each block $e^{-\tau_s L}$ we apply the matrix logarithm. As L needs to have trace n (by definition), we can obtain L and each τ_s from $\mathcal{D}$. If we have more signals than needed ($NS < M$), as long as the signals are consistent, we just need to take a subset NS signals from the M available.

Were the signals not sparse, then the probability of having a full rank H matrix after generating N signals (over a graph with N nodes) would be 1, as the singular matrices have measure 0 in $\mathbb{R}^{N\times N}$. Nevertheless, as our algorithm has—and needs—a sparsity assumption (without a regularizer the inverse problem is numerically ill-posed), this question needs to be answered with sparse signals. In particular, if signals are just deltas chosen at random, which is the case in the real-life experiments made in [31], then the problem is the well-known **Coupon collector's problem** (and if signals are sparse with k non-zero entries then it is the generalized problem). This well-studied problem asks what is the probability, after buying N envelopes which contain random cards from a collection, to completely fill an album. The expected signals needed to have a rank N matrix H would be

$$\mathbb{E}[M] = nH_n \approx n \log n$$

where $H_n = \sum_{i=1}^{n} \frac{1}{i}$, the n-th harmonic number. In the previous formulation: if the album has n cards, then we need, on average, to buy $n \log n$ envelopes to fill it. This is an interesting

subject on its own studied also by Paul Erdős and Alfréd Rényi, see [11] for proof and tail estimates. The distribution of M is not trivial, in fact, it is somewhat complicated, but it is reassuring to see that the expected number of signals doesn't blow too quickly, $\frac{M}{N} \approx \log N$ is rather convenient for big N.

Proposed methodology: diffused deltas

To avoid the probabilistic problem mentioned above, and to isolate the randomness of our experiments to only one parameter, we propose to feed the algorithm just the right amount of information to fully learn the Laplacian. So, in this thesis, there will be a signal for each node N and each diffusion process τ_s which will be $e^{-\tau_s L}\delta_i$, the diffused delta at the node v_i one unit of time at rate τ_s, see Figure 14.

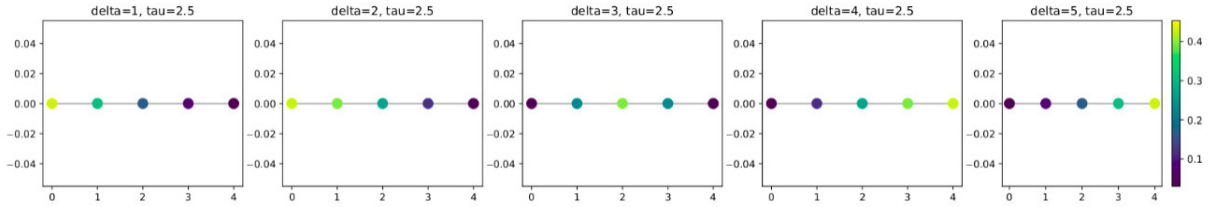


Figure 14: $X_{5+i} = e^{-2.5L}\delta_i$ for $i = 1, \ldots, 5$

Any signal can be written as a linear combination of these diffused deltas. A diffused delta is just a signal concentrated in a node that has diffused. We will exclusively feed these signals to the algorithm. Signals will still be random each time, but now the randomness is exclusively loaded onto L, the topology, not onto H.

3.4 The challenge of choosing a threshold without priors

To talk about thresholds, mainly: how well do we threshold, we will use the following tools [26]. tp and fp refer to true and false positives, tn and fn refer to true and false negatives. A positive, which is an off-diagonal non-zero entry in the lower (or upper, it is symmetric) part of the learned Laplacian $L_{ij} < 0$, can be true if, in the groundtruth Laplacian —the graph that has diffused the signals—, $\hat{L}_{ij} < 0$, or false if $\hat{L}_{ij} = 0$. Substituting $L_{ij} < 0$ by $L_{ij} = 0$, we have the definition for true and false negatives.

Definition 14 (Precision, recall and F-measure) *Given a learned Laplacian, from now on, $L_{learned}$, and the groundtruth Laplacian L_{ground}, their binary classification scores are defined as*

$$precision = \frac{tp}{tp + fp}, \qquad recall = \frac{tp}{tp + fn}, \qquad F\text{-}measure = \frac{2}{precision^{-1} + recall^{-1}}$$

We use F-measure as our score. It takes values in $(0, 1]$ and will be 1 when $precision = recall = 1$, which means $fp = fn = 0$ so the learned graph is isomorphic to the groundtruth graph, ignoring weights. Now we will explain how the graphs are thresholded. Given a learned

graph $L_{learned} = L$ and a real number $\rho \in [0, 1]$ we threshold L by ρ with the following formula

$$L^{\rho}_{ij} = \begin{cases} L_{ij} & \text{if } |L_{ij}| > \rho \text{ or } i = j \\ 0 & \text{else} \end{cases}$$

when the exact value of ρ is not relevant, we will refer to L^{ρ} as $L_{threshold}$ or L_{thresh}.

TABLE I
GRAPH LEARNING PERFORMANCE FOR CLEAN DATA

Graph model	*F-measure*	*Precision*	*Recall*	*NMI*	ℓ^2 *weight error*
Gaussian RBF (LearnHeat)	0.9779	0.9646	**0.9920**	0.8977	0.2887
Gaussian RBF [14]	**0.9911**	**0.9905**	0.9919	**0.9550**	**0.2081**
Gaussian RBF [8]	0.8760	0.8662	0.8966	0.5944	0.4287
ER (LearnHeat)	**0.9303**	**0.8786**	**0.9908**	**0.7886**	**0.3795**
ER [14]	0.8799	0.8525	0.9157	0.65831	0.3968
ER [8]	0.7397	0.6987	0.8114	0.4032	0.5284
BA (LearnHeat)	**0.9147**	**0.8644**	**0.9757**	**0.7538**	0.4009
BA [14]	0.8477	0.7806	0.9351	0.6009	**0.3469**
BA [8]	0.6969	0.6043	0.8459	0.3587	0.5880

TABLE II
GRAPH LEARNING PERFORMANCE FOR NOISY DATA

Graph model	*F-measure*	*Precision*	*Recall*	*NMI*	ℓ^2 *weight error*
Gaussian RBF (LearnHeat)	**0.9429**	**0.9518**	**0.9355**	**0.7784**	**0.3095**
Gaussian RBF [14]	0.8339	0.8184	0.8567	0.5056	0.3641
Gaussian RBF [8]	0.8959	0.7738	0.9284	0.5461	0.4572
ER (LearnHeat)	**0.8217**	0.7502	**0.9183**	**0.5413**	**0.3698**
ER [14]	0.8195	**0.7662**	0.8905	0.5331	0.3809
ER [8]	0.6984	0.5963	0.8690	0.3426	0.5172
BA (LearnHeat)	0.8155	0.7503	0.8986	0.5258	0.4036
BA [14]	**0.8254**	**0.7613**	**0.9068**	**0.5451**	**0.3980**
BA [8]	0.7405	0.6800	0.8230	0.3980	0.5899

Figure 15: Scores obtained by the authors of the paper using the number of edges as priors. (14) refers to the spectral template algorithm [27]. 8 refers to the smooth learning algorithm [9] explained in this chapter Section 2.2. Table I corresponds to noiseless generated signals. Table II contains noisy signals $\sigma = 0.02$. NMI refers to normalized mutual information. ℓ^2 weight error, or just ℓ^2 error, will be defined and utilized in Chapter 2 Section 2. We mainly consider F-measure in this thesis. Further material on binary classification scores is found here [26]. Figure found in [31].

As we said earlier, the authors in [31] carried out multiple experiments obtaining excellent F-measure results, (shown in Figure 15). As good as these results look, they are deceiving. Let's try to grasp why through, again, an example. Suppose an informant gives you X, a set of signals from Line 6 in Metro de Madrid, a circular line (suppose that the line is isolated from the rest of the lines). They ask you to apply the algorithm you just learned to these signals and they assure you that by using $\alpha = 0.01$ and $\beta = 0.1$ as regularizers, you will recover L. Then you run the algorithm. As you expected, you get a fully-connected graph, and you start to threshold, as in Figure 16.

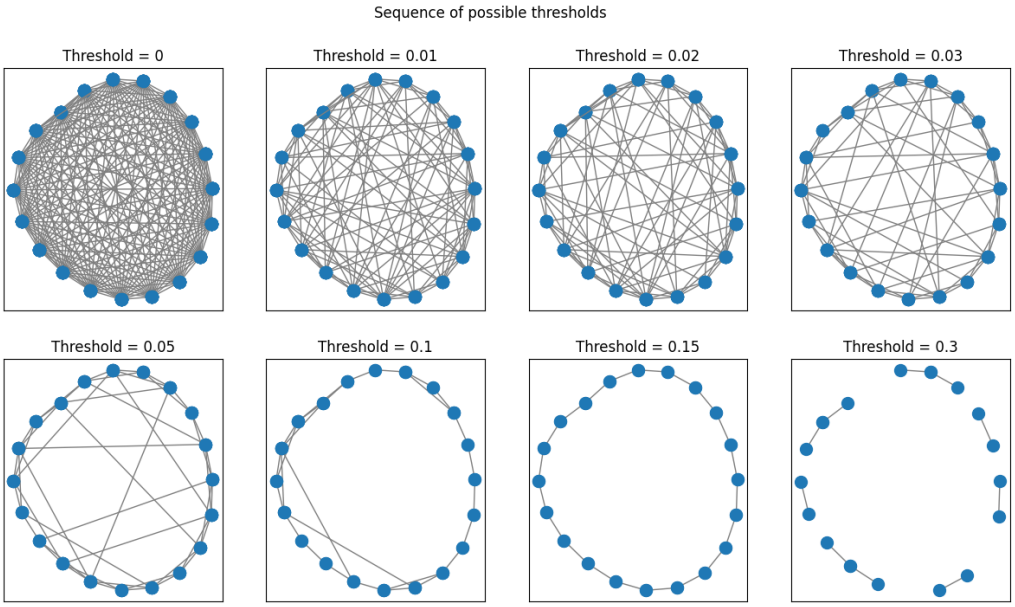


Figure 16: Main weakness of the algorithm: which one of these is closer to the groundtruth graph? At each subfigure, we delete all weights below the threshold stated in their title. It looks like there is no meaningful way to choose one without prior information.

