2 The limits of the algorithm

2.1 The number of signals needed to uniquely determine a graph

Let's start exploring the limiations of the algorithm through simple examples. Perhaps the most simple one is a path graph P_{21} with a delta $\delta_{10}(i)$ in the middle diffused ten units of time.

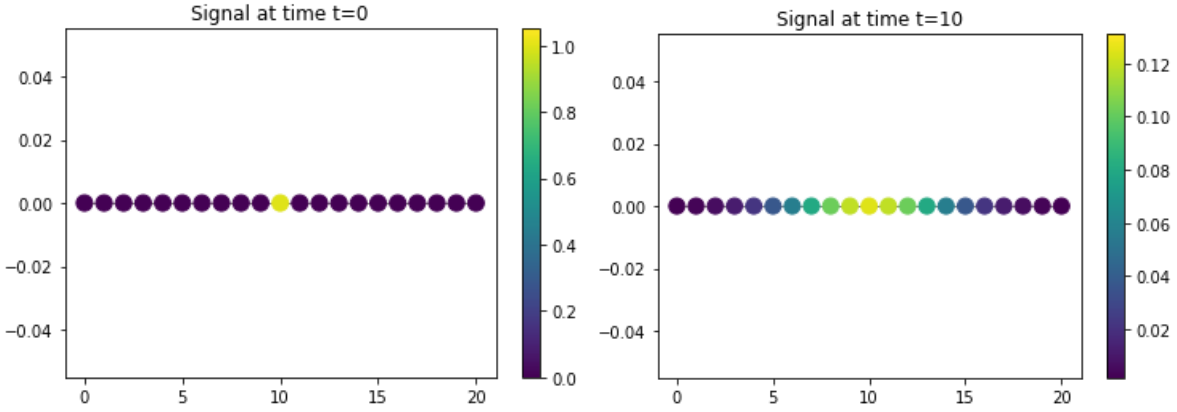


Figure 1: Timestamps of signal $\delta_{10}(i)$

Feeding the signal on the right to the algorithm leads to the following learned graphs.

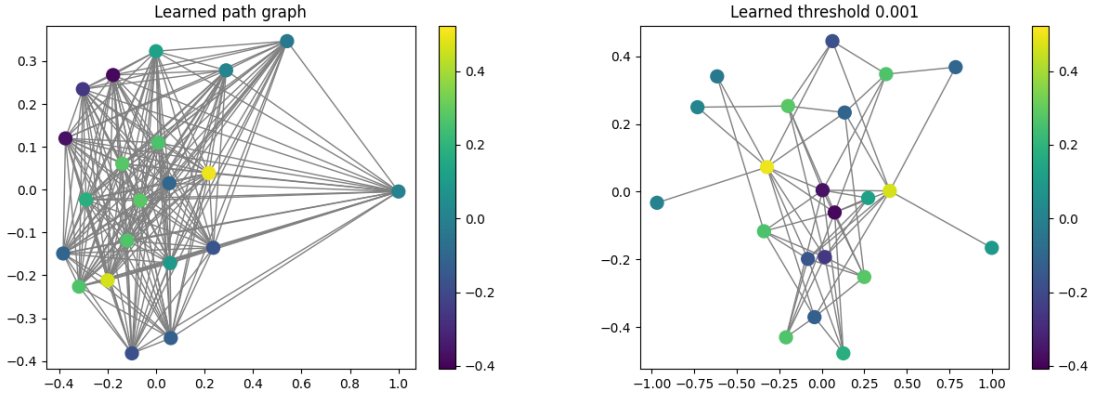


Figure 2: Learned L and H with and without threshold

To the left, we observe that the learned graph without a threshold is just the complete graph, this issue will be prevalent throughout the whole thesis. An exploration of this problem will be done in the next section, for now, the reader can keep in mind that the `LearnHeat` algorithm doesn't drag the weights to exactly floating point 0. To the right, we threshold the weights by deleting all $w_{ij} < 0.001$. There is a significant number of edges that disappear, but the graph is still far from being, as a topologist would say, nullhomotopic (a path graph). This second issue is caused by the lack of signals, as there are plenty of graphs and initial sparse signals that can be diffused to the vector which we observe in Figure 1 and furthermore, the optimization

problem is not jointly convex, therefore the algorithm will go to the first local minima that it encounters, which can be far from the real solution if there is not enough signals.

Our last attempt to retrieve the real topology is to make use of prior information, as it is done in Thanou et al. [2017] to obtain their binary classification results. In Figure 3, we have deleted all but the $|E|$ most weighted edges, where $|E|$ will from now on denote the number of edges of the groundtruth graph.

