# Advances in Graph Signal Processing

## Fast graph construction & Node-adaptive graph signal reconstruction

by

Maosheng Yang

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday August 24, 2020 at 14:00 PM.


| | | |
|---|---|---|
| Student number: | 4797396 | |
| Project duration: | November 25, 2019 – August 24, 2020 | |
| Thesis committee: | Prof. dr. ir. G. Leus, | TU Delft, supervisor |
| | Dr. E. Isufi, | TU Delft, daily supervisor |
| | Prof. M. Loog, | TU Delft |

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

# PREFACE

This thesis is prepared to fulfill the requirements for obtaining a M.Sc. degree in Electrical Engineering from Delft University of Technology. The project has been carried out from November 2019 to August 2020 in the group of Circuits and Systems in the Department of Microelectronics.

This thesis focuses on the topic of data science and signal processing over graphs. Data science has covered many real-world scenarios, like recommender systems, social media platforms, and road/water/sensor networks. As the amount of data grows everyday, efficient data processing is needed. Researchers have shifted their attention to graph domain. Graph signal processing has gained the attentions of signal processing community in the last decade. It can process signals on an irregular domain, and is a promising tool to deal with the real-world high-dimensional data.

In the first part of this thesis, we aim to solve the problem of graph construction in big data scenario, which is critical for practical tasks, like collaborative filtering in recommender systems, spectral embedding or clustering in learning algorithms. We achieve to accelerate the data-driven graph construction algorithms by relying on an approximation technique for large matrix multiplication, diamond sampling. We show its potential in real problems by extensive experiments. In the second part, we improve the performance of the graph signal reconstructions by exploiting the local properties of graph signals. We propose a node-adaptive regularization with an improved degree of freedom, so a more general signal smoothness assumption is allowed. Different regularization weights design methods are proposed to achieve its best performance. By comparing it with Tikhonov regularization, we observe its superiority in graph signal reconstruction and interpolation, also in graph signal sampling.

Through-out the thesis, we involve knowledge in the fields of, statistical signal processing, estimation and detection, graph signal processing, optimization theory, and recommender systems, and spectral clustering.

I hope that this thesis is of interest to you and give an insight of my contributions to the field of data science and signal processing.

# ACKNOWLEDGMENT

At this moment, it is one more step closer for me to obtain my Master of Science degree. I would like to express my gratitude to my three beloved and respectful supervisors. Without their help, this thesis work is not able to be done well during these nine months.

First, to my daily supervisor, Elvin Isufi, who has been making every effort to guide me in research. He keeps at least one meeting per week with me, even in this difficult pandemic situations and gives very detailed and clear comments on my work, though I don't always agree. Also, he often encourages me to build my confidence and helps me in many other matters. Doing research with you is a fun trip and I'm happy to work with you in the next five years. Thank you, Elvin!

I would also like to thank Professor Geert Leus for his great supervision. From time to time, he has been commenting my work very detailed. He also supports me to make a careful decision about my future choices. From him, I see what kind of researcher I want to be in the end. It would be such a great opportunity to work with you in the future.

And Mario, as a PhD student, has been always nice to me like a good friend. Besides giving me very professional suggestions, his attitude to research also impressed me and drove me to work harder. I wish to keep work with you.

Studying in TU Delft was impossible for me without the scholarship from Microelectronics department, which is a big fortune for me not only at the moment. I would like to say, Thank you, to Microelectronics department, TU Delft.

Lastly, I want to show my love to all my friends and my family. Without your support and trust, I am not able to obtain all my achievements.

Maosheng Yang
Delft, the Netherlands
6 August 2018

# NOMENCLATURE

**Graph theory**

| | |
|---|---|
| **A** or **W** | Graph adjacency matrix |
| **B** | Graph incidence matrix |
| **L** | Graph Laplacian |
| **S** | Graph shifting operator |
| $\mathcal{G}(\mathcal{V},\mathcal{E})$ | Graph $G$ with nodes set $\mathcal{V}$ and edges set $\mathcal{E}$ |
| $\mathcal{N}_i$ | Neighbors of node $i$ |
| $\deg(i)$ | Degree of node $i$ |

**Linear algebra**

| | |
|---|---|
| $(\cdot)^\top$ | Matrix transpose |
| $(\cdot)^{-1}$ | Matrix inverse operation |
| $\lambda(\mathbf{A})$ | Eigenvalue $\lambda$ of matrix **A** |
| $\|\cdot\|_0$ | $\ell_0$ norm |
| $\|\cdot\|_1$ | $\ell_1$ norm (sum norm) |
| $\|\cdot\|_2$ | $\ell_2$ norm (Euclidean norm) |
| $\|\cdot\|_*$ | Nuclear norm of a matrix |
| $\mathbb{R}^n$ | Real vector space of real $n$-vectors |
| $\mathbf{1}$ | All-ones vector |
| $\mathbf{A} \succ \mathbf{B}$ | Matrix $\mathbf{A}-\mathbf{B}$ is positive definite |
| $\mathbf{A} \succeq \mathbf{B}$ | Matrix $\mathbf{A}-\mathbf{B}$ is positive semi-definite |
| $\mathbf{I}$ | Identity matrix |
| $\mathcal{S}_+^{n\times n}$ | Set of positive semi-definite matrices of dimensions of $n \times n$ |
| $\odot$ | Hadamard product or element-wise product |
| $\rho(\mathbf{A})$ | Spectral radius matrix **A** |

$\det(\cdot)$ Determinant of a matrix

$\mathrm{supp}(\cdot)$ Support of a matrix

$\mathrm{tr}(\cdot)$ Trace operation

**Other Symbols**

$\mathbb{E}(\cdot)$ Expectation operation of a random variable

$\mathcal{O}(\cdot)$ Order operation

$\mathrm{cov}(\cdot)$ Covariance matrix of a random variable

$\mathrm{nnz}(\cdot)$ Number of non-zeros operation

$\mathrm{sgn}(\cdot)$ Sign operation

**Mathematical objects**

$\mathbf{X}$ Matrix $\mathbf{X}$

$\mathbf{x}$ Vector $\mathbf{x}$

$\mathbf{x}_{*i}$ The $i$-th column of matrix $\mathbf{X}$

$\mathbf{x}_{i*}$ The $i$-th row of matrix $\mathbf{X}$

$x$ Scalar $x$

$x_i$ The $i$-th element of vector $\mathbf{x}$

$X_{ij}$ Entry $(i, j)$ of matrix $\mathbf{X}$

**Sets**

$\cap$ Intersection

$\cup$ Union

$\in$ Belong to

$\mathbb{C}$ Set of complex numbers

$\mathbb{R}$ Set of real numbers

$\mathbb{R}^{m\times n}$ Set of real-valued matrices of dimensions $m \times n$

$\mathcal{X}$ Set

$\subset$ Strict subset

$\subseteq$ Subset

# CONTENTS

# 1

# INTRODUCTION

As the demand of dealing with large amount of data increases rapidly, recent research has been shifted from dealing with the typical low Euclidean regular dimensional data, e.g., 1D time signals, 2D images, to irregular high dimensional data [1]. In particular, graphs are able to represent these high dimensional data because they model objects by nodes and relationships between entities by edges. These data come from practical networks like, Facebook or Twitter [2], sensor networks, transport/road networks and so on [3, 4]. On the other hand, other types of real data, like the customer's online shopping record [5], authorship attribution in online bibliography libraries [6, 7], and protein-protein interaction networks [8, 9], can also be modeled as data sit on a graph.