The problem which we want to highlight is that, without any a priori information of the graph, such as sparsity or structure, *any of these graphs could be the groundtruth graph, even the complete graph.* Moreover, if optimal regularizers α and β are not used, the real groudtruth graph might not be hidden in the learned graph. Maybe there is no threshold with which $L_{ground} \cong L_{threshold}$. This constitutes an even bigger problem, as optimal α and β will change as graph structure, number of signals, and number of nodes change. In this thesis, we use the optimal parameters stated in [31], but we want to state clearly how not knowing the optimal parameters might affect results. To show how, we will use precision-recall curves. When a parameter (the threshold) is controlling a binary vector (if the threshold is the maximum weight, all weights become 0, if we don't threshold, all weights stay above 0), we can go through the different possible values of the threshold and record the different precision-recall values of the different graphs (think of Figure 16).

Our objective with $L_{learned}$ would be to somehow get access to all precision-recall curves for all combinations of parameters that we consider and pick the parameters with the curve closer to $(1, 1)$ (see Figure 17). This is close to what they do in [31], they check the results for a wide range of combinations (α and β) and then they pick the best-performing pair.

Finding optimal regularizers for a given graph is beyond the scope of this thesis. Contrary to what is claimed in [31], in our experiments, β does not properly enforce sparsity. The regularizer, recall, was $\|\cdot\|_F$, which is nothing but the ℓ^2 norm of the vectorized matrix $\|\text{vec}(\cdot)\|_2$, which, to no surprise, does not enforce sparsity. It also does not control volume, that task is loaded to the trace constrain, which makes it so that the sum of all weights grows linearly with the size

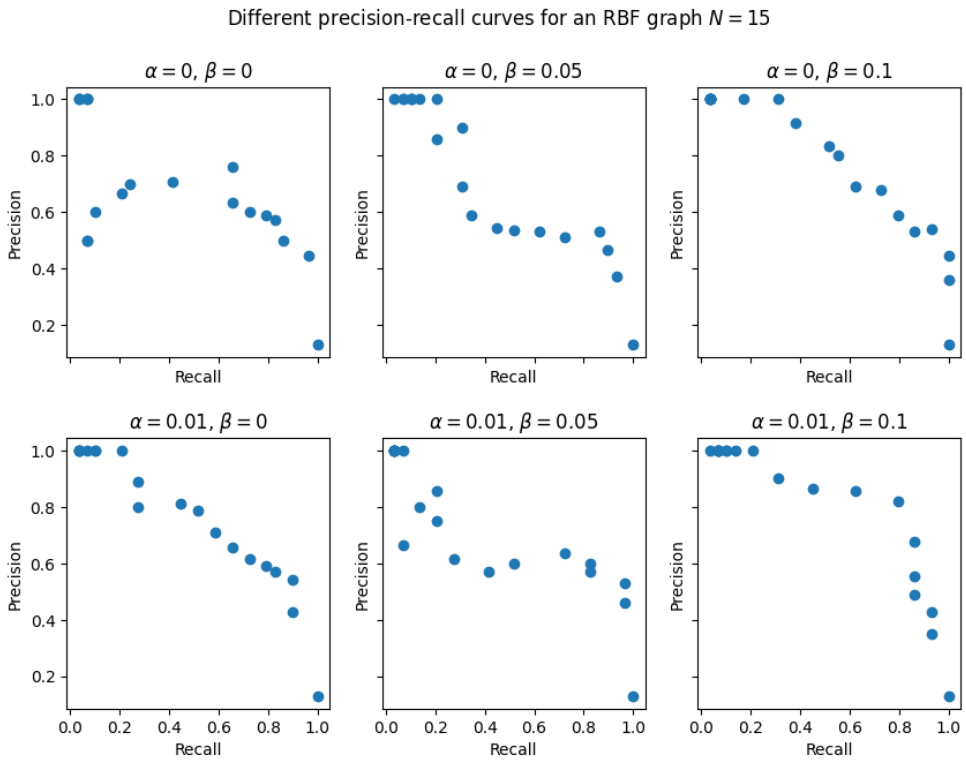


Figure 17: Figure depicts 6 precision-recall curves from one set of signals with 6 combinations of regularizers. Each point in each subfigure represents the precision and recall scores for one threshold $\rho \in [0, 1]$. Results consistent with those in [31], as the optimal parameters, the last subfigure, go the closest to $(1, 1)$.

of the graph

$$\sum_{(v_i,v_j)\in E} w_{ij} = n$$

The β-regularizer has already found its critique, as we said, in [15]. Optimal β can be used, and it works, with certain graphs of a certain size, but there is absolutely no guarantee that it will work with, for example, bigger $N \approx 300$ graphs.

In the next chapter, we will explore some solutions to the problems we have just mentioned.

Chapter 2

Proposed solutions to the thresholding problem

1 Presentation and evaluation of two thresholds

In this section, we introduce and validate two new thresholding algorithms. In order to assess their performance, we first threshold the learned graphs (BA, ER, and inverted[1] RBF, defined in Chapter 1, Section 3) through the quantiles of the non-zero weights. We prefilter all learned weights by 10^{-3} to account for round-off errors in order to make fair comparisons between the raw learned graphs and our thresholds.

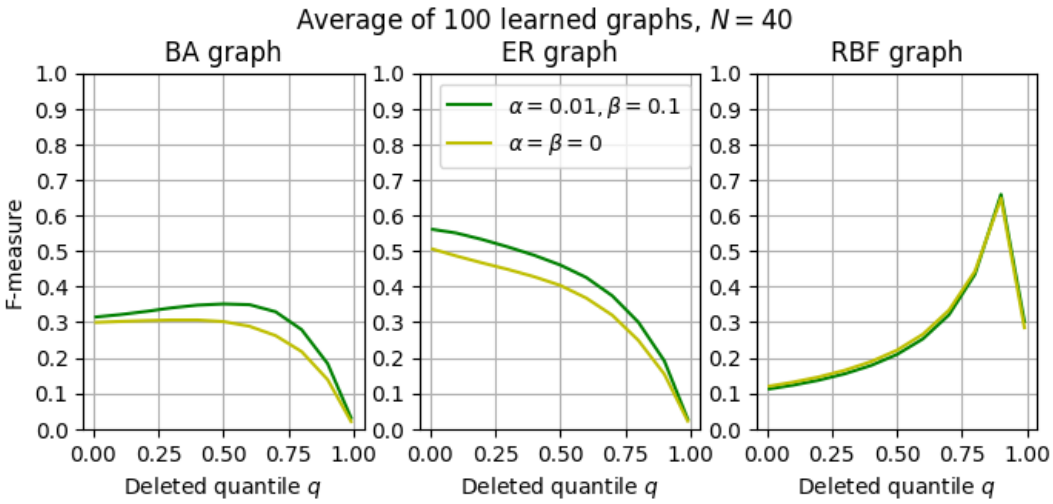


Figure 18: Quantile filtering, in the x-axis, the quantile of deleted edges, in the y-axis, the F-measure of the filtered Laplacians. The maximum of these curves represents, with an error of $\pm 0.05 \cdot |E|$, the best threshold we can get. Optimal results—maximal values of the curves—are not as good with $N = 40$ as with $N = 20$ using $\alpha = 0.01$ and $\beta = 0.1$.

[1] We chose to invert the cutoff to have a wider variety of graphs as the non-inverted RBF exhibited too similar behavior with the ER graph.

We can take Figure 18 as a graphical reference for the optimal threshold. The curves represent, for each combination of parameters, the F-measure obtained after deleting 1%, 10%, ..., 90%, 99% of the least weighted edges. The better our thresholds are, the closer they will be to the maximum of the curve. The purpose of the 0.99 quantile is to account for cases where the number of edges grows sub-quadratically. For instance, the case of the BA graph. This enables us to study the limit of the F-measure at the total deletion of edges (see how it spikes in the BA graph). 0.01 was used just for convenience.

The computer used for the experiments was equipped with an AMD Ryzen 5 5625U CPU with 12 cores. We used Python version 3.8.10 and networkx[2] for the ER and BA graph generation. RBF was done by ourselves. We generated 100 random graphs, each of $N = 40$ nodes, adding up to a total of 300 graphs created. Each graph was fed to `LearnHeat`[3] with two combinations of parameters. The algorithm was executed 600 times 50 iterations each, using 30 hours of computing time for just Figure 18. Results were prethresholded by 10^{-3} which for $N = 40$ is clearly noise. This does not play in our favour but it is made as a fair comparison to the no-threshold scores.

Our main objective is to offer a systematic way of thresholding $L_{learned}$ —deleting weights from below— which is

- Better than not thresholding at all.
- Agnostic to regularizers—it shouldn't depend on α and β.
- Sparsity independent—it should not make use of explicit assumptions on the sparsity of the underlying graph, it should not directly take the x most weighted edges.
- As general as possible—we do not want algorithms that work perfectly only in our study graphs. Instead, we want to make use of general concepts.

Given those four objectives, our heuristic assumption will be:

Groundtruth graph weights have compact support.

This is true in most common graphs. In the RBF graph, for example, due to our cutoffs, all weights, before dividing the Laplacian by the trace, cover the interval $[\exp(-\kappa^2/2\sigma^2), 1]$, which is a compact set. Our insight is as follows: the algorithm is unpredictable, it starts iterating at a random Laplacian each time and has multiple local minima, but we believe that the boundary of the support of the weights might be carried through the gradient descent to a more concentrated area of weights. Thus, if we detect an accumulation point of the weights, we threshold there, as the lower bound of the original weights might have been carried there. This phenomenon is depicted in Figure 19.