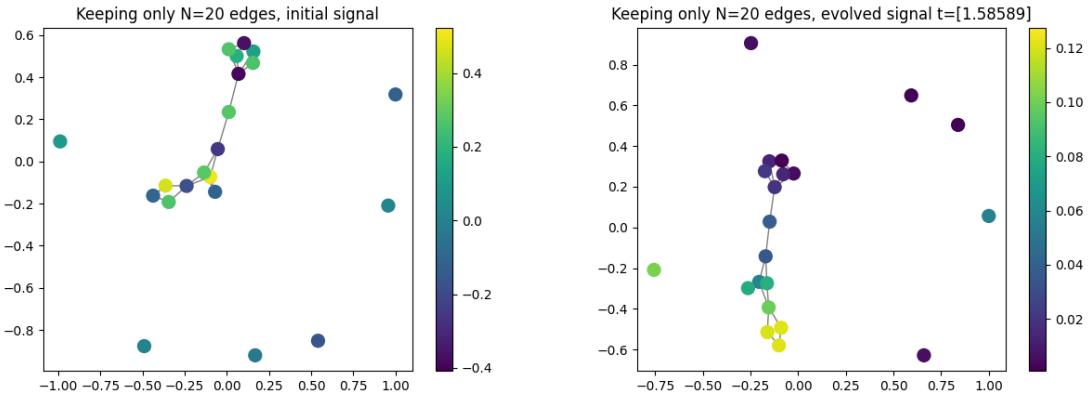


Figure 3: Learned path graph with an optimal threshold

Even by applying this threshold, the algorithm still fails, as it is underdetermined. In Thanou et al. [2017] experiments are done with up to $M = 20000$ signals to obtain excellent results, even with noise over the signals. We will not follow this line of thought, instead, we will explore the difficulties of emulating those results without any prior information from the groundtruth graph whatsoever.

The number of signals needed to determine a graph

This is a crucial part of the thesis, it marks our first deviation from the original paper and, as such, there needs to be some sort of justification to it. In Thanou et al. [2017], 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. 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?

The best-case scenario turns out to be $M = N \cdot S$ signals. As $\mathcal{D} = [e^{-\tau_s L}]_{s=1}^{S}$ is of size $N \times NS$, to be able to retrieve each entry $\mathcal{D}_{ij}$ we need H to be an $NS \times NS$ full-rank matrix. If $H = \mathbf{I}_{NS}$, the $NS \times NS$ identity, then each X_{ij} is exactly $\mathcal{D}_{ij}$, if H is not the identity then we can retrieve

$\mathcal{D}$ by taking XH^{-1}. Having found $\mathcal{D}$ then taking the matrix log over the S blocks of size $N \times N$, we obtain $-\tau_s L$ for $s = 1, \ldots, S$.

Do we need as much as 20000 signals to test the algorithm properly?

The answer is no for two reasons. The first one is that even them, that take random sparse H_i as explained before, get great results (around 0.95 F-measure score) for as little as $M = 32$ signals. This is where they claim that the strength of the algorithm lies on, in that, compared to other state-of-the-art algorithms, theirs requires a small sized training set to start being effective. They still use as much as 20000 signals in order to obtain even better results and to compensate for the noise that they add to X. The second reason is that we are not going to randomly choose the signals, but to use the minimum signals needed.

Diffusing the deltas

In all our experiments we will use the approach mentioned earlier, where we take the matrix H to be the identity $\mathbf{I}_{NS}$. Let's try to visualize what are we feeding to the algorithm. Recall the first example, the path graph. Now, instead of only feeding one delta $e^{-\tau L}\delta_i$ (where δ_i is the i-th vector of the canonical basis), we will feed all evolved deltas through each diffusion rate.

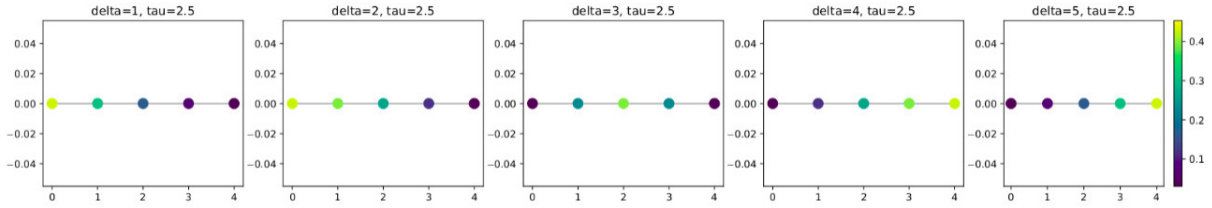
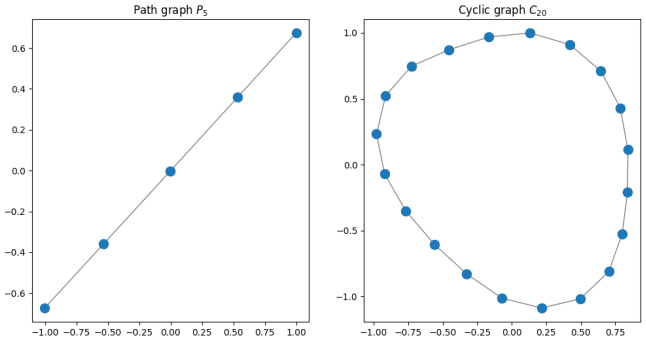


Figure 4: $X_{5+i} = e^{-2.5L}\delta_i$ for $i = 1, \ldots, 5$

We wont restrict ourselves to just the path graph. In the following we create another instance of a very simple graph, the cyclic graph, and two random instances of well-known random graphs: the Erdős–Rényi and the Barabási–Albert graph models.