Graph modeling has gained more and more attention in data science, and is proven to be beneficial as well. In the machine learning community, the main tasks (semi-)supervised learning [10–12], and spectral clustering [13, 14], spectral embedding, have been carried out over networks. In deep learning, the conventional neural networks have been generalized to Graph Neural Networks (GNN) [15–21]. In the signal processing community, *Graph Signal Processing* (GSP) [22–24] has emerged, where regular domain techniques, e.g., filtering, sampling, reconstruction, and so on, have been developed over networks as graph filtering, graph signal sampling and graph signal reconstruction [25, 26]. In addition, in many real-world data science applications such as recommendation systems [27, 28], some performance improvement has been made from both GSP and GNN perspective [29–31].

However, we are still facing many challenges in these directions. For instance, efficiency needs to be improved to deal with the massive amount of data. Some graph signal processing methods can be further improved by exploiting the properties of the underlying graph. Current algorithms have scalability issues. Or in online or real-time application scenarios, the more involved dynamic adaptation is required, e.g., the underlying graph topology may change, the data sit on the networks is steaming. Many recent work has been dedicated into meeting these challenges [12, 22–24, 29].

### RESEARCH STATEMENT

In this thesis work, we focus on the following challenging situations:

- In the data science applications of recommendation systems, spectral embedding, and spectral clustering, a similarity graph is important and necessary for the further processing. However, with the rapid growth of data amount, computing or building a similarity graph is getting more and more challenging. An efficient similarity graph construction is needed in the context of big data processing.

- In graph signal processing, the current graph signal reconstruction algorithms heavily rely on the global signal smoothness prior. This is not always true, since some graph signals only vary locally. The latter prevents us from improving the reconstruction performance to the next level.

To face the above two challenges, the following two research questions are proposed:

- How to construct a similarity graph fast and efficiently when having large amount of data? This is a common and necessary question in the applications, like collaborative filtering in recommendation systems, spectral clustering, and spectral embedding. Part I of this thesis is committed to answer this question by leveraging matrix approximation techniques, which avoids directly computing the matrix multiplication by sampling.

- How can we improve graph signal reconstruction by exploiting local signal variation. In Part II of this thesis, we propose a new graph signal regularization framework to enable a more flexible signal smoothness assumption, with a significant improved performance.

### THESIS OUTLINE

- Part I: Graph construction

  - Chapter 2-Background: The goal of this chapter is to prepare the background knowledge for our approach in later chapters. We first introduce the general question of similarity graph construction and review the state-of-the-art work. Then we recall the main technique, *diamond sampling* [32], that will be used to develop our fast graph construction method. Lastly, we introduce two popular applications, namely, collaborative filtering in recommendation systems and spectral clustering.
  - Chapter 3-Diamond sampling based graph construction: This chapter contains the main results of our approach. We use diamond sampling to approximate matrix multiplications during similarity graph construction. We study its stability properties and discuss how to further improve its efficiency in some special cases.
  - Chapter 4-Numerical results and conclusion: We conclude the first part with extensive numerical results on real-world data. Then, we recall the main aspects of this part and discuss future research directions.

- Part II: Graph signal reconstruction
  - Chapter 5-Background: we review the basics of graph signal processing, mainly the graph signal reconstruction methods. In particular, the most common approach, Tikhonov regularization [33] is reviewed, as well as its bias-variance trade-off, which motivates the road map of our method.
  - Chapter 6-Node-adaptive graph signal regularization: In this chapter, we unveil our approach and propose node-adaptive graph signal regularization. We conduct the bias-variance trade-off to show its superior reconstruction performance compared with the Tikhonov regularization. Then, we study how to deign the regularization parameters in a mean squared error (MSE) sense. Finally, we conduct experiments with synthetic and real-world data.
  - Chapter 7-Node-adaptive regularization based graph signal sampling: Nodes subset selection is important when the signal observation can not be collected from all nodes; i.e., graph signal sampling, which is strongly related to graph signal reconstruction. In this chapter, we conduct the greedy subset sampling based on the node-adaptive regularization, which outperforms Tikhonov regularization based.
  - Chapter 8-Conclusions and future work: We conclude the node-adaptive graph signal regularization and propose future research directions.

# REFERENCES

[1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, *A comprehensive survey on graph neural networks,* IEEE Transactions on Neural Networks and Learning Systems (2020).

[2] M. A. Russell, *Mining the social web: Analyzing data from Facebook, Twitter, LinkedIn, and other social media sites* (" O'Reilly Media, Inc.", 2011).

[3] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, *Model-driven data acquisition in sensor networks,* in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30* (2004) pp. 588–599.

[4] K.-M. Lee, B. Min, and K.-I. Goh, *Towards real-world complexity: an introduction to multiplex networks,* The European Physical Journal B **88**, 48 (2015).

[5] M. C. Scroggie, M. E. Kacaba, D. A. Rochon, and D. M. Diamond, *System and method for providing shopping aids and incentives to customers through a computer network,* (2000), uS Patent 6,014,634.

[6] S. Segarra, M. Eisen, and A. Ribeiro, *Authorship attribution through function word adjacency networks,* IEEE Transactions on Signal Processing **63**, 5464 (2015).

[7] A. Mehri, A. H. Darooneh, and A. Shariati, *The complex networks approach for authorship attribution of books,* Physica A: Statistical Mechanics and its Applications **391**, 2429 (2012).

[8] U. Stelzl, U. Worm, M. Lalowski, C. Haenig, F. H. Brembeck, H. Goehler, M. Stroedicke, M. Zenkner, A. Schoenherr, S. Koeppen, *et al.*, *A human protein-protein interaction network: a resource for annotating the proteome,* Cell **122**, 957 (2005).

[9] J.-F. Rual, K. Venkatesan, T. Hao, T. Hirozane-Kishikawa, A. Dricot, N. Li, G. F. Berriz, F. D. Gibbons, M. Dreze, N. Ayivi-Guedehoussou, *et al.*, *Towards a proteome-scale map of the human protein–protein interaction network,* Nature **437**, 1173 (2005).