[2] https://networkx.org/
[3] https://github.com/semink/learnHeat

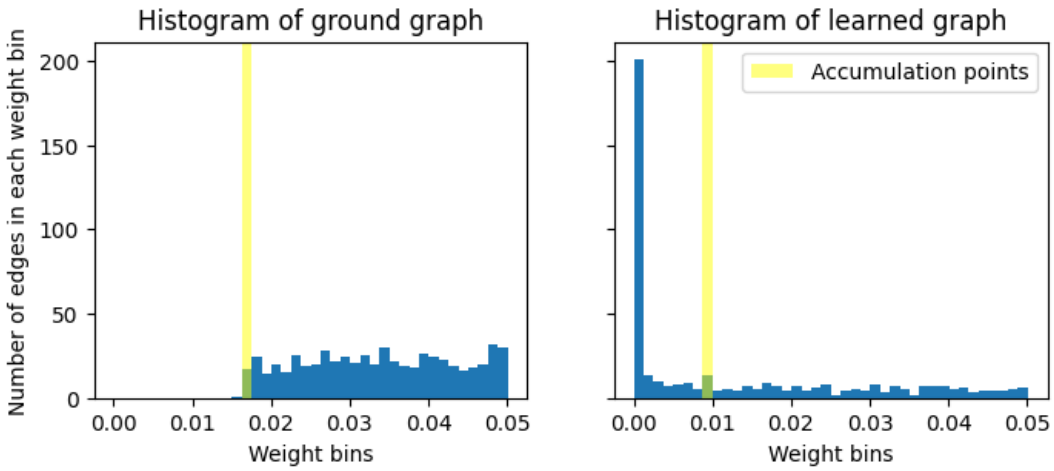


Figure 19: RBF groundtruth graph. The edges lying around the accumulation point on the left subfigure, which is a lower bound for the weights, are carried to the weight cluster depicted in the right subfigure with a yellow stripe. If we delete all weights before that cluster, we expect to only delete the edges mislearned by the algorithm.

Remark: When $n \to \infty$, as we work with normalized (standard) Laplacians, the minimum weights w_{ij} will go to zero in a controlled fashion $\mathcal{O}(n^p)$. In the case of the ER graph, as the expected value of edges is quadratic in n, weights will go to zero as $\mathcal{O}(\frac{N}{2|E|}) = \mathcal{O}(n^{-1})$, whereas the BA graph, which is a tree, will have $\mathcal{O}(1)$ weights for all n. In the RBF case, there is a lower and upper connectivity bound, $\exp(-\kappa^2/2\sigma^2)$ and either 1 or $\exp(-1/2\sigma^2)$—depending on how one takes the cutoff—, which again, after normalizing the Laplacian to have trace equal to n, will make weights go to zero as $\mathcal{O}(n^{-1})$.

1.1 First algorithm: searching small gaps of weights near $w_{max}/2$

This was the first algorithm that was better than random in a big number of PyGSP[4] sample graphs. The idea is to look at the biggest weight learned w_{max} and divide it by two, then pick a threshold close to that weight which looks like an accumulation point. To detect these accumulation points, we resorted to the idea of persistent homology, a central topic in Topological Data Analysis (TDA) [29], [4], see Figure 20.

[4]Python library for graph signal processing.

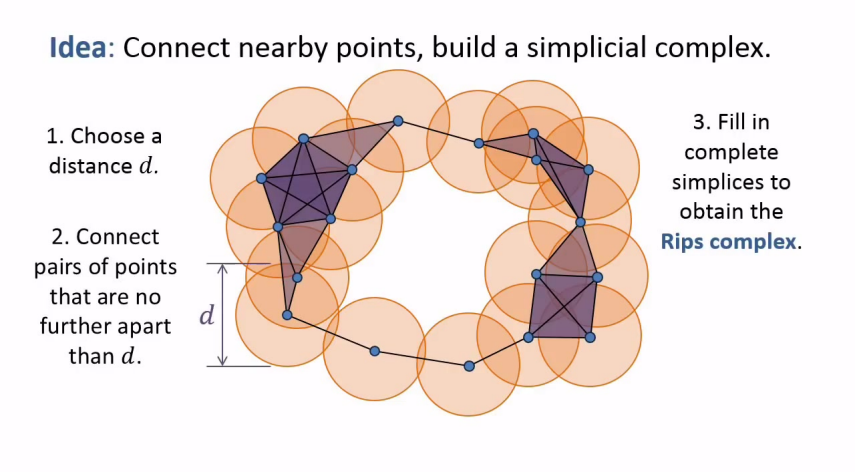


Figure 20: Simplicial complexes (topological spaces) can be built from a cloud of points and a given distance d, which would be our threshold. To choose the distance d, one often looks at homology and how it persists through time. That technique is called persistent homology. We can simplify this to graphs by taking d as a threshold and studying which is the biggest gap for which the rank of the homology groups doesn't change.

We implemented the persistent homology algorithm through a histogram, dividing the continuum of weights into small intervals Δd. The algorithm was returning sparse graphs, as there are weights which get bigger with the algorithm and create a maximal gap on the right part of the histogram (see 21). The weight continuum is contaminated by plenty of residual weights from the PALM descent, breaking a lot of big gaps where the threshold should be taken.

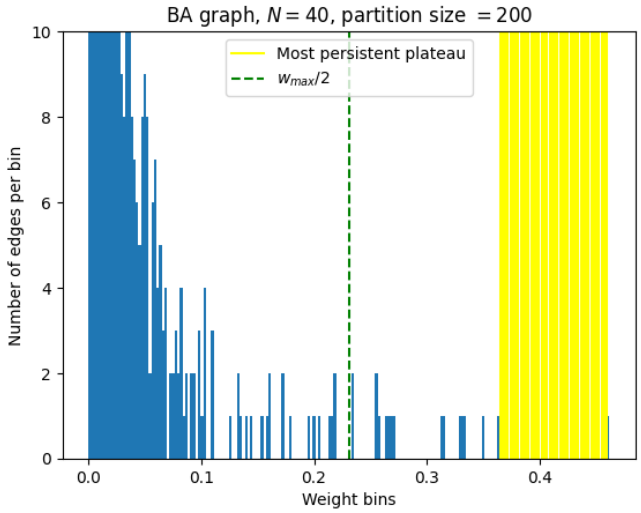


Figure 21: Figure showcasing a reinterpretation of the persistent homology algorithm applied to weighted graphs. We search for the graph which persists the most through time. In a histogram: the widest gap, which we call plateau (stripe in yellow), between non-empty bins. An outlier at 0.5 makes the algorithm fail and returns a graph with only two edges.

The insight taken from this observation is to look from small gaps instead of big gaps. This lead to the **persistent algorithm**. Here is a schematic set of instructions. We will provide rigorous pseudocode at the end of this subsection and an explanation.

- Create a histogram of the weights with a certain partition size (number of bins).
- Assign a score to each non-empty bin: how many empty bins has to the left.
- Select the bins with minimum greater than zero score—the smallest plateaus in the histogram. Pick the closest to $w_{max}/2$. Use it as the threshold.

The following figure contains the results within the experimental setup mentioned earlier.

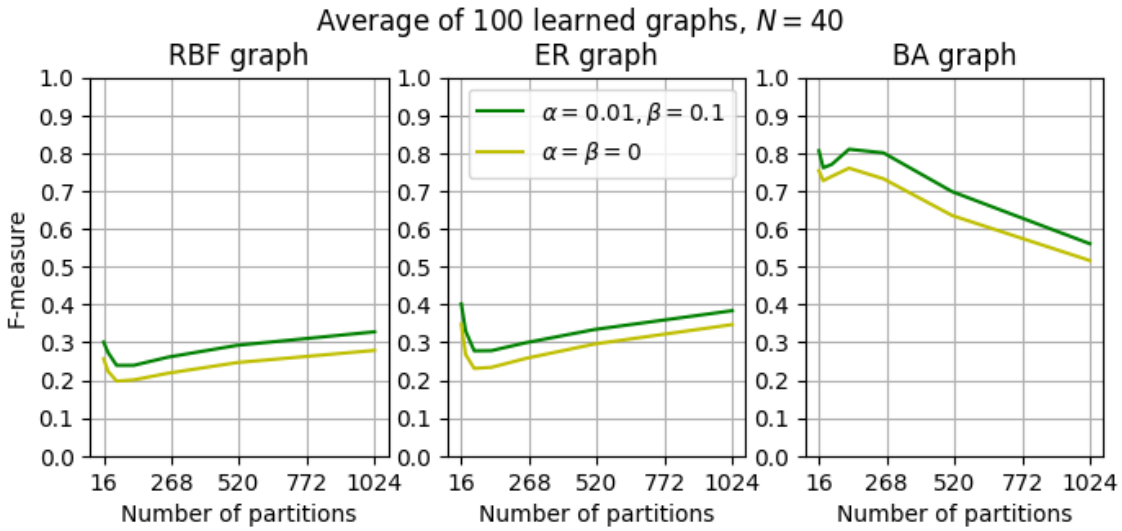


Figure 22: Least persistent filtering, the x-axis represents the number of partitions (the number of bins) of the histogram. Results are remarkable in the non-sparse two first graphs. In the RBF graph, the score at 1024 partitions is better than at 10. Taking 10 corresponds, most of the time, to just taking $w_{max}/2$. If compared with Figure 18 we can see that we are getting near the approximated optima, beyond there in some cases. The BA scores seem to go down with the number of partitions but even at 1024, the scores are near the maximum of the quantile curves.

The following figure contains the results compared with the quantile deletion curves, which represent, with an error of $\pm 0.05|E|$ edges, with $|E| \approx |K_n|$, the cardinality of a complete graph of n nodes. In the following figure we can see how, by searching for small gaps in the histogram, we manage to avoid both the big weight outliers and the cluster of weights around zero.

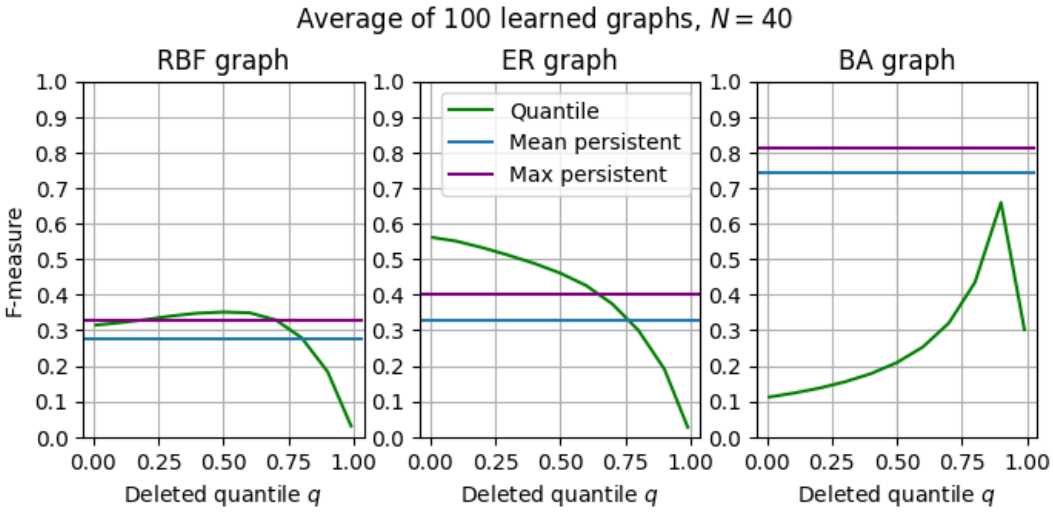


Figure 23: Note that the best results—which are obtained in both taking a lot of partitions or almost no partitions—outperform the no-threshold (0.001 threshold) policy except in the ER graph, which is to be expected from the structure-free nature of the ER graphs which makes weights wrongly ordered in the learned graph. It works best in highly sparse graphs such as the BA.