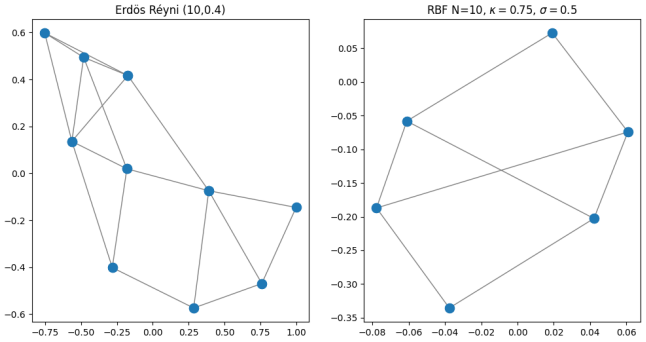


Figure 5: Four ground graphs

Then we diffuse the deltas and run the `LearnHeat` algorithm over the synthesised signals to retrieve the following graphs. If the reader recalls, as we are working in the continuum, **the learned graphs are over-connected more often than not,** therefore we establish a numerical threshold for now just for visualization purposes.

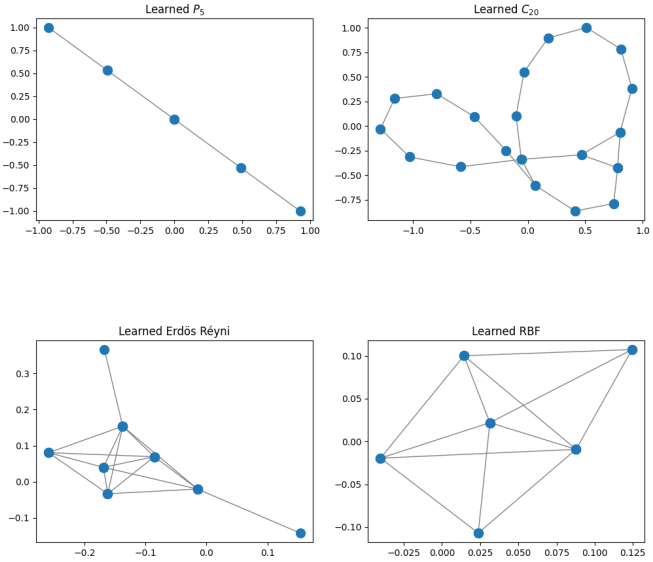


Figure 6: Four learned graphs

Immediately four issues arise:

- **Issue 1:** Were we not using the diffused deltas, how many randomly chosen sparse signals are expected to be needed in order to recover the whole graph?
- **Issue 2:** Regarding the continuous nature of the algorithm: When do we decide when two vertices are connected or not?
- **Issue 3:** How do we measure the quality of the learned graphs?

- **Issue 4:** How do the regularization parameters affect the retrieval of the graph?

These issues are ordered from least to most difficult. Giving a holistic proper answer to **Issue 4** is beyond the scope of this work. To answer **Issue 3** briefly, we refer to the reader to the subsection 1.4 of Appendix 1. In the thesis, we will answer **Issue 3** jointly with **Issue 2**.

Expected number of fully-characterizing signals (Issue 1)

Where the signals not sparse, then the probability of having a full rank H matrix after 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 Thanou et al. [2017], 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). The expected signals needed to have a rank N matrix H would be

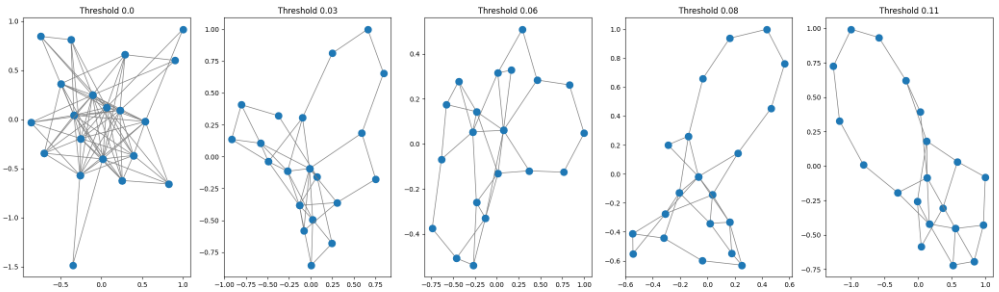
$$\mathbb{E}[M] = nH_n \approx n \log n$$

where $H_n = \sum_{i=1}^{n} \frac{1}{i}$, the n-th harmonic number. This is an interesting subject on its own in which Paul Erdős and Alfréd Rényi, hungarian mathematicians that name a graph that we will use in the experiments, have a paper (see Erdős and Rényi [1961]). 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.