[10] X. Zhu and A. B. Goldberg, *Introduction to semi-supervised learning,* Synthesis lectures on artificial intelligence and machine learning **3**, 1 (2009).

[11] X. J. Zhu, *Semi-supervised learning literature survey*, Tech. Rep. (University of Wisconsin-Madison Department of Computer Sciences, 2005).

[12] S. Chen, F. Cerda, P. Rizzo, J. Bielak, J. H. Garrett, and J. Kovačević, *Semi-supervised multiresolution classification using adaptive graph filtering with application to indirect bridge structural health monitoring,* IEEE Transactions on Signal Processing **62**, 2879 (2014).

[13] U. Von Luxburg, *A tutorial on spectral clustering,* Statistics and computing **17**, 395 (2007).

[14] A. Y. Ng, M. I. Jordan, and Y. Weiss, *On spectral clustering: Analysis and an algorithm,* in *Advances in neural information processing systems* (2002) pp. 849–856.

[15] E. Isufi, F. Gama, and A. Ribeiro, *Edgenets: Edge varying graph neural networks,* arXiv preprint arXiv:2001.07620 (2020).

[16] F. Gama, E. Isufi, G. Leus, and A. Ribeiro, *From graph filters to graph neural networks,* IEEE Signal Processing Magazine; Special Issue on Graph Signal Processing: Foundations and Emerging Directions. (2020).

[17] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, *Convolutional networks on graphs for learning molecular fingerprints,* in *Advances in neural information processing systems* (2015) pp. 2224–2232.

[18] F. Gama, E. Isufi, G. Leus, and A. Ribeiro, *Graphs, convolutions, and neural networks,* arXiv preprint arXiv:2003.03777 (2020).

[19] P. Liang and N. Bose, *Neural network fundamentals with graphs, algorithms, and applications,* Mac Graw-Hill (1996).

[20] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, *The graph neural network model,* IEEE Transactions on Neural Networks **20**, 61 (2008).

[21] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro, *Convolutional neural network architectures for signals supported on graphs,* IEEE Transactions on Signal Processing **67**, 1034 (2018).

[22] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, *The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains,* IEEE Signal Processing Magazine **30**, 83 (2013).

[23] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, *Graph signal processing: Overview, challenges, and applications,* Proceedings of the IEEE **106**, 808 (2018).

[24] E. Isufi, *Graph-time signal processing: Filtering and sampling strategies*, Ph.D. thesis, Delft university of technology (2019).

[25] S. Chen, A. Sandryhaila, J. M. Moura, and J. Kovacevic, *Signal denoising on graphs via graph filtering,* in *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)* (IEEE, 2014) pp. 872–876.

[26] Y. Tanaka, Y. C. Eldar, A. Ortega, and G. Cheung, *Sampling on graphs: From theory to applications,* arXiv preprint arXiv:2003.03957 (2020).

[27] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, *Collaborative filtering recommender systems,* in *The adaptive web* (Springer, 2007) pp. 291–324.

[28] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, *Evaluating collaborative filtering recommender systems,* ACM Transactions on Information Systems (TOIS) **22**, 5 (2004).

[29] W. Huang, A. G. Marques, and A. R. Ribeiro, *Rating prediction via graph signal processing,* IEEE Transactions on Signal Processing **66**, 5066 (2018).

[30] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, *Session-based recommendation with graph neural networks,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33 (2019) pp. 346–353.

[31] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, *Graph neural networks for social recommendation,* in *The World Wide Web Conference* (2019) pp. 417–426.

[32] G. Ballard, T. G. Kolda, A. Pinar, and C. Seshadhri, *Diamond sampling for approximate maximum all-pairs dot-product (mad) search,* 2015 IEEE International Conference on Data Mining (2015), 10.1109/icdm.2015.46.

[33] C. Groetsch, *The theory of tikhonov regularization for fredholm equations,* 104p, Boston Pitman Publication (1984).

# I

# GRAPH CONSTRUCTION

# 2

# BACKGROUND

In many data-science and machine learning applications, e.g., spectral embedding, spectral clustering, link prediction, and collaborative filtering, a similarity graph is required before processing [1–6]. The construction of such graph is necessary because most data are collected without the underlying topology being known, unless connected in advance. As reviewed in [7], most of the network topology identification works rely on solving an optimization problem to estimate the graph Laplacian. This is challenging due to its high computational complexity, especially when the data dimension is of order up to millions of points. As the demand for big data processing emerges, improving the efficiency of the similarity graph construction is a timely challenge [8].

In the first part of this thesis, we build a similarity graph from large dimensional data in an efficient way. Before we dive into the proposed methods, we first review the similarity graph construction approaches. Then, we introduce and develop the main technique used in our method, an efficient approximation of large matrix product by sampling procedure, *diamond sampling* [9]. Lastly, we briefly review the collaborative filtering in recommender systems and spectral clustering for the experiments in the later chapters.

## 2.1. INTRODUCTION TO GRAPH CONSTRUCTION

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is made up of a set of vertices (nodes) $\mathcal{V} = \{1, \dots, N\}$ and a set of edges $\mathcal{E}$ that connect the nodes. We consider a data matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$, where $N$ is the number of nodes and $M$ the dimension of the features in data. A weight matrix $\mathbf{W}$ is assigned to the graph to measure the link strength of each edge, which is usually the pairwise relations between the nodes. The edges can be directed or undirected, $W_{ij} \neq 0$ if existing an edge between nodes $i$ and $j$, and zeros otherwise. Determining the edge set and the weights is the main task of the graph construction.

A similarity score measures the distance between features vectors sit on different nodes. An example of feature vectors is each user's preference on different movies. Popular choices are cosine similarity, Pearson's correlation, partial correlation, Hamming distance and so on [8]. In this work, we only focus on cosine similarity, which has been

widely used in text mining, information retrieval, and bioinformatics [4, 10, 11]. Cosine similarity is simple and endows with real meaning, i.e., it measures the angle difference of two vectors and it is defined as

$$C_{ij} = \frac{\mathbf{x}_i^\top \mathbf{x}_j}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2}, \tag{2.1}$$

where $\mathbf{x}_i \in \mathbb{R}^M$ is the row feature vector on node $i$. If the data matrix is normalized along the rows, i.e., the norm of each row is one, then we can compute a fully connected cosine similarity matrix as

$$\mathbf{C} = \mathbf{X}\mathbf{X}^\top, \tag{2.2}$$

where we do not differentiate the normalized data matrix and unnormalized one, and we will specify if it is normalized for clarity. The next step is to determine the edge set by sparsifying this fully connected similarity matrix $\mathbf{C}$, since it improves efficiency, interpretation, and robustness [8]. There are different ways to determine the edge set, e.g., $k$-nearest neighbors and $\epsilon$-ball neighborhood, minimum spanning tree, Gabriel graph [8]. In this work, we focus on the following two most common ones:

- $\epsilon$-ball neighborhood ($\epsilon$-N). With a predefined threshold $\epsilon$, the $\epsilon$-N method only preserves edges with weights larger than $\epsilon$. This strategy is commonly used in both brain networks and gene coexpression networks [8, 10, 12, 13]. Regarding the cosine similarity matrix $\mathbf{C}$, this implies looking for entries larger than threshold $\epsilon$; i.e., if $C_{ij} \geq \epsilon$, then edge $e_{ij}$ exits, otherwise, no. It can also be seen as searching the top-$T$ entries corresponding to the number of qualified entries, which avoids selecting the threshold. We will discuss both ways. We use operator $\epsilon(\cdot)$ to indicate the $\epsilon$-N search.

- $k$-nearest neighbors ($k$NN). For each node, the $k$NN method preserves the edges with a weight in the top-$k$ similarity scores. If a node $i$ is within the $k$NN of a node $j$, then edge $e_{ij}$ exists. Regarding the cosine similarity matrix, this implies looking for the top-$k$ entries in each vector-matrix multiplication, $\mathbf{x}_i\mathbf{X}^\top$ for each node $i$. We use k$NN(\cdot)$ to indicate the $k$NN search operation.

In both settings, a sparse graph structure is desired for learning algorithms to be efficient [8, 14]. After the edge set is determined by either the $\epsilon$-N or the $k$NN method, we can determine the edge weights directly by the similarity scores (i.e., cosine similarity here), that is, $\mathbf{W} = \epsilon(\mathbf{C})$ or $\mathbf{W} =$ k$NN(\mathbf{C})$. The weights can also be redefined by other measures, for instance, the Gaussian kernel (also called heat kernel or RBF kernel). Applying these methods is not our focus and we shall not discuss them further.

The graph construction procedure is relatively simple. First, select edges based on the similarity scores between each pair of nodes. Then, assign the weights. However, when the dimension of the data is large, the pairwise similarity score evaluation is inefficient. The cosine similarity has a computational complexity of order $\mathcal{O}(N^2M)$, therefore, it is impossible to conduct a matrix product as in (2.2). Our main goal is to construct a graph based on cosine similarity measure when the data dimension $N$ and $M$ is extremely large in an efficient way.

### 2.1.1. LITERATURE REVIEW

To build similarity graphs efficiently, the authors of [10] have proposed an efficient NN descent method with arbitrary similarity measures. The method is based on local search and has a complexity of order $\mathcal{O}(MN^{1.14})$ corroborated empirically. However, this method is only applied to small dataset upto dimension of 100 for $k$NN graph construction. Starting from dimension of 50, it starts to fail as dimensionality grows. Locality Sensitive Hashing (LSH) is another successful method, where the hash functions have been designed for a range of different similarity measures [15, 16]. However, the computational cost remains high for achieving accurate approximation. Authors in [12] proposed a L2Knng method to prune the search space using L2 norm bounds. Other methods are based on product quantization or $k-d$ generalized random forests [17]. Some recent methods [18–20] based on proximal graphs, such as small-world navigable graphs, and coarsely approximate the $k$NN graphs with various levels of computational costs. Bus most of these methods rely on high-performance computing tools, such as MapReduce or L2Knng [12]. Instead, our method stands out because it is independent of such tools. Beyond efficient $k$NN graph construction, [14] proposed a novel graph construction method through exploring the auction algorithm, which is as fast as $k$NN method. A novel sample dependent approach of graph construction is proposed in [21], but it aims to improve the parameter ($\epsilon$ or $k$) selections, which is not our focus.

In the efficient $\epsilon$-N similarity graph construction, [13] proposed TOP-DATA by leveraging the monotone property of cosine upper bound to efficiently prune item pairs without computing their exact cosine values. Later on, a max-first traversal strategy is used to further reduce the space complexity. The execution time is indeed reduced but the computation time is not, and no accuracy is evaluated through out the work. The work in [11, 22] rely on high performance computing tools, e.g., hadoop, MapReduce. The work in [11] can deal with big data but is limited to binary data only. Our proposed method can deal with general data.

## 2.2. DIAMOND SAMPLING

In this section, we introduce *diamond sampling* to approximate large matrix multiplications [9]. Consider the fundamental problem: find the top-$T$ largest entries in the product $\mathbf{C} = \mathbf{A}^\top \mathbf{B}$, given two matrices $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_m] \in \mathbb{R}^{d\times m}$ and $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n] \in \mathbb{R}^{d\times n}$. When $m = 1$, this problem reduces to the MIPS (maximum inner produce search) problem [9].

Diamond sampling is an *index sampling* method to sample pair $(i, j)$ with a probability proportional to the dot product $c_{ij} = \mathbf{a}_i \cdot \mathbf{b}_j$. With enough samples, we want to find the top-$T$ largest entries of $\mathbf{C}$. Figure 2.1 illustrates the idea. Suppose the real product of two matrices $\mathbf{A}$ and $\mathbf{B}$ is the matrix in Figure 2.1. Without actually computing this result, we expect to build an *indicator* to reflect the greatness of each entry value in the product result. For example, an ideal indicator for Figure 2.1 shows how big the entry value is by different colors; the darker the color, the bigger the value. We observe the value at entry $(1,1)$ is the largest. Then, we would just compute the vector inner product $\mathbf{a}_1^\top \mathbf{b}_1$, if only the largest entry is of interest.

Carrying this idea, we avoid computing the matrix product directly, but design the indicator to represent the matrix product value entry-wise. Therefore, we need to develop

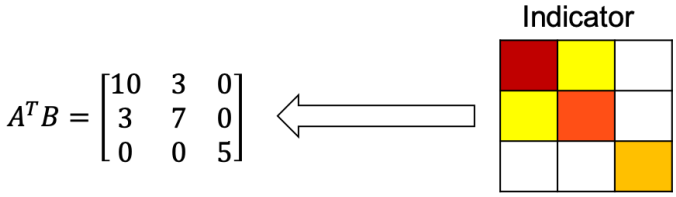



Figure 2.1: Illustration of diamond sampling to approximate the matrix product, where the colors in the indicator represent the values, the darker the bigger.

a strategy such that obtaining the indicator takes less computational effort compared with the direct inner product. Note this method only works for situations where we are interested in a small amount of top entries (i.e., the number of interest is much smaller than the total entries, $T \ll MN$), which corresponds to the preferred sparse structure in $\epsilon$-N or $k$NN graph construction.