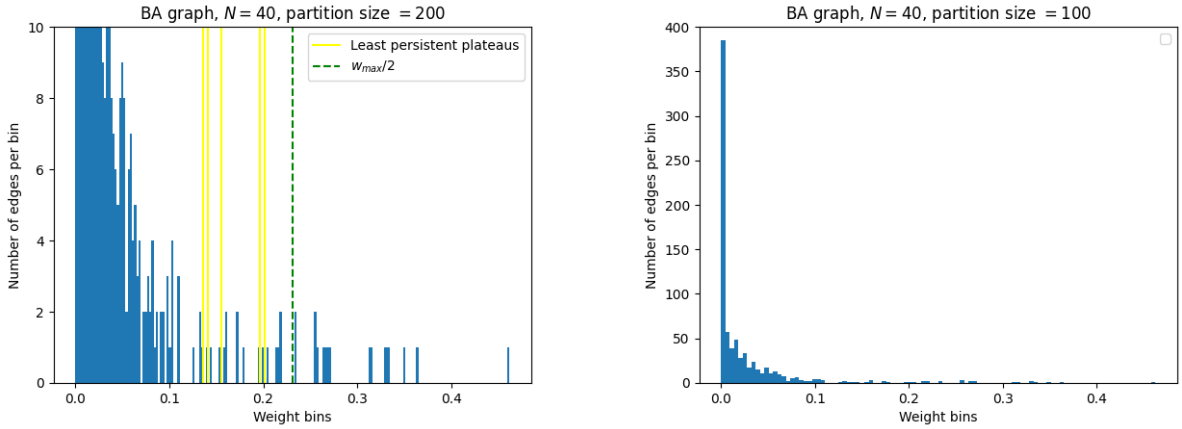


Figure 24: A graphical depiction of the algorithm. We search within all intervals of empty bins and choose the start of the smallest interval of empty bins as the threshold, if there is more than one interval with the same width (yellow lines have width 1 in this case), we pick the closest to the median bin (green line, $w_{max}/2$). To the right, zoomed-out histogram. Note the accumulation of weights around 0. None of them is exactly 0, smallest is around 10^{-7}.

The idea behind this heuristic algorithm lies in what it intended to do: perform better than random. We define a random threshold by sampling a random number from the uniform distribution in $[0, 1]$ $u \sim \mathcal{U}(0, 1)$ and multiplying it by the maximum weight, leaving $w_{max}/2$ as a threshold. The expected value of that threshold is $w_{max}/2$. We experimentally checked that the expected F-measure of $L^{uw_{max}}$ is smaller than the expected value of $L^{w_{max}/2}$ and thus

taking $w_{max}/2$ would already be a better option than random thresholds. Then we coupled this idea with the other insight about thresholds being carried to accumulation points to finally create this algorithm.

1.2 Second algorithm: moving window technique

The next algorithm remains simple and, unlike the persistent algorithm, takes into account the number of edges per bin. It consists in sweeping an indicator function, what we call *moving window*, through the histogram. At each step, we add up the number of edges in the window and, at the end, we go back to the step in which the window had the most edges and delete all weights before the window.

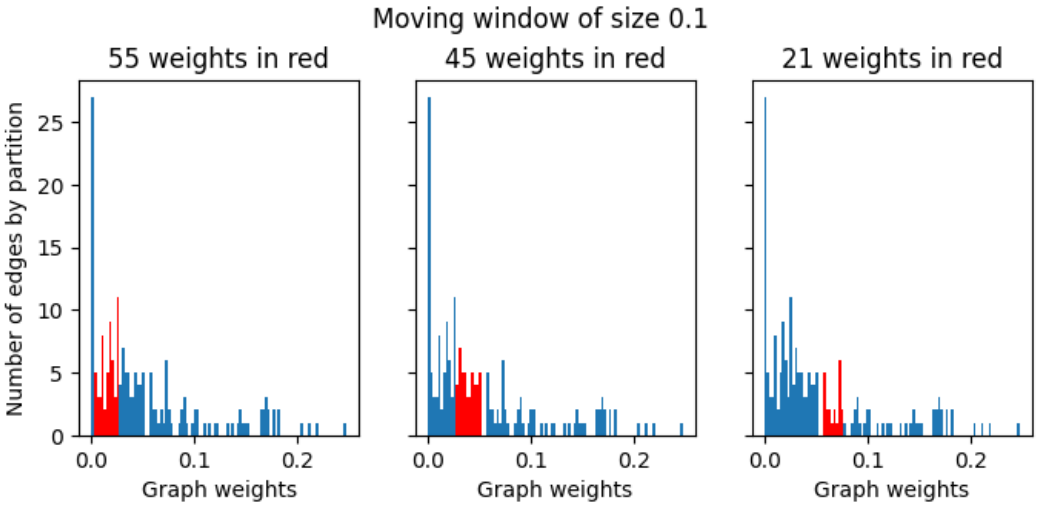


Figure 25: (Inverted) RBF graph. In blue: histogram of weights. In red: moving window. Goes from left to right counting the number of weights it contains inside at each step. After sweeping through the histogram, it returns the position in which it attained its maximum. We use that position as a threshold.

As we will see in the next figures, if we provide a good threshold—which we default to $4/N$—the results of the algorithm, unlike with partition size in the persistent algorithm, are stable across a wide range of window size values. This implies that in non-synthetic experiments the results will be most likely solid and usable without access to prior information. Furthermore, not only does the algorithm match the maximum of quantile curves, but in one case (RBF) almost outperforms it, which suggests that we are close to optimality. Recall that the maximum F-measure possible is not necessarily one, as getting a possible good F-measure relies on using optimal α and β parameters and overdetermining the algorithm with signals.

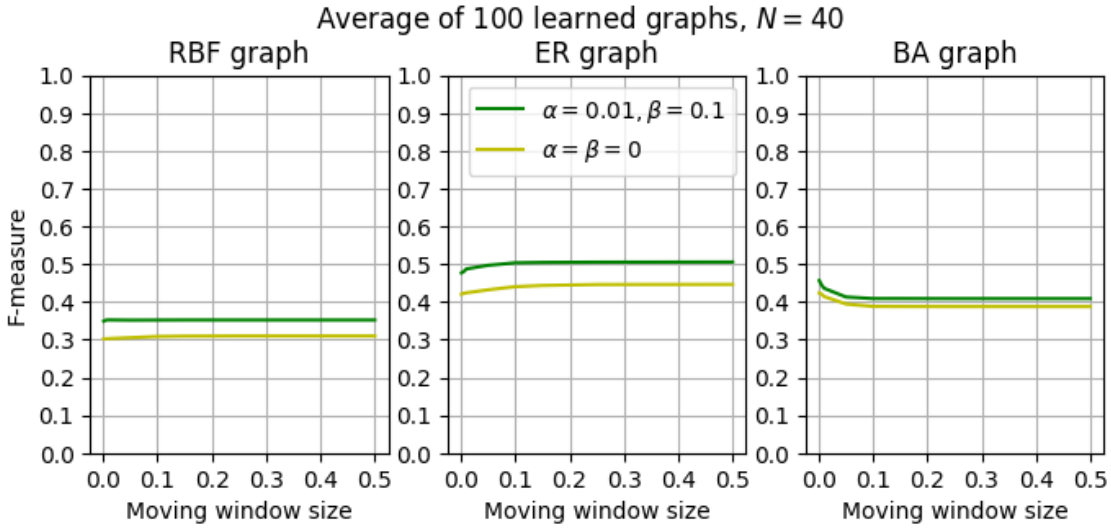


Figure 26: Moving window filtering, the x-axis represents the size of the window which we slide through the weights.

For the moving window, we obtain good stable results up to size 0.5, going in all cases near the best obtainable F-measure from edge-deleting for all parameters. The next

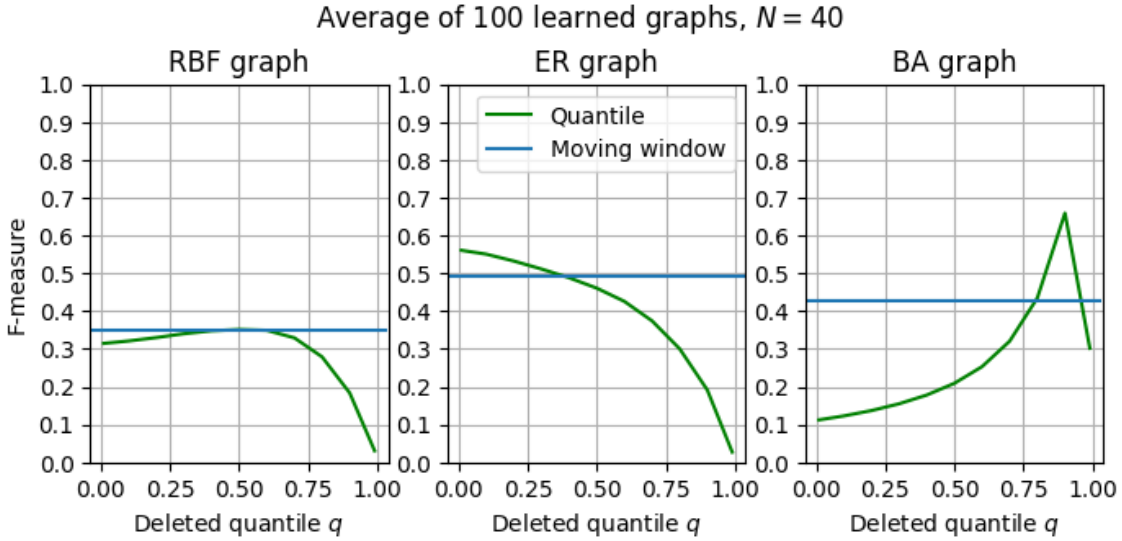


Figure 27: Comparing the mean of the results obtained via moving windows versus all possible quantile filters for $\alpha = 0.01, \beta = 0.1$. Note that the moving window F-measure value is taken as the average of all considered window sizes, therefore these results are representative of what we would get without any prior knowledge of the optimal window size.