2.2 The challenge of choosing a threshold (Issue 3)

If we think that binary classification is the way of assesing the quality of a graph, a natural issue arises: **How do we decide when the weight of an edge is sufficiently small to be considered** 0**?** Going back to the previous example of a learned cyclic graph, for different thresholding levels, that means, for $t_i \in [0, 1], i = 1, \ldots, n$, we threshold the graph and plot it for different values of the threshold.

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



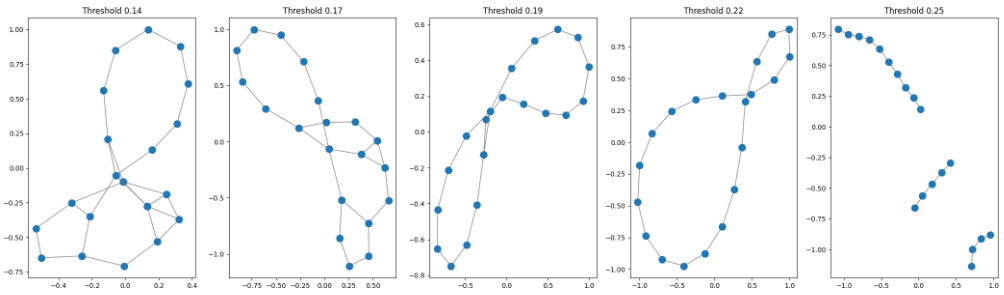


Figure 7: Different thresholds for the learned ring

The ninth snapshot seems to be the correct one, being the exact topology of the groundtruth graph. Nevertheless, we do not know that, to our knowledge, it could be any graph —*except the last one, as we would probably get runtime errors in the gradient of the Laplacian as we will see later.* **How do the authors get such good scores?**

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 8: Scores obtained by the authors of the paper

It turns out that, in order to get such pristine F-measures, they keep only the $|E|$ most weighted edges where, as we said, $|E|$ is the number of edges of the groundtruth graph they are comparing against. This is lightly mentioned in the paper but we believe **it should be written in bold as a disclaimer**, mainly because that means that scores at Figure 8 only apply to graphs for which we already know the groundtruth number of edges, in which case we will probably never apply the algorithm for binary classification in the first place.

Therefore, even if they are reassuring —if we didn't have good F-measures even after using the groundtruth information it would be a catastrophe— we will be skeptical of them, and, until further explanation from the authors, we will not use groundtruth information to modify the learned graphs.

To further illustrate the issue: for every threshold there is two values of precision and recall, therefore, the parametrization of the threshold defines a curve in the plane, which is called, in this case, the "2-class Precision-Recall curve". There is a great ipynb on the subject by the people at scikit-learn, see this page and Pedregosa et al. [2011]. This curve contains a good amount of information to visually judge whether an algorithm is performing good or bad. In the following figure, we plot, for different instances of random graphs and different regularization parameters, their learned-graphs' precision-recall curves.