The early work along with a similar idea is wedge sampling for MIPS problem proposed by Cohen and Lewis [23], the indicator of which has a value at entry $(i, j)$ proportional to $\mathbf{a}_i \cdot \mathbf{b}_j$. Diamond sampling can sample a pair $(i, j)$ such that the indicator is proportional to $(\mathbf{a}_i \cdot \mathbf{b}_j)^2$, requiring less samples, and computation time [9]. In the following, we follow the work in [9] to detail diamond sampling.

### 2.2.1. BINARY CASE

Let us first consider two binary matrices $\mathbf{A} \in \{0,1\}^{d\times m}$ and $\mathbf{B} \in \{0,1\}^{d\times n}$ and their product $\mathbf{A}^\top \mathbf{B} = \mathbf{C} \in \mathbb{R}^{m\times n}$. Then, we can represent the product as a tripartite graph on $m + d + n$ nodes as in Figure 2.2a. Let us further define the neighbor sets for a left node $i$, a middle node $k$ and a right node $j$

$$\mathcal{N}_i^A = \{k \in [d] | a_{ki} = 1\}, \quad \mathcal{N}_k^A = \{i \in [m] | a_{ki} = 1\},$$
$$\mathcal{N}_j^B = \{k \in [d] | b_{kj} = 1\}, \quad \mathcal{N}_k^B = \{j \in [n] | b_{kj} = 1\},$$

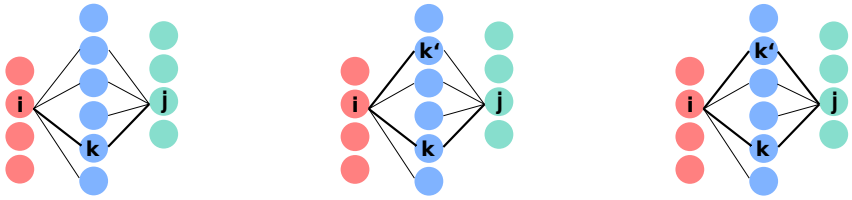
where $a_{ki}$ is the entry $(k, i)$ of matrix $\mathbf{A}$, others likewise. The node degrees can be defined as

$$\deg_i^A = |\mathcal{N}_i^A| = \|\mathbf{a}_{*i}\|_1, \quad \deg_k^A = |\mathcal{N}_k^A| = \|\mathbf{a}_{k*}\|_1,$$
$$\deg_j^B = |\mathcal{N}_j^B| = \|\mathbf{b}_{*j}\|_1, \quad \deg_k^B = |\mathcal{N}_k^B| = \|\mathbf{b}_{k*}\|_1,$$

where $\mathbf{a}_{*i}$ and $\mathbf{b}_{*j}$ are the $i$-th row vector of matrix $\mathbf{A}$ and $j$-th row vector of $\mathbf{B}$, and $\mathbf{a}_{k*}$ and $\mathbf{a}_{k*}$ are their $k$-th columns, respectively; and $\|\cdot\|_1$ is the $\ell_1$-norm.

In Figure 2.2a, the left nodes, indexed by $i$ correspond to the columns of $\mathbf{A}$; the right indexed by $j$ correspond to the columns of $\mathbf{B}$; and the middle nodes indexed by $k$, represent the common rows of $\mathbf{A}$ and $\mathbf{B}$. An edge $(i, k)$ exists if $a_{ki} = 1$. With this tripartite graph interpretation, we can see that $c_{ij}$ equals the number of common neighbors of nodes $i$ and $j$: $c_{ij} = |\{k | k \in \mathcal{N}_i^A \cap \mathcal{N}_j^B\}|$.

**Wedge sampling [23].** If a middle node $k$ has a neighbor $i$ on the left and a neighbor $j$ on the right, we call $(i, k, j)$ a "wedge". This wedge implies $c_{ij} \geq 1$. There are exactly $c_{ij}$




(a) A wedge $(i,k,j)$. (b) A three-path $(k',i,k,j)$. (c) A diamond $(k',i,k,j)$.

Figure 2.2: Binary matrix product represented as a tripartite graph, where left red color represents the column indices of **A**, middle blue color represents the common row indices shared by **A** and **B** and right green color represents the column indices of **B**. An edge linking $i$ and $k$ (or $k$ and $j$) means $a_{ki} = 1$ ($b_{kj} = 1$).

distinct wedges connecting pair $(i,j)$, see Figure 2.2a. From a statistical perspective, a wedge connecting $(i,j)$ can be selected randomly with probability proportional to $c_{ij}$.

**Diamond sampling [9].** A diamond $(k',j,k,i)$ in Figure 2.2c, formed by two wedges connecting the same endpoints $(i,j)$. If nodes $k$ and $k'$ are the same, there are $c_{ij}^2$ diamonds connecting pair $(i,j)$. Thus, a diamond $(k',j,k,i)$ for pair $(i,j)$ can be selected randomly with a probability proportional to $c_{ij}^2$. To find the largest dot products for a given sampling budget, we can first sample diamonds randomly with fixed probabilities, then count the number of diamonds connecting each pair $(i,j)$. The pair with the largest number of diamonds will likely have the largest dot product.

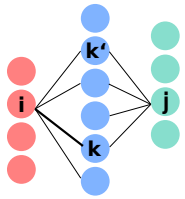
However, directly sampling a diamond is complicated. So, we first find a random three-path of the form $(k',i,k,j)$ as in Figure 2.2b. If this path is closed and forms a diamond as in Figure 2.2c, then it is counted, otherwise, not. Samples are independent from each other.

**Algorithm 1** Diamond sampling with binary inputs

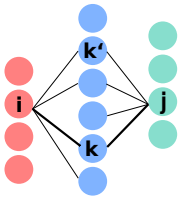
**Input:** Matrices $\mathbf{A} \in \{0,1\}^{d\times m}$ and $\mathbf{B} \in \{0,1\}^{d\times n}$, number of samples $s$, all zeros matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$.
**Output:** Indication matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$.

1: **for** $(k,i) \in [d] \otimes [m]$ **do**
2: $\quad w_{ki} \leftarrow \deg_i^A \deg_k^B$
3: **end for**
4: **for** $l = 1,\ldots,s$ **do**
5: $\quad$ sample $(k,i)$ with probability $w_{ki}/\|\mathbf{W}\|_1$
6: $\quad$ sample $j$ from $\mathcal{N}_k^B$
7: $\quad$ sample $k'$ from $\mathcal{N}_i^A$
8: $\quad x_{ij} \leftarrow x_{ij} + b_{k'j}$
9: **end for**
10: postprocessing (see Algorithm 2)

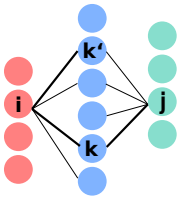
Algorithm 1 completes the above procedure. In Line 2, each edge $(i,k)$ is assigned a weight equal the number of three-paths for which $(i,j)$ is the center, i.e., $\deg_i^A \deg_k^B$. Thus, the weight matrix **W** measures the frequency of each edge being on the left side




(a) Sample a center edge $(k, i) \propto w_{ki}$.