We now provide the pseudocode to the algorithms written in a C-like manner.

Algorithm 2 Least persistent filter

Input: Laplacian L, precision N
Output: Filtered Laplacian $\hat{L}$
$count \leftarrow 0$
$current \leftarrow |\text{tril}(\hat{L})|$ ▷ Store number of edges
$dt \leftarrow \frac{1}{N}$
$jumps \leftarrow (0, \ldots, 0) \in \mathbb{R}^N$
for i=1:N **do**
    $\hat{L} \leftarrow L[L > i \cdot dt]$ ▷ Filter out weights smaller than $i \cdot dt$
    $e \leftarrow |\text{tril}(\hat{L})|$ ▷ Count edges
    $count \leftarrow count + 1$
    if $e \neq current$ **then** ▷ Check if number of edges has changed
        $jumps(i) \leftarrow count$ ▷ Store how many steps without a change in edges
        $current \leftarrow e$
        $count \leftarrow 0$ ▷ Reset count and edges
    end if
end for
$jumps \leftarrow jumps(2 : end - 1)$ ▷ Deprecate extrema
$i_{opt} \leftarrow m \in \text{argmin}(jumps)$ closest to $N//2$ ▷ Promote being close to the median edges.
$\hat{L} \leftarrow L[L > (i_{opt} - 1) \cdot dt]$
Return $\hat{L}$

Algorithm 3 Moving window technique

Input: Laplacian $L \in \mathbb{R}^{N \times N}$, size of window ω, threshold $thresh$ (default is $4/N$)
Output: filtered Laplacian
$\rho \leftarrow \min(L_{ij} \mid i \neq j)$
$W \leftarrow -L/\rho$ ▷ Normalize weights.
$W[W < thresh] \leftarrow 0$ ▷ Delete weight cluster around zero and diagonal.
$i \leftarrow 0$
while $\min(W) \neq 0$ **and** $i < N^2$ **do** ▷ $i < N^2$ security stop
    $\texttt{left} \leftarrow \min(W)$
    $W \leftarrow \text{threshold}(W, \texttt{left})$ ▷ Delete weights until actual weight
    $e_1 \leftarrow \text{number\_of\_edges}(W)$
    $e_2 \leftarrow \text{number\_of\_edges}(\text{threshold}(W, \texttt{left} + \omega))$
    $\Delta \leftarrow e_2 - e_1$ ▷ Number of edges between w_i and $w_i + \omega$
    $res[i, 0] \leftarrow \texttt{left}$ ▷ Store position
    $res[i, 1] \leftarrow \Delta$ ▷ Store number of edges in window
    $W \leftarrow \text{threshold}(W, \texttt{left} + \epsilon_{machine})$ ▷ Delete actual weight
    $i \leftarrow i + 1$
end while
$arg \leftarrow \text{argmax}(res[:, 1])$ ▷ Search which window has more edges
$opt\_thresh \leftarrow res[arg, 0]$ ▷ Save left limit to the window.
return threshold(L, $opt\_thresh \times \rho$) ▷ Rescale and threshold to return filtered L.

1.3 Further considerations on parameter scaling

In previous experiments we only considered $m = m_0 = 1$ of the BA graph, in this subsection we report our results with $m = 2, 5, 11$ and $N = 100$ nodes, as there is overall a special interest in scale-free networks as models of real-world networks.

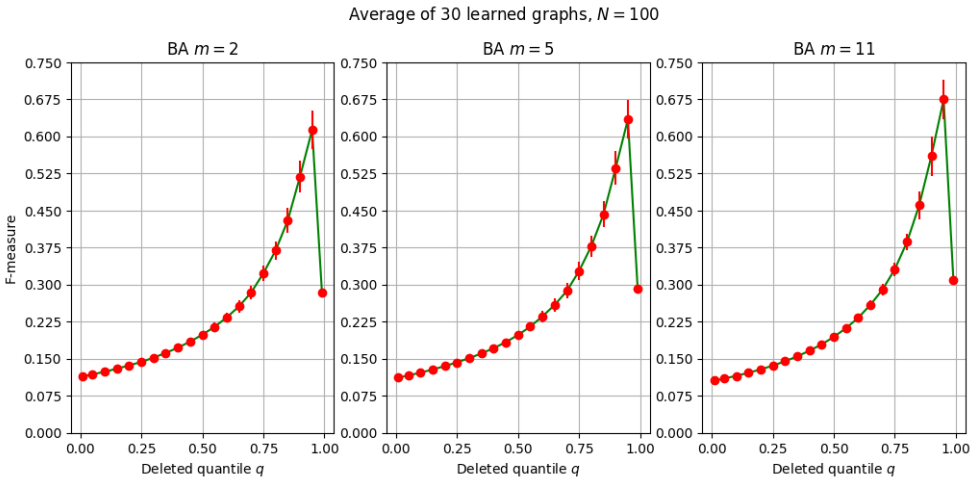


Figure 28: Running the quantile filtering over 90 random graphs of 100 nodes, each one starting with $m_0 = m$ nodes ($m = 2, 5, 11$) with a preferential attachment of new nodes through m edges. Error bars are standard deviations.

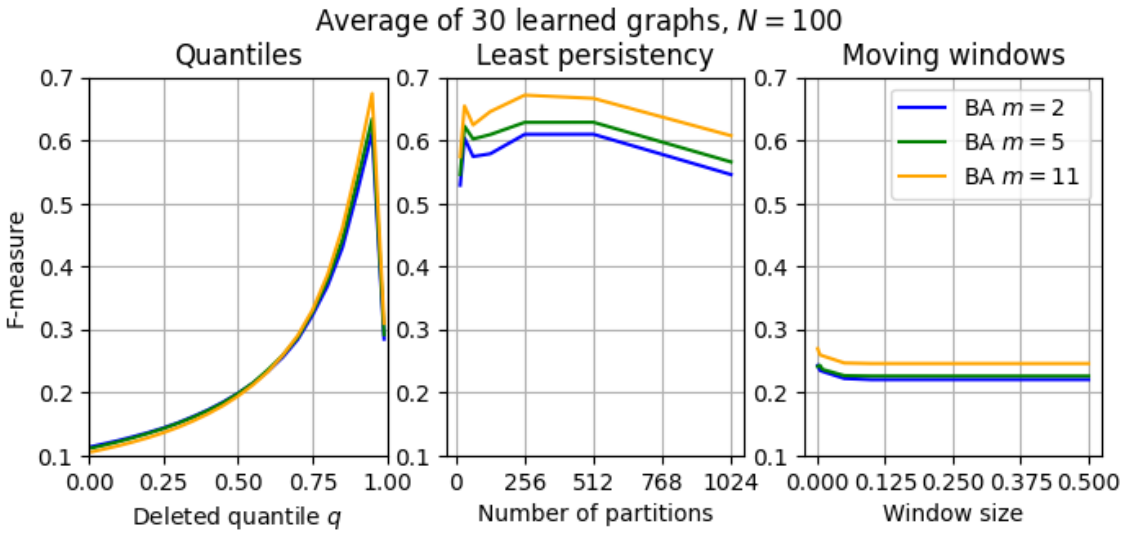


Figure 29: To the left, curves of deleted quantiles. On the center and on the right both thresholding techniques. The threshold we used for the moving windows was inappropriate, it remains to be explored how it needs to scale with N. The persistent algorithm scales perfectly, even obtaining better results for $N = 100$ than $N = 40$.

Note that in Figure 28 standard deviations are small, meaning that the algorithm is stable within the same family. We observe an acute shape, as we expect from sparse graphs: the number of false positives when we do not delete any edge drags the F-measure down. The next natural question is: Can we guess, for an arbitrary graph of N nodes, what is the optimal number of bins to take in the histogram used in both algorithms?

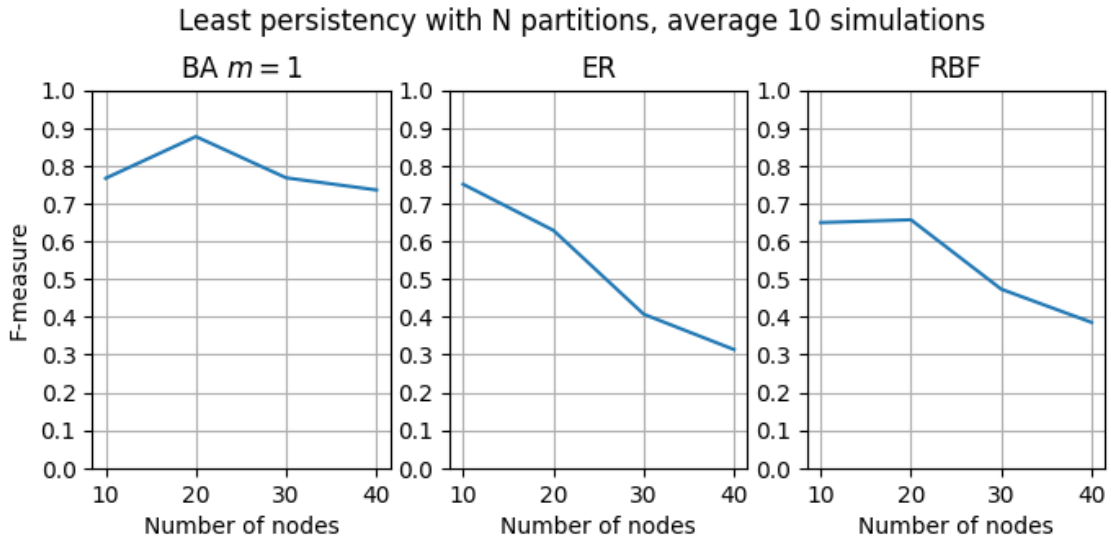


Figure 30: Scaling of the N partition technique. In the x-axis: number of nodes of the synthetic graphs. In the y-axis, the average F-measure after applying the persistent algorithm with N bins in the histograms. For the BA graph it works nicely, not with ER and RBF, we speculate that this discrepancy happens due to different density patterns.