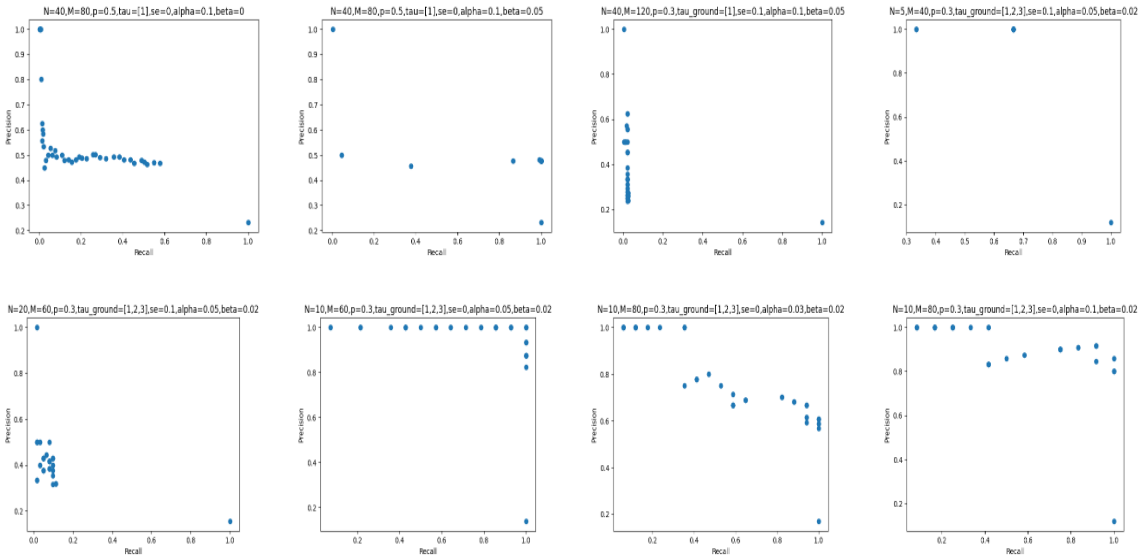


Figure 9: Different precision-recall curves for different parameters

As the reader can observe, here we find the **second disclaimer**, although this is clearly stated in the paper. **The performance of the algorithm is greatly affected by the choice of α and β,** and there is apparently no way of knowing beforehand which choice is the optimal one. In Thanou et al. [2017] they show a surface plot of the raw F-measure scores (we don't know whether they used the same filter as in Figure 8) against α and β and, as we observed in our own experiments, the results are horrible except for some special α and β combinations, even with an absurd amount of signals M, **therefore they pick the best performant combination for their scores** which is, again, using non-accessible groundtruth information to asses performance. The reader can imagine that, without groundtruth information, we would have to choose from a set of learned graphs without access to a similar figure to Figure 9 where we would be able to see which α and β works better and which optimal threshold to choose. **We need systematic criteria to process learned graphs.**

Perfect thresholding is an intractable problem

From now on, we will set $\alpha = 0.01$ and $\beta = 0.1$, the best combination found in Thanou et al. [2017] to the graphs that they use. We will not attempt to calibrate this parameters, as it is far beyond our computational capabilities to study it in a rigorous fashion. Let's try to grasp the magnitude of the problem with a diagram.

$$(L_{ground}, L_{rand}) \xrightarrow{H,\tau} (X, L_{rand}) \xrightarrow{\texttt{LearnHeat}} L_{learned}$$

The synthesis-learning-algorithm can be though of as a random non-continuous map from the set of feasible Laplacians to itself. The L_{rand} parameter is greatly affecting to which local minima will the algorithm converge. We believe that reverting the algorithm in a meaningful way is beyond our capabilities. Nevertheless, we will try to, at least, offer a solution which might work better than random, and for that we need an assumption over the groundtruth graphs and the algorithm intricacies.

Our main assumption is that non-negative weights gather around an accumulation point, both in real life and in the algorithm. We think that it is fair to assume some minimal connectivity over the networks which might be preserved through the proximal operators in the algorithm. For all the normalized combinatorial graphs this is true, all weights weight the same: $\frac{N}{2|E|}$, so we expect the algorithm to conserve an accumulation point near that value. For RBF graphs, the same happens, as we deprecate all points that lie at a distance further than κ, then there is a subtle lower bound around $\exp(-\kappa^2/\sigma^2)$ which we assume that is translated through the algorithm to an accumulation point.

This idea of accumulation points comes as the inverse of our main idea, which turns out to be worse than random, **persistent homology**. This tool is a central concept in Topological Data Analysis (TDA). There is no easy introduction to these tools, we refer the reader to Skraba and Vejdemo-Johansson [2013] for a flavour on the subject and to Bauer [2021] for a computational introduction, but to apply all these ideas to graphs is considerably simple. Graphs are just a the most simple case of both CW-Complexes and Simplicial Complexes. The rank of the first Betti number of a graph G is given by

$$\dim(H_1(G, \mathbb{Z})) = r = m - n + c$$

where m is the number of edges on the graph, n is the number of vertices and c is the number of connected components, which is the zero Betti number $\dim(H_0(G, \mathbb{Z})) = c$ (see this great article Wikipedia contributors [2022] for a more in depth explanation). That formula is telling us that, as long as our graph remains connected, a change in the Betti numbers is equivalent to a change in the number of edges. Then, to pick the most persistent graph over time, where time is a way of thinking of thresholding through $[0, 1]$, is to pick the weights that lie furthest from the next weight, and to do the opposite, which we call **least persistency**, is to pick the weights that are likely to be around an accumulation point from above. We do not claim that the following implementation is similar to what is known as persistent homology, it is even simpler, but, as we will see, it is better than random.

Algorithm 1 Least persistency filter

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

The implementation of the most persistency has the same structure, we only change line 18, where we pick the cluster which is preceded by the most jumps without a change (so that the topology has persisted over time). Keep in mind that the least persistent cluster is not the most persistent cluster when time is reverted, so the duality is only mnemonic, not mathematical. Let's see how the algorithm works with an example.

$$L_{learned} = \begin{pmatrix} 2.0497 & -1.0191 & -0.5121 & -0.5185 & 0 & 0 \\ -1.0191 & 1.5401 & 0 & -0.5110 & 0 & 0 \\ -0.5121 & 0 & 0.5121 & 0 & 0 & 0 \\ -0.5185 & -0.5110 & 0 & 1.0613 & -0.0153 & -0.0166 \\ 0 & 0 & 0 & -0.0153 & 0.7949 & -0.0153 \\ 0 & 0 & 0 & -0.0166 & -0.0153 & 0.0318 \end{pmatrix}$$