(b) Sample a right node $j \in \mathcal{N}_k^B$.




(c) Sample the last node $k' \in \mathcal{N}_i^A$.

Figure 2.3: Diamond sampling process.

of a three-path, and it shares the same sparsity as matrix $\mathbf{A}$. In Line 5, we first sample a center edge $(k, i)$ with a probability proportional to $w_{ij}$, as shown in Figure 2.3a. Then we sample a node $k'$ and $j$ from the neighbors of $k$ in $\mathbf{A}$ and $\mathbf{B}$, i.e., $\mathcal{N}_k^B$, $\mathcal{N}_i^A$, respectively, as shown in Figures 2.3b and 2.3c. This yields a uniform random three-path. If the three-path $(k', i, k, j)$ is complete, then a diamond is sampled, i.e., $b_{k'j} = 1$, otherwise, $b_{k'j} = 0$. So, matrix $\mathbf{X}$ stores the number of diamonds connecting each pair $(i, j)$ as $x_{ij}$, i.e., the indicator. The larger entry $x_{ij}$, the larger the matrix product $c_{ij}$.

**Algorithm 2** Postprocessing

**Input:** Indices set matrix $\mathbf{X}$ for non-zero entry set, $\Omega_s = \{(i, j) | x_{ij} > 0\}$, number of top entries required $t$, and the product computation budget $t' > t$.

**Output:** An approximation of $\mathbf{C}$ by computing only the top-$t$ entries.

1: Extract the indices of the top-$t'$ entries of $\mathbf{X}$ in set $\Omega_{t'}$
2: $\mathbf{C} \leftarrow$ all zeros matrix of size $m \times n$
3: **for** $(i, j) \in \Omega_{t'}$ **do**
4: $\quad c_{ij} \leftarrow \mathbf{a}_i^\top \mathbf{b}_j$
5: **end for**
6: Extract the top-$t$ entries of $\mathbf{C}$

Furthermore, we follow Algorithm 2 to find the top-$t$ largest entries. Suppose there is a budget of $t' \geq t$ dot products. Let $\Omega_s$ be the set containing the indices of all nonzeros in $\mathbf{X}$ and $\Omega_{t'}$ the set containing the top-$t'$ entries in $\mathbf{X}$. Line 1 takes a sorting of at most $s$ items, and generally many fewer [1] ($\text{nnz}(\mathbf{X}) \ll s$). Then, we compute the top-$t'$ entries in $\mathbf{C}$. Finally, we sort these $t'$ values and extract the top-$t$ ones.

### 2.2.2. GENERAL CASE

We generalize diamond sampling for two real-valued matrices $\mathbf{A} \in \mathbb{R}^{d \times m}$ and $\mathbf{B} \in \mathbb{R}^{d \times n}$ in Algorithm 3. In this case, the entry of the weight matrix $\mathbf{W}$ is defined as

$$w_{ki} = |a_{ki}| \|\mathbf{a}_{*i}\|_1 \|\mathbf{b}_{k*}\|_1.$$

This weight measures again the frequency of each edge being on the left side to form a three-path. In the sampling procedure, the same idea is followed. First, we find a

[1] The nonzeros induced by sampling will concentrate on the entries with large values.

three-path, then check if this three-path completes a diamond. But after the center is sampled, in Lines 6 and 7, the sampling is non-uniform and more complex compared with the binary case. The postprocessing remains the same.

**Algorithm 3** Diamond sampling with general inputs

**Input:** Matrices $\mathbf{A} \in \mathbb{R}^{d\times m}$ and $\mathbf{B} \in \mathbb{R}^{d\times n}$, number of samples $s$, all zeros matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$.
**Output:** Indication matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$.

1: **for** $(k,i) \in [d] \otimes [m]$ **do**
2: $\quad w_{ki} \leftarrow |a_{ki}| \|\mathbf{a}_{*i}\|_1 \|\mathbf{b}_{k*}\|_1$
3: **end for**
4: **for** $l = 1, \ldots, s$ **do**
5: $\quad$ sample $(k,i)$ with probability $w_{ki}/\|\mathbf{W}\|_1$
6: $\quad$ sample $j$ from $\mathcal{N}_k^B$ with probability $|b_{kj}|/\|\mathbf{b}_{k*}\|_1$
7: $\quad$ sample $k'$ from $\mathcal{N}_i^A$ with probability $|a_{k'i}|/\|\mathbf{a}_{*i}\|_1$
8: $\quad x_{ij} \leftarrow x_{ij} + \mathrm{sgn}(a_{ki} b_{kj} a_{k'i}) b_{k'j}$
9: **end for**
10: postprocessing (see Algorithm 2)

- Nonnegative inputs: If $\mathbf{A}$ and $\mathbf{B}$ are nonnegative, we can omit the sign operations in Line 8. This reduces the expensive random memory accesses.
- Equal inputs (Gram matrix): If $\mathbf{B} = \mathbf{A}$, then $\mathbf{C} = \mathbf{A}^\top \mathbf{A}$ is symmetric. We consider the following step before postprocessing: $\mathbf{X} \leftarrow (\mathbf{X} + \mathbf{X}^\top)/2$. This guarantees a symmetric approximation output.

### 2.2.3. MAIN RESULTS OF DIAMOND SAMPLING

COMPLEXITY AND SPACE [9]

Let $s$ be the number of samples, $t'$ the budget of dot products in Algorithm 3, $T$ the number of largest entries to search, $\alpha = \mathrm{nnz}(\mathbf{A})$, and $\beta = \mathrm{nnz}(\mathbf{B})$. In the dense case, $\alpha = md$ and $\beta = nd$, the total complexity is of order

$$\mathcal{O}(\alpha + \beta + s\log(s\alpha\beta)).$$

The total storage (not including inputs $\mathbf{A}$ and $\mathbf{B}$) is

$$2\,\mathrm{storage}(\mathbf{A}) + \mathrm{storage}(\mathbf{B}) + 5s + 3t' + 3t.$$

The total computational cost of a direct matrix product is of order $\mathcal{O}(mnd)$, from which we can see the quadratic order of diamond sampling brings efficiency for large dimensional data.