To answer the previous question, an investigation needs to be done over the parameters α and β. In Figure 30, the central issue is that the optimal parameters should change with the number of nodes N and we do not know which relation these three parameters have all in common, which leads to worse optimal scores in denser graphs (such as the RBF and the ER) which cannot be retrieved through the algorithms.

2 Further comments

In this brief section, we will justify why getting good F-measure scores is beneficial to the predictive power of the Laplacian. We will also play the devil's advocate and ask: what can we do if there is actually some prior information about the graph's topology. We will show that these priors can be loaded onto the algorithm and be beneficial to the F-measure score.

2.1 The importance of getting good F-measure scores

Let's start, as always, with an example. Let A, B, and C be an ensemble of cities connected through a line (groundtruth graph), with no connection whatsoever between city A and city C. Consider the same ensemble of cities but this time with a narrow connection between city A and city C, where few people can pass at the same time (learned graph). Suppose we observe the diffusion of the population in the first city, through multiple days of work, at multiple hours, and we run the learning algorithm to retrieve the topology.

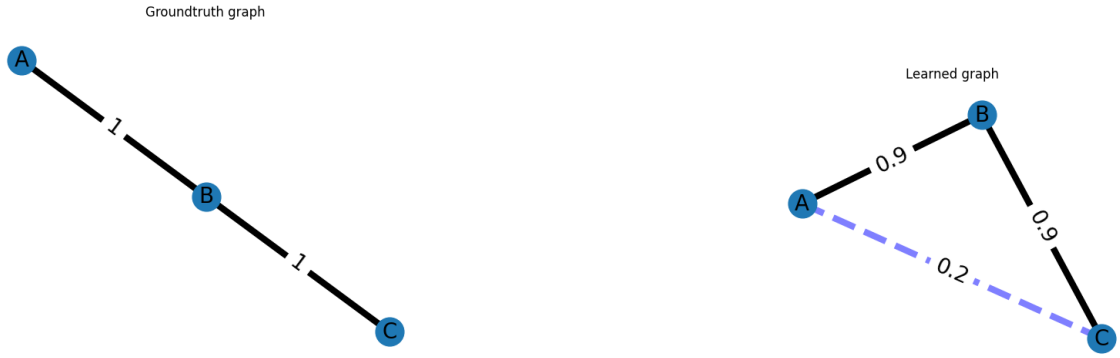


Figure 31: To the left, groundtruth graph, to the right, learned graph

As we have discussed before, the learned graph, with enough signals, will not be able to set w_{AC} to 0. The algorithm will return something similar to $w_{AB} \approx w_{BC} \approx 1$ and $w_{AC} \approx \epsilon$ with $\epsilon \ll 1$ before normalizing by the trace. We built two algorithms to solve this issue. The main concern now is: will deleting the ϵ weight make further predictions about diffused signals better or worse?

We have experimental evidence and theory backing up the idea that getting the topology right will make Laplacians predict better the diffusion process. When F-measure —technically only precision, reducing false positives— goes up, the pure loss goes down (see Figure 32). We believe that

$$\|X - \mathcal{D}(L_{threshold}, \tau_{ground})H_{ground}\|_F \lesssim \|X - \mathcal{D}(L_{learned}, \tau_{ground})H_{ground}\|_F \tag{1}$$

We have a heuristic argument based on this inequality [13]

$$\|e^A - e^B\| \leq \|A - B\| e^{\max(\|A\|, \|B\|)}$$

Substituting B by the real graph and A by our two candidates $L_{learned}$ and $L_{threshold}$, which fulfill the following inequality

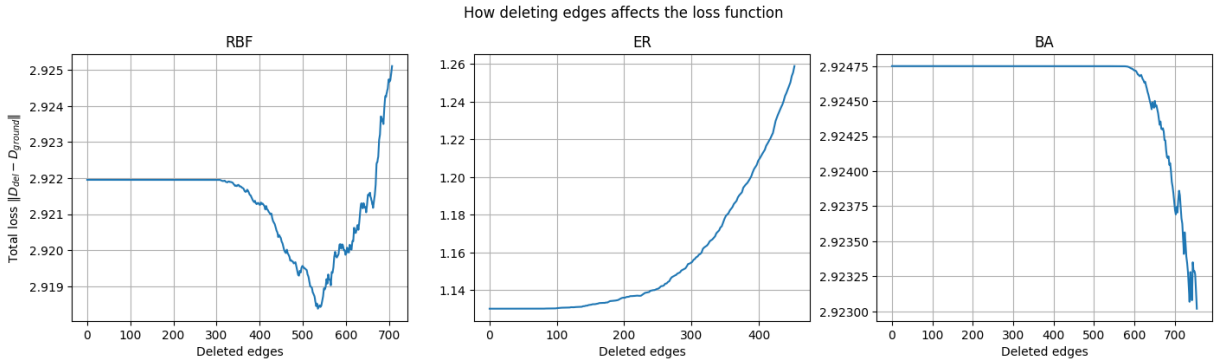


Figure 32: Change of the pure loss (no regularizers) w.r.t. deleted edges in our three study graphs. $H = \mathbf{I}, \tau = [0.5, 1, 2.5, 4]$, rest of the parameters as in previous sections. Only one simulation. Results as expected, compare with F-measure plots.

$$bound_1 = \|L_{threshold} - L_{ground}\|_F \leq \|L_{learned} - L_{ground}\|_F = bound_2$$

Together with $\|L\|_F = \mathcal{O}(N)$ (due to the trace constraints), we can split $X = \mathcal{D}H$ onto the sum $\sum_i e^{-\tau_i L} H_i$ and apply the triangular inequality and the bound to each summand of X to obtain

$$\|X - \mathcal{D}(L_{threshold}, \tau_{ground}) H_{ground}\|_F \leq S \cdot e^{C(N)} \cdot bound_1$$

and

$$\|X - \mathcal{D}(L_{learned}, \tau_{ground}) H_{ground}\|_F \leq S \cdot e^{C(N)} \cdot bound_2$$

with S the number of diffusion processes, $C(N)$ grows linearly with N and $bound_1 < bound_2$, which, together with the experiments, hints that Equation 1 might hold up over a wide range of practical cases.

2.2 Modifying the algorithm to include prior knowledge at runtime

In this subsection, we would like to address an issue that we haven't given enough attention to. In the previous subsection, we argued why including priors at post-execution is beneficial to the result (in the end, the thresholds we make are based on some controlled minimal connectivity), now we want to address the possibility of including the priors at runtime. Say we observed some signals over a graph whose topology we know already. Think, for example, of a railway network, the railroads of a city, or the flight connections network all over the world, to which the signals are the number of passengers at each node of the network. It is of interest to still apply the algorithm because we might know the topology, but not the weights, and we need them to simulate diffusion processes correctly, as assigning $w_{ij} = 1,\ \forall i, j$ might lead to catastrophic results. In this section, we show that priors can be effectively loaded as constraints in the optimization problem together with some experimental results.

Prior as constraints

Recall that for each L-update, we solve the following optimization problem:

$$\begin{aligned} \underset{L}{\text{minimize}} \quad & \langle L - L^t, \nabla_L Z(L^t, H^{t+1}, \tau^t)\rangle + \frac{d_t}{2}\|L - L^t\|_F^2 + \beta\|L\|_F^2 \\ \text{subject to} \quad & \operatorname{tr}(L) = N, \\ & L_{ij} = L_{ji} \leq 0, \ i \neq j, \\ & L \cdot \mathbf{1} = \mathbf{0} \end{aligned}$$

The feasible set is a cone, so the optimization problem can be solved via splitting conic solvers, but the prize we have to pay for making the feasible set a cone is that it is not exactly the set of possible Laplacians. On top of the diagonal not being integer-valued, there is no restriction over the multiplicity of the 0-th eigenvalue. If, at execution, we want to make use of our information, we would add the following constraint

$$\chi_0(L_{prior}) \Longrightarrow \chi_0(L)$$

where χ_0 is the 0 indicator function acting entrywise on the matrices

$$\chi_0(L_{ij}) = \begin{cases} 1 & \text{if } L_{ij} = 0 \\ 0 & \text{else} \end{cases}$$

which enables us to load our prior beliefs onto the learned Laplacian. We can implement those constraints to the same solvers, SCS and MOSEK, but see how we are not imposing any rank to $L_{learned}$. We made a first exploratory example to see how the improvement in F-measure scales with the fraction of zeros covered from L_{ground}. The x label in the following graph is

$$\text{prior zeros / total zeros} = \frac{|\{L_{ij} \in L_{prior} | L_{ij} = 0\}|}{|\{L_{ij} \in L_{ground} | L_{ij} = 0\}|}$$

which reflects in some sense the fraction of available groundtruth information.