As we divide the interval $[0, 1]$ in 100 steps, all weights w_{ij} that belong to $[\frac{i}{100}, \frac{i+1}{100})$ are grouped together, in this case, we have two clusters of weights

$$0.0153,\ 0.0153,\ 0.0166 \in [0.01, 0.02)$$

and

$$0.5110,\ 0.5121,\ 0.5185 \in [0.51, 0.52)$$

candidates for accumulation points, both clusters have the same number of edges which would be deleted if the threshold went past the clusters and, in case of tie, the cluster closer to the median wins. Therefore 0.51 is chosen instead of 0.01 as threshold and the small weights are

deleted. Where there four weights in the first cluster, they would be considered significant and not deprecated, where there four weights in $[0, 0.01)$, they would be deprecated anyways, as our algorithm always deletes the first and last clusters, as the first always contains a lot of near-floating point garbage. When we run the experiments we see a great performance from the algorithm given our expectations of impossibility to the threshold task. We do not claim that this will work in any graph, but we believe it is a good option to consider for anyone who is interested in carrying the algorithm further.

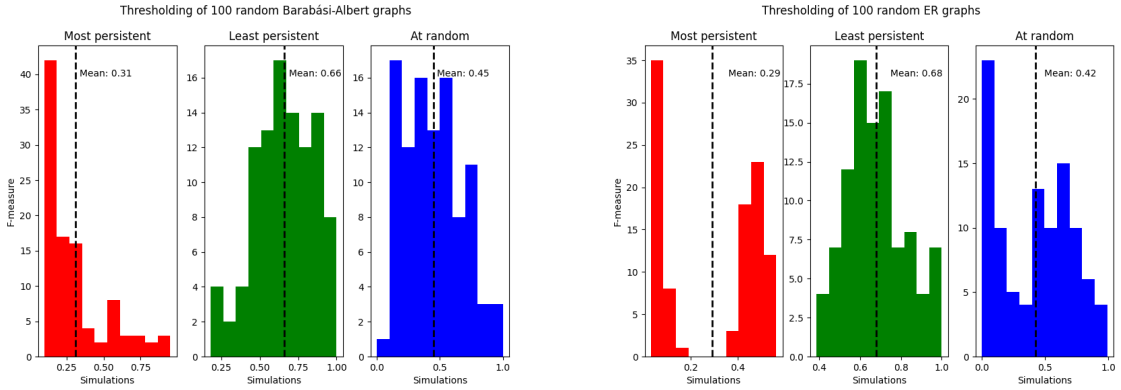


Figure 10: Barabási-Albert and Erdös-Rényi graphs, both have normalized combinatorial weights, therefore learned weights gravitate heavily towards the real weights, we expect abrupt changes in the Betti numbers to be correlated with good F-measure scores.

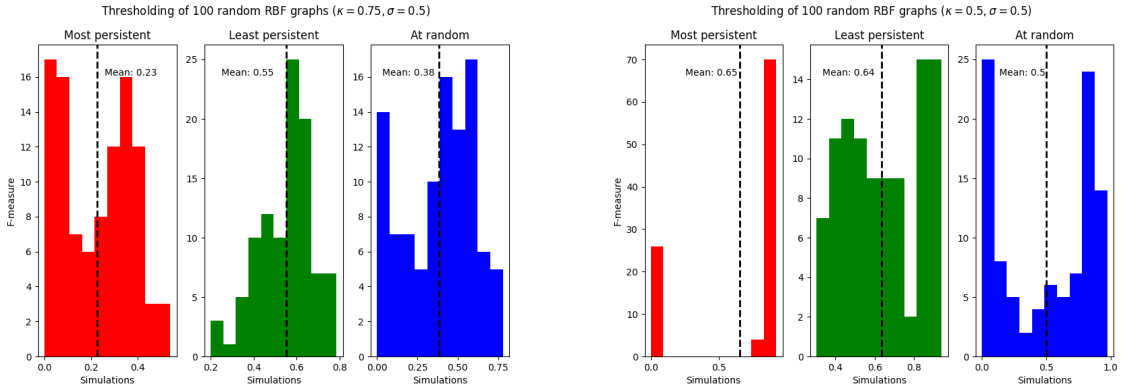


Figure 11: RBF graphs have exponentially varying weights lower bounded by $\exp(-\kappa^2/\sigma^2)$, thresholding is doable as long as the lower bound is carried through the algorithm. The right graph is much sparser than the left graph and the lower bound is rather small, therefore it is more difficult to catch with the algorithm.

Note for reproducibility: This was done with 50 **iterations each run,** 100 **independent runs, random** τ **parameters varying from** 1 **to** 3 **in length and from** 0 **to** 4 **in value. Graphs of size** $N = 20$ **(therefore** NS **signals using the deltas). The ER and BA are implemented through networkx and pygsp versions 0.51 and 3.1**

respectively, with the default parameters. The persistency was done from 0 to 1, deprecating the extrema, and prioritizing the median, with $N = 100$.

After introducing this simple algorithm it is natural to ask for more options. One would like to build some kind of maximum likelihood estimator of the threshold given the weights. After presenting this problem to various people, the usual first suggestion was to pick the mean or some sort of quantile. Maybe the reader was thinking of something on those lines.