EXPECTATION [9]

Diamond sampling is a random process. Thus, it is necessary to study its expectation and variance. For a single instance of Lines 5 to 7 in Algorithm 3, we define the event

$$\mathrm{CTP}_{k'ikj} = \text{ choosing three-path } (k', i, k, j).$$

**Lemma 1.** *The probability that event $CTP_{k'ikj}$ happens is*

$$Pr(CTP_{k'ikj}) = |a_{ki}b_{kj}a_{k'i}|/\|\mathbf{W}\|_1.$$

For the arbitrary $l$-th sample, we can define $X_{i,j,l} = \text{sgn}(a_{ki}b_{kj}a_{k'i})b_{k'j}$ and $x_{ij} = \sum_{l=1}^{s} X_{i,j,l}$.

**Lemma 2.** *The expected value of the entries of matrix* $\mathbf{X}$ *is* $\mathbb{E}[x_{ij}/s] = c_{ij}^2/\|\mathbf{W}\|_1$.

CONCENTRATION BOUNDS [9]

**Lemma 3.** *Fix $\varepsilon > 0$ and error probability $\delta \in (0,1)$. Assume all entries in* $\mathbf{A}$ *and* $\mathbf{B}$ *are nonnegative and at most $K$. If the number of samples*

$$s \geq 3K\|\mathbf{W}\|_1 \log(2/\delta)/(\varepsilon^2 c_{ij}^2),$$

*then*

$$Pr[|x_{ij}\|\mathbf{W}\|_1/s - c_{ij}^2| > \varepsilon c_{ij}^2] \leq \delta.$$

Lemma 3 provides a concentration bound for diamond sampling. When the number of samples is large, the probability of error is guaranteed. Lemmas in this section are proved in [9]. We present them here for convenience. In the following, we propose our variance study for diamond sampling.

VARIANCE STUDY OF DIAMOND SAMPLING FPR TOP-N MATRIX PRODUCT SEARCH

From Lemma 2, we see the entries of $\mathbf{X}$ are proportional in expectation to the square of the actual matrix product value. However, the expectation itself does not provide insight on stability of diamond sampling. Let us analyze the variance of $x_{ij}/s$.

**Lemma 4.** *For diamond sampling, the variance of the indicator matrix* $\mathbf{X}$ *is proportional to the inverse of square of number of samples s, i.e.,*

$$Var[x_{ij}/s] \propto \mathcal{O}(\frac{1}{s^2}). \tag{2.3}$$

*Proof.* From the definition of variance, we can write

$$\text{Var}[x_{ij}/s] = \mathbb{E}[(x_{ij}/s)^2] - \mathbb{E}^2[x_{ij}/s]. \tag{2.4}$$

We note that $x_{ij} = \sum_l X_{i,j,l}$, where $X_{i,j,l}$ is the $l$-th value of $X_{ij}$ drawn by $l$-th diamond sampling procedure and $X_{i,j,l}$ is i.i.d. for fixed $i, j$ and varying $l$. Thus, with the number of samples increasing to infinity, i.e., $s \to \infty$, we have

$$x_{ij}/s = \sum_l X_{i,j,l}/s = \mathbb{E}[X_{i,j,1}] = c_{ij}^2/\|\mathbf{W}\|_1. \tag{2.5}$$

This indicates the sample mean equals the expectation asymptotically. When the number of samples is infinity, from Lemma 2, the second term equals the first term in (2.4). Thus, the variance is zero asymptotically.

While the number of samples $s$ is finite, the term $x_{ij}/s$ will converge to $c_{ij}^2/\|\mathbf{W}\|_1$ with a linear convergence rate $\mathcal{O}(1/s)$. Thus, the variance will converge to zero with a quadratic convergence rate $\mathcal{O}(1/s^2)$. □

This variance results indicate that with a large number of samples, diamond sampling will lead to a stable approximation of the matrices product. This vanishing variance is corroborated in chapter 4.

## 2.3. COLLABORATIVE FILTERING

In recommender systems, we have user-item interactions, such as ratings or purchasing history. Users consume only a small fraction of the large library. Thus, the interactions are sparse if we see the user-item interactions as a matrix, such as in Table 2.1.

Table 2.1: An example of rating matrix, where $\mathscr{U}$ and $\mathscr{I}$ represents the user and item id.

| $\mathscr{U} \backslash \mathscr{I}$ | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| A | 5 | 3 | 1 | 4 | 2 | 5 |
| B | 1 | | 2 | | 1 | 3 |
| C | | 3 | | 5 | 1 | 5 |
| D | 2 | | 1 | 3 | | |

Collaborative filtering models these interactions to make recommendations [3, 24, 25]. Users who have the similar tastes are likely to consume similar items, i.e., similar users have similar ratings on the same item. This inference based on users similarity is called *user-based collaborative filtering* (CF). The two primary recommendation approaches of user-based CF are:

- *Rating predictions*: The task here is to predict the ratings for each user-item pair. With the observed ratings, this corresponds to a matrix completion problem. The missing or unobserved values are predicted using training model as detailed in subsection 2.3.1. Then by sorting the predicted ratings, we can make the top-$N$ recommendations for certain target users.

- *Ranking prediction*: In practice, it is not necessary to predict all missing values or estimate a specific rating score for user-item pairs. Instead, we can generate the recommendation list by a collaborative model. With a similarity matrix, we can determine the top-$k$ similar users for a target user from these neighbors preference. Furthermore, for example, we can choose a prediction function as the average ratings of the top-3 most similar users on certain item. We can also add the full rating items from the top-5 users to the recommendation list for the target user. More details can be seen in [25]. In the numerical experiments, we make recommendations by listing the highly rated movies by neighbors.

### 2.3.1. RATING PREDICTION OF USER-BASED COLLABORATIVE FILTERING

Consider a rating matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$ with $m$ users and $n$ items. If we consider the cosine similarity, the similarity score $c_{uv}$ between each user pair $(u, v)$ is computed as

$$c_{uv} = \frac{\mathbf{r}_{u*}\mathbf{r}_{v*}^{\top}}{\|\mathbf{r}_{u*}\|_2 \|\mathbf{r}_{v*}\|_2}, \tag{2.6}$$

where $\mathbf{r}_{u*}$ is the $u$-th row vector in the rating matrix. If the rating matrix $\mathbf{R}$ is normalized along the rows, the full similarity matrix can be computed by $\mathbf{C} = \mathbf{R}\mathbf{R}^{\top}$. The top-$k$ nearest neighbors for each user can be found by sorting the cosine similarity scores, then a $k$NN user-user similarity graph can be constructed by k$NN(\mathbf{C})$.