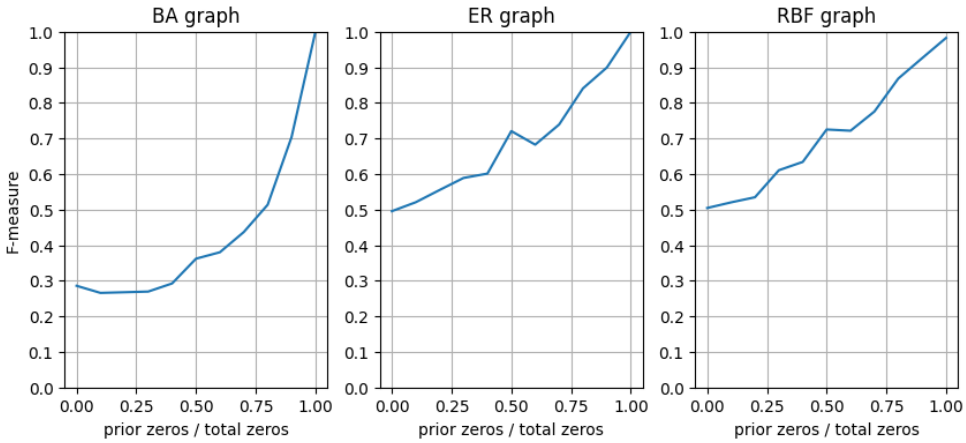
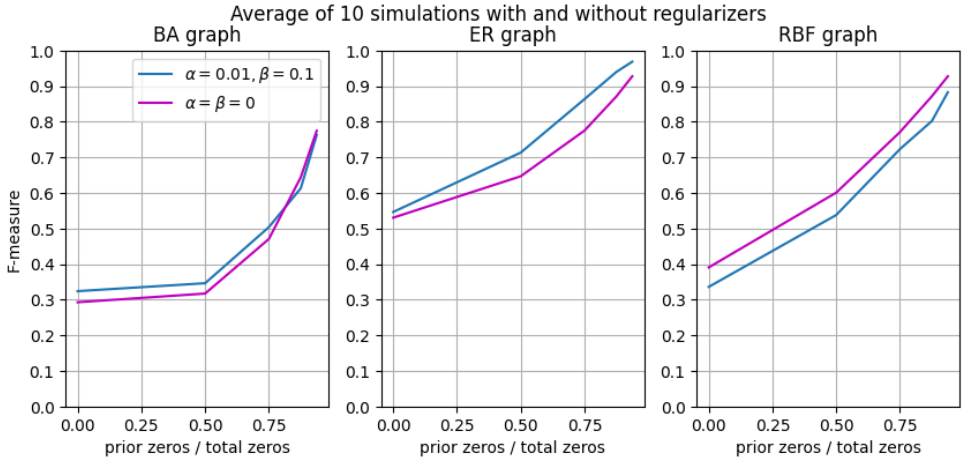
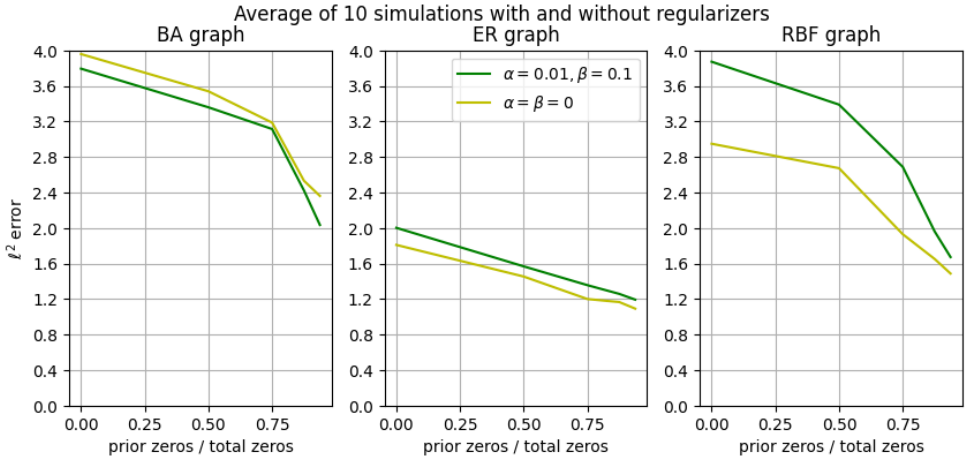


Figure 33: First exploration of usage of prior topological information for $N = 20$ sized graphs with usual parameters ($\alpha = 10^{-2}, \beta = 10^{-1}$). Threshold equal to 0.001, no additional post-processing.

In order to do several experiments and obtain statistically relevant results, we will carry 10 different experiments with $N = 20$ sized random graphs covering $1/2^i, \quad i = 0, \dots, 4$ of available information (thus having $1 - 1/2^i$ available zeros for learning) in the three different graphs, all of this with the usual regularizers, and then again without, which sums up to a total of $10 \cdot 5 \cdot 3 \cdot 2 = 300$ executions of the algorithm, each one with 50 iterations, a great computational effort.



We do not observe any interesting trait common to all graphs except that the F-measure is increasing, which is not theoretically necessary. The BA graph seems to be the one to struggle the most. We plot the ℓ^2 error with respect to the groundtruth graph $\|L_{ground} - L_{learned}\|_F^2$



This is what we are interested in and it certainly makes our point. Prior information on the topology helps adjust the weights and, furthermore, no regularization seems to help even more. We believe that the relation between available information and the measure scores doesn't follow any general power law applicable to every graph.

Chapter 3

Conclusions and further work

We have primarily focused our efforts on making a detailed critical review of the LearnHeat algorithm [31]. This algorithm has had a considerable impact on the GSP community since it was published, as acknowledged in [6], [10], [18], [32]. It is a brand new approach—dates back to 2017—and its mathematics are surprisingly rich. To fully understand it, we had to dive deep into plenty of additional references. The theory behind LearnHeat is built upon spectral graph theory [7], optimization [5], PDE modeling [12], and signal processing theory [25].

The bulk of our work was to understand and implement the algorithm. The interplay of fields we have mentioned makes the implementation by an inexperienced individual a difficult task. We managed to write a working implementation ourselves[1]. After getting what we thought were non-sensical results, we tried to get in contact with the authors. Thankfully, we found a repository[2] on GitHub from a member of the same university as the authors which contained an implementation in Python.

After running their code with a myriad of examples, we found that their implementation had some unaddressed issues—which we explain in Chapter 1 Section 3—the algorithm was learning complete graphs: graphs whose nodes were connected to every other node, as seen in Figure 13. In [31], no solution was given to this issue. In synthetic experiments, only the number of edges that they knew the groundtruth graph had were not deleted in the learned graph, prior information which is far from being available without access to the groundtruth. In real-life experiments—mentioned in Chapter 1 Section 1—only a handful of heavy edges were kept, following visualization criteria. Due to this fact, the last part of our work has had an unexpected research aspect—discussed in Chapter 2. We managed to build two new thresholding algorithms, covered in Section 1 of the mentioned chapter, which, in structure-rich graphs, managed to improve the available options to date: not thresholding at all. This represents a net advantage without a downside risk, and the odds of being applicable to graphs present in nature look fair.

Our opinion on these issues is that, no matter how interesting the efficient implementation of LearnHeat might be, there is a big question mark as to whether the algorithm is really applicable. There is still a lot of investigation to be done. We hope that our humble contribution, or at least our insights, might be of interest to the active researchers in the field.

[1] My initial attempt can be seen here https://github.com/aner-sanchez/heat_learning
[2] https://github.com/semink/learnHeat

Leaving aside the implementation issues, we want to mention that the spectral template algorithm [27] exhibited similar results to the heat diffusion model in synthetic experiments [31], even without explicitly utilizing the knowledge of heat diffusion driving the signal. This speaks highly of [27], and we believe that non-physically based models should focus on that direction. The heat diffusion model, on the other hand, should aspire to extend itself to a more complex, nonlinear, differential equations over graphs framework.

We also want to mention how working with graphs presents challenges computationally. It is beneficial for the code structure to build a graph class that can perform all the routine operations. The challenge relies on using different libraries—networkx and PyGSP—at the same time. Each one has some desired functionalities. For example, PyGSP is made to do GSP and has a lot of tools to deal with graph signals. All our figures were made using that library. To our knowledge, there is no other library that can plot signals over graphs in Python. Nevertheless, apart from being poorly documented, it only serves as a GSP tool. When wanting to do graph embeddings or any procedure outside GSP—which we did to gather ideas for GSP—we need networkx. This adds up to the issue that adjacency and laplacians are stored as sparse matrices in both graph classes, so, unless you modify the libraries yourself, you need to manually call the `.todense()` attribute when doing algebraic manipulation over the matrices. The interplay of taking the attribute from the class, modifying it, and feeding it to another class or to the original class again was proven to be a difficult task.

Additionally, the time required for each experiment made it unfeasible to conduct desirable experiments with a large number of nodes N. Exploring and understanding how the algorithm works as N grows requires some sort of parallelization to reach meaningful values and a sufficient number of simulations, but parallelizing the code is only possible after fully understanding the algorithm, which creates a circular dependency. Our investigations were not that agile for this reason.

To conclude the thesis, we want to mention potential new lines of work which might carry this investigation further.

Algorithm acceleration. Several strategies can be explored:

- Implementing the algorithm in efficient programming languages such as C++ or Julia.
- Establishing theoretical bounds for the Lipschitz constant L to avoid backtracking—see Chapter 1 Subsection 2.3. It might be a good idea to explore the usage of symbolic libraries to solve this task.
- Exploring alternative parallel optimization algorithms, such as parallel coordinate descent [24], and explore the viability of Monte-Carlo methods [1] and [2] with this new formulation. This would improve speed and scalability.
- Truncating the diffusion operator and computing gradients through automatic differentiation [22] to avoid dealing with the matrix exponential. We have already managed to implement it using `ForwardDiff.jl` [23] and it should require further study to see whether automatic differentiation is worth it. If the answer is negative, go back to the

idea of computing the gradients and Lipschitz constants of the new truncated operator via symbolic libraries.

Mathematically backing up the heuristics. We would like to throw some light on the black-box nature of the PALM algorithm. If we could explore the landscape of local minima of the loss function and gather probabilistic information on the sparsity pattern of all minima, we could not only mathematically back up our heuristics, but also find better thresholds and better regularizers.

Convert the number of diffusion processes to a learnable parameter. In Chapter 1 Subsection 2.3, to start the LearnHeat algorithm, we provide a random τ vector which will be used to iterate along H and L. When providing τ, or randomly generating it, we make a strong prior assumption that goes unnoticed: the length S of the vector τ is the number of different diffusion processes which we believe are driving the signal across the graph. We did not comment on this as it is L and not τ the output of the algorithm of our interest. Nevertheless, it would be appropriate to know how a miscalibration of the number of diffusion processes S affects the Laplacian L. A possible alternative to blindly guessing S would be to consider τ to always live in $\mathbb{R}^N$ and enforce sparsity through a regularizer $\lambda\|\tau\|_1$. This will cause problems in the scalability of the algorithm. If $S = N$ then computational bottlenecks caused by the computation of Lipschitz constants $\mathcal{O}(S^2N^3)$—see [31] subsection IV.C—which we were considering $\mathcal{O}(N^3)$ now become $\mathcal{O}(N^5)$. It should be explored whether the sparsity of τ can be exploited to reduce the complexity to $\mathcal{O}(N^3)$ or not, but even if we manage to implement it efficiently, the value of the regularizer λ becomes, together with the value of α and β—and the ignored γ in the PALM algorithm [5] which we take as 1.1—another issue for us which we do not how to deal with when N and M change.

Applying the algorithm to the Metro system. As said in Chapter 1 Section 1.2, following the experiments mentioned in Section 1.1, our main motivation behind the thesis was to follow up the theoretical and experimental digress with a real-life application to Madrid's underground system. We have managed to establish contact with the Instituto Nacional de Estadística (INE) and Metro de Madrid to obtain the necessary data for our study.