The β-regularizer and statistical filters to the weights

The big problem with treating weights as observations of some random variable —recall that the algorithm carries the initial random Laplacian to different local minima— is that the distribution of the learned weights is totally dependant of the choice of β: the bigger β, the denser the Laplacian. The problem is that we have no theoretical treatment for that relationship. This issue affects any median-like, mean-like treatment, as the random small weights, and even the size of the problem, will tamper with any of these statistics. Furthermore, we don't have any idea of which distribution form the natural occurring heat diffusion processes, so we don't know the law induced by the algorithm onto the set of feasible Laplacians.

We did some experiments with the mean filter and some kind of regularized $1/r$-persistency, $r > 0$, that we thought made sense, as it tries to encourage the deletion of small weight clusters. This index is computed through the non-zero weights, after 10^{-5} thresholding

$$k_r = \arg\min_i \left(W_i - \frac{\sum W_j^{1/r}}{|E|} \right)^2$$

which is used to rescale the importance of each cluster, promoting the value of clusters that lie near the index k_r

$$jumps(i) = (current - next)(1 + |i - k_r|^r)$$

Experiments show that best results are achieved for $r < 1$, which is surprising, and, more importantly, that this thresholds are probably useless.

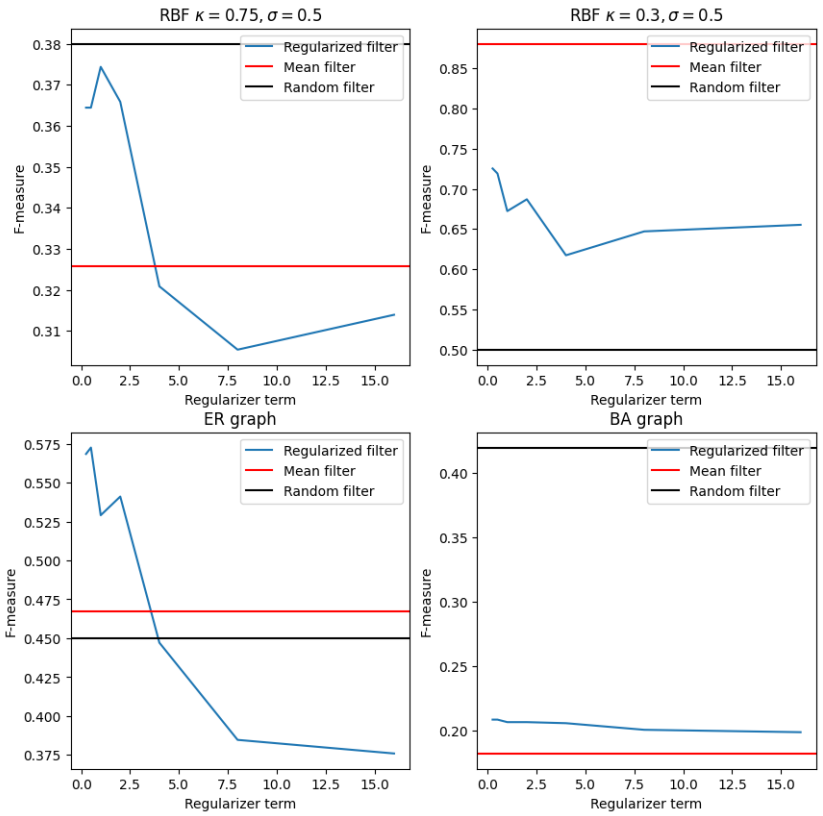


Figure 12: 100 total experiments over 4 different graphs. See how the mean filter works in "easy graphs" where there is a clear lower bound for the weights, big enough to be mistaken with outliers, and how it fails in the rest of the graphs.

No matter how we twist this approach, it won't produce results that satisfy usage. Instead of still searching a general threshold criteria, we now take a step back and explore why binary classification might not be optimal to measure the algorithm's performance.

2.3 How do we measure the quality of the learned graphs? (Issue 3)

Is the classification problem enlightening the subject? What practical information is it providing? It is indeed reassuring that we can retrieve such good F-measures by selectively filtering the edges, but in the physical context, it doesn't make a lot of sense to force continuously learned weights onto the binary $\{0, 1\}$. As always, let's try to convince ourselves with an example. Let A, B and C be three cities connected through a line. **Can the algorithm distinguish between a path graph and a continuous graph with little weight between A and C without any prior over the edges?**