We plan on conducting two experiments: one by imposing the network topology —as in subsection 2.2—and another without imposing it. By comparing the results of these experiments, we aim to understand up to what point the diffusion in the Metro is physically driven. If it is, then most people taking the Metro on a station will end up in a nearby station, which one could accomplish by walking. If not, then there will be connections crossing the whole map. And we believe that it is this last case as people take the Metro because distances are large. One of our multiple objectives is to use the algorithm to learn which is this minimal distance that makes people take the Metro.

Figure 34: Section of Metro de Madrid's map. Many people go from Plaza España to Moncloa and back. Not a lot of people go from these stations to Ventura Rodríguez, as you can comfortably take a walk. How does this reflect in the learned graph? There should be a strong weight between Moncloa and Plaza España, but if we impose the topology, we disable this connection. Does this translate in strong connections overall in the Línea 3 (yellow line)?.

We believe plenty of revealing questions can be asked from this future experiment, not only of great theoretical interest but of industrial interest. Many establishments would benefit from knowing which are the mobility trends at each time of the day and how they change over time. Even the Metro's planning would be benefited from that knowledge so as to adapt and be more efficient. This will be all seen once we get the data.

With this last idea, we finalize the thesis. We humbly hope that something written here was insightful to the reader.

Bibliography

[1] Juan A Acebrón. A Monte Carlo method for computing the action of a matrix exponential on a vector. *Applied Mathematics and Computation*, 362, 2019. 7, 45

[2] Juan A Acebrón, José R Herrero, and José Monteiro. A highly parallel algorithm for computing the action of a matrix exponential on a vector based on a multilevel Monte Carlo method. *Computers & Mathematics with Applications*, 79(12):3495–3515, 2020. 7, 45

[3] David J Bartholomew. *Latent variable models and factor analysis*. Kendall's library of statistics ; 7. Arnold, London, 2nd ed. edition, 1999. ISBN 034069243X. 15

[4] Ulrich Bauer. Ripser: efficient computation of Vietoris-Rips persistence barcodes. *J. Appl. Comput. Topol.*, 5(3):391–423, 2021. ISSN 2367-1726. doi: 10.1007/s41468-021-00071-5. URL https://doi.org/10.1007/s41468-021-00071-5. 31

[5] Jérôme Bolte, Shoham Sabach, and Marc Teboulle. Proximal alternating linearized minimization for nonconvex and nonsmooth problems. *Mathematical Programming*, 146 (1-2):459–494, 2014. 17, 44, 46

[6] Fenxiao Chen, Yun-Cheng Wang, Bin Wang, and C-C Jay Kuo. Graph representation learning: a survey. *APSIPA Transactions on Signal and Information Processing*, 9:e15, 2020. 44

[7] Fan R.K Chung. *Spectral graph theory*. Regional conference series in mathematics ; 92. Published for the Conference Board of the Mathematical Sciences by the American Mathematical Society, Providence (RI), 1997. ISBN 0821803158. 3, 9, 11, 44

[8] José Vinícius de Miranda Cardoso and Daniel P. Palomar. Learning Undirected Graphs in Financial Markets, 2020. URL https://arxiv.org/abs/2005.09958. 3, 4

[9] Xiaowen Dong, Dorina Thanou, Pascal Frossard, and Pierre Vandergheynst. Learning Graphs from Signal Observations under Smoothness Prior. *CoRR*, abs/1406.7842, 2014. URL http://arxiv.org/abs/1406.7842. 5, 13, 14, 15, 18, 26

[10] Xiaowen Dong, Dorina Thanou, Michael G. Rabbat, and Pascal Frossard. Learning Graphs from Data: A Signal Representation Perspective. *CoRR*, abs/1806.00848, 2018. URL http://arxiv.org/abs/1806.00848. 4, 14, 44

[11] Paul Erdős and Alfréd Rényi. On a classical problem of probability theory. *Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei*, 6:215–220, 1961. URL http://www.renyi.hu/~p_erdos/1961-09.pdf. 25

[12] Rubén A Hidalgo and Mauricio Godoy Molina. Navier–Stokes equations on weighted graphs. *Complex Analysis and Operator Theory*, 4(3):525–540, 2010. 44

[13] Nicholas J. Higham. *Functions of Matrices*. Society for Industrial and Applied Mathematics, 2008. doi: 10.1137/1.9780898717778. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898717778. 7, 13, 40

[14] Weiyu Huang, Leah Goldsberry, Nicholas F Wymbs, Scott T Grafton, Danielle S Bassett, and Alejandro Ribeiro. Graph frequency analysis of brain signals. *IEEE Journal of Selected Topics in Signal Processing*, 10(7):1189–1203, 2016. 5

[15] Vassilis Kalofolias. How to learn a graph from smooth signals, 2016. URL https://doi.org/10.48550/arXiv.1601.02513. 5, 9, 13, 14, 17, 28

[16] Brenden Lake and Joshua Tenenbaum. Discovering structure by learning sparse graphs. 2010. URL http://hdl.handle.net/1721.1/112759. 14

[17] Xianming Liu, Gene Cheung, Xiaolin Wu, and Debin Zhao. Random Walk Graph Laplacian-Based Smoothness Prior for Soft Decoding of JPEG Images. *IEEE Transactions on Image Processing*, 26(2):509–524, 2017. doi: 10.1109/tip.2016.2627807. URL https://doi.org/10.1109%2Ftip.2016.2627807. 4

[18] Gonzalo Mateos, Santiago Segarra, Antonio G Marques, and Alejandro Ribeiro. Connecting the dots: Identifying network structure via graph signal processing. *IEEE Signal Processing Magazine*, 36(3):16–43, 2019. 2, 4, 44

[19] K. Nodop, R. Connolly, and F. Girardi. The field campaigns of the European Tracer Experiment (ETEX): overview and results. *Atmospheric Environment*, 32(24):4095–4108, 1998. ISSN 1352-2310. doi: https://doi.org/10.1016/S1352-2310(98)00190-3. URL https://www.sciencedirect.com/science/article/pii/S1352231098001903. 6

[20] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José M. F. Moura, and Pierre Vandergheynst. Graph Signal Processing: Overview, Challenges, and Applications. *Proceedings of the IEEE*, 106(5):808–828, 2018. doi: 10.1109/JPROC.2018.2820126. 4, 5, 12

[21] Jiahao Pang and Gene Cheung. Graph Laplacian Regularization for Image Denoising: Analysis in the Continuous Domain. *IEEE Transactions on Image Processing*, 26(4):1770–1785, 2017. doi: 10.1109/tip.2017.2651400. URL https://doi.org/10.1109%2Ftip.2017.2651400. 4

[22] Christopher Rackauckas, Yingbo Ma, Vaibhav Dixit, Xingjian Guo, Mike Innes, Jarrett Revels, Joakim Nyberg, and Vijay Ivaturi. A Comparison of Automatic Differentiation and Continuous Sensitivity Analysis for Derivatives of Differential Equation Solutions. *CoRR*, abs/1812.01892, 2018. URL http://arxiv.org/abs/1812.01892. 45

[23] J. Revels, M. Lubin, and T. Papamarkou. Forward-Mode Automatic Differentiation in Julia. *arXiv:1607.07892 [cs.MS]*, 2016. URL https://arxiv.org/abs/1607.07892. 45

[24] Peter Richtárik and Martin Takáč. Parallel Coordinate Descent Methods for Big Data Optimization, 2013. URL https://doi.org/10.48550/arXiv.1212.0873. 45

[25] Aliaksei Sandryhaila and José M. F. Moura. Discrete Signal Processing on Graphs. *CoRR*, abs/1210.4752, 2012. URL http://arxiv.org/abs/1210.4752. 12, 44

[26] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge, 2008. 25, 26

[27] Santiago Segarra, Antonio G. Marques, Gonzalo Mateos, and Alejandro Ribeiro. Network Topology Inference from Spectral Templates. *CoRR*, abs/1608.03008, 2016. URL http://arxiv.org/abs/1608.03008. 2, 5, 9, 13, 14, 15, 16, 24, 26, 45

[28] 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. *IEEE signal processing magazine*, 30(3):83–98, 2013. 2

[29] Primoz Skraba and Mikael Vejdemo-Johansson. Persistence modules: Algebra and algorithms. *CoRR*, abs/1302.2015, 2013. URL http://arxiv.org/abs/1302.2015. 31

[30] Felice T Sun, Lee M Miller, and Mark D'Esposito. Measuring interregional functional connectivity using coherence and partial coherence analyses of fMRI data. *NeuroImage*, 21 (2):647–658, 2004. ISSN 1053-8119. doi: https://doi.org/10.1016/j.neuroimage.2003.09.056. URL https://www.sciencedirect.com/science/article/pii/S1053811903006062. 5

[31] Dorina Thanou, Xiaowen Dong, Daniel Kressner, and Pascal Frossard. Learning heat diffusion graphs. *IEEE Transactions on Signal and Information Processing over Networks*, 3(3):484–499, 2017. ii, 5, 6, 7, 14, 15, 17, 18, 19, 20, 21, 23, 24, 26, 27, 28, 44, 45, 46

[32] Feng Xia, Ke Sun, Shuo Yu, Abdul Aziz, Liangtian Wan, Shirui Pan, and Huan Liu. Graph learning: A survey. *IEEE Transactions on Artificial Intelligence*, 2(2):109–127, 2021. 2, 13, 44

[33] Jin Zeng, Jiahao Pang, Wenxiu Sun, and Gene Cheung. Deep Graph Laplacian Regularization for Robust Denoising of Real Images, 2019. URL https://arxiv.org/abs/1807.11637. 4

[34] Fan Zhang and Edwin R Hancock. Graph spectral image smoothing using the heat kernel. *Pattern Recognition*, 41(11):3328–3342, 2008. 4, 5

[35] Xuan Zhang, Xiaowen Dong, and Pascal Frossard. Learning of structured graph dictionaries. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3373–3376, 2012. doi: 10.1109/ICASSP.2012.6288639. 5, 14, 15

[36] Xiaofan Zhu and Michael Rabbat. Graph spectral compressed sensing for sensor networks. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2865–2868, 2012. doi: 10.1109/ICASSP.2012.6288515. 5