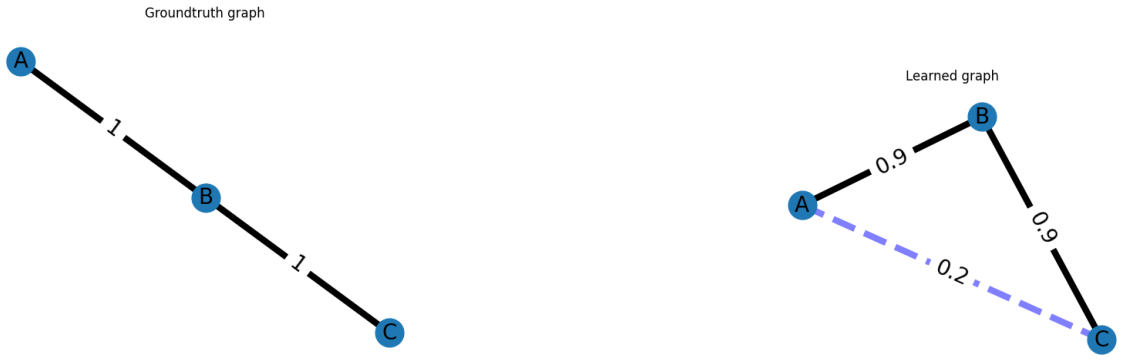


Figure 13: To the left, groundtruth graph, to the right, learned graph

It can be shown theoretically and experimentally that the algorithm can't tell apart from the left graph and the right graph if we make the small weight arbitrarily close to 0. In fact, it is embarrassingly trivial. This means that the binary classification problem is numerically ill-posed if no priors are present, **but it doesn't mean that it is mathematically ill-posed**. The delta-diffusion function that we use in our experiments

$$L \longmapsto e^{-L} = X$$

is injective over Laplacians, as they are orthogonally diagonalizable with non-negative spectrum, so instead of applying the algorithm to X we can just diagonalize the signal and take the logarithm over the spectrum

$$X \longmapsto -\log X = L$$

to theoretically retrieve the Laplacian. The numerical issue resides in taking the log of the really small $e^{-\lambda_k}$, which is not numerically stable in any way. This is the well-known issue with inverse problems of the heat equation. Nevertheless, let it be known that given an observed signal u_t and an initial signal u_0 then, if the heat kernel exists, it is unique, which means that from two signals at two different known timestamps the topology is already retrievable.

Leaving numerics aside and going back to the discrete-continuous duality: even if two graphs are topologically different, they evolve signals in equal fashion, we can even measure the MSE of the evolved deltas when deleting the k-th least weighted edge in a graph.

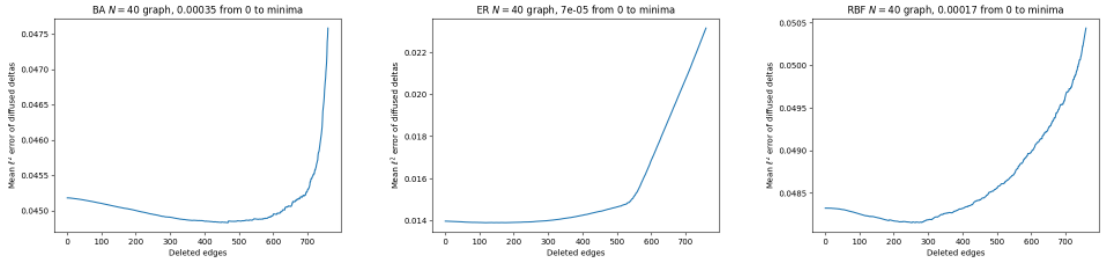


Figure 14: Change of the mean loss of diffused deltas w.r.t deleted edges

The way to read Figure 14 is that, even if we get better F-measure scores with thresholds, they can even be detrimental to the predictive capabilities of our learned Laplacian. Therefore we argue that the algorithm shouldn't be evaluated in terms of any binary classification on any synthesised dataset, but by speed of convergence, scalability, and how good does the model fit onto real world diffusion processes. However, before entering that discussion, there is a crucial idea that needs to be addressed.

2.4 The challenge of using prior knowledge at runtime

Once we have determined that prior knowledge is needed to threshold in a significant way at post-execution, why not feed the algorithm with topological information at runtime? Suppose that we are aware already of the underlying connections. Think, for example, of the railway network of your country, or the flight connections network all over the world. It is still of great interest, even if we know the existing edges (the topology), to apply the algorithm in order to reveal their weights (the geometry).

Unavoidable runtime errors when including priors as constraints

Recall that 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

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

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 $L_{learned}$. That was a problem before which now becomes unavoidable. The levels of sparsity being imposed over $L_{learned}$ are so high that in almost all simulations, even if we set $\beta = 0$, the second eigenvalue goes to 0. That leads to a runtime error caused by the L-gradient, which contains the B matrix for which

$$B_{21} = \frac{e^{\lambda_2} - e^{\lambda_1}}{\lambda_2 - \lambda_1}$$

This issue remains to be explored. For now, we have discarded three options.

- **Imposing rank**$(L) = N - 1$ **constraints** for which the feasible set would stop being a convex cone.
- **Splitting the Laplacian matrix** onto its connected components $L = \bigoplus_k L_k$ and optimize in each component, which would lead to an uncontrolled branching of Laplacians at runtime.
- **Delete edges at runtime** which also causes runtime division by zero.

And we are left with only one option

- **Delete edges at post-execution.**

which is just what the authors have been doing, for which they have, as we said, great results.