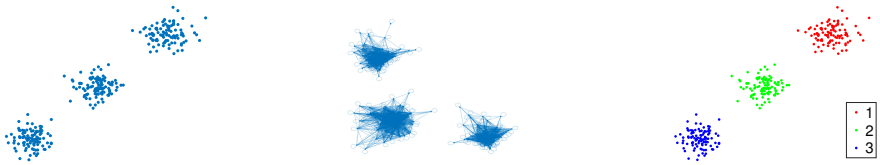
Let $P_u$ denote the set of top-$k$ similar users for user $u$. The predicted rating $\hat{r}_{uj}$ of target user $u$ for item $j$ is estimated as

$$\hat{r}_{uj} = \bar{r}_u + \frac{\sum_{v \in P_u(j)} c_{uv}(r_{vj} - \bar{r}_v)}{\sum_{v \in P_u(j)} |c_{uv}|}, \tag{2.7}$$

where $P_u(j)$ denotes the subset [2] rated item $j$ from $P_u$, $c_{uv}$ is the cosine similarity score between users $u$ and $v$, and $\bar{r}_u$ is the average rating scores of user $u$.

As it can be seen above, similarity plays an important role in CF and needs to be computed per user-user pair. This is where we will use *diamond sampling* to speed up the recommender systems.

## 2.4. SPECTRAL CLUSTERING




(a) Unclustered data. (b) Constructed similarity graph. (c) Clustered data.

Figure 2.4: Spectral clustering illustration.

Spectral clustering refers to a family of unsupervised learning algorithms, that compute a spectral embedding of the original data based on the eigenvectors of similarity graph. Spectral clustering has attracted a large attention in the last two decades due to its good performance. In general, spectral clustering comprises a similarity graph construction, a spectral embedding, and a clustering step. The procedure is detailed in Algorithm 4, which is the one what we use in the numerical experiments [2, 6, 26, 27].

A few comments are in order:

- In Line 2, the similarity matrix can be built by the strategy of $\epsilon$-N or $k$NN graph. The factor $\sigma$ in heat kernel can be user-defined or learned.
- In Line 3, usually the Laplacian choices are: combinatorial graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{W}$ [2], the normalized Laplacian $\mathbf{L}_n = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ [6], or the random walk Laplacian $\mathbf{L}_{rw} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{W}$ [26].
- In the last two lines, after the spectral embedding, the $k$-means identifies centroids by minimizing $\sum_{\mathbf{x} \in X} \min_{\mathbf{x} \in C} \|\mathbf{x} - \mathbf{c}\|^2$, which is NP-hard [28]. One can rely on approximations and heuristic solutions, such as Lloyd-Max heuristic algorithm [29].

[2] In many cases, not all $k$ similar users have ratings for certain item $j$. This is common in sparse rating matrices. In such cases, the set $P_u(j)$ will have cardinality less than $k$.

**Algorithm 4** Spectral clustering

**Input:** A set of $N$ data points $\mathbf{X} \in \mathbb{R}^{N \times M}$ in dimension $M$ and a number of desired clusters $k$.

**Output:** A partition of the $n$ points in $k$ clusters.

1: For each pair of data point $(i, j)$ in $\mathbf{X}$, compute the pairwise distance $d_{ij}$ based on a pre-selected similarity score, such as cosine similarity.
2: Compute the similarity matrix $\mathbf{W}$ based on either the radius search method or nearest neighborhood method. The weights in $\mathbf{W}$ can be the same as the similarity score $d_{ij}$ or redefined by heat kernel $W_{ij} = \exp(-d_{ij}^2/\sigma^2)$.
3: Construct the similarity graph, and compute the graph Laplacian $\mathbf{L}$, which can be (un)normalized or transformed into a random walk Laplacian.
4: Compute the first $k$ unitary eigenvectors $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k]$of the graph Laplacian corresponding to the $k$ smallest eigenvalues.
5: Embed the $i$-th node to $\mathbf{x}_i = \mathbf{U}(i,:)$, i.e., treat each row of $\mathbf{U}$ as a point.
6: Use $k$-means or $k$-medoids on $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to identify centroids $C = (\mathbf{c}_1, \ldots, \mathbf{c}_k)$.
7: Construct one cluster per centroid and assign each object $i$ to the cluster of the centroid closest to $\mathbf{x}_i$.

When dealing with large data, every step in the algorithm will be cumbersome, graph construction, spectral embedding and $k$-means. Many researchers have proposed to reduce the computational cost specifically for spectral clustering problem, e.g., Nystrom approximation, landmarks [30], coarsening, coresets, and compressive spectral clustering [27, 31]. But most of them assume the graph is constructed or coarsened. We propose *diamond sampling* to reduce the cost of similarity graph construction. Thus, it is suitable to scale spectral clustering to big data.

## REFERENCES

[1] X. J. Zhu, *Semi-supervised learning literature survey*, Tech. Rep. (University of Wisconsin-Madison Department of Computer Sciences, 2005).

[2] U. Von Luxburg, *A tutorial on spectral clustering,* Statistics and computing **17**, 395 (2007).

[3] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, *Collaborative filtering recommender systems,* in *The adaptive web* (Springer, 2007) pp. 291–324.

[4] A. Alexandrescu and K. Kirchhoff, *Data-driven graph construction for semi-supervised graph-based learning in nlp,* in *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference* (2007) pp. 204–211.

[5] X. Zhu and A. B. Goldberg, *Introduction to semi-supervised learning,* Synthesis lectures on artificial intelligence and machine learning **3**, 1 (2009).

[6] A. Y. Ng, M. I. Jordan, and Y. Weiss, *On spectral clustering: Analysis and an algorithm,* in *Advances in neural information processing systems* (2002) pp. 849–856.

[7] G. Mateos, S. Segarra, A. G. Marques, and A. Ribeiro, *Connecting the dots: Identifying network structure via graph signal processing,* IEEE Signal Processing Magazine **36**, 16 (2019).

[8] L. Qiao, L. Zhang, S. Chen, and D. Shen, *Data-driven graph construction and graph learning: A review,* Neurocomputing **312**, 336 (2018).

[9] G. Ballard, T. G. Kolda, A. Pinar, and C. Seshadhri, *Diamond sampling for approximate maximum all-pairs dot-product (mad) search,* 2015 IEEE International Conference on Data Mining (2015), 10.1109/icdm.2015.46.

[10] W. Dong, C. Moses, and K. Li, *Efficient k-nearest neighbor graph construction for generic similarity measures,* in *Proceedings of the 20th International Conference on World Wide Web*, WWW '11 (Association for Computing Machinery, New York, NY, USA, 2011) p. 577–586.

[11] W. Zhao, V. S. Martha, G. Chen, and X. Xu, *Fast information retrieval and social network mining via cosine similarity upper bound,* in *2013 International Conference on Social Computing* (IEEE, 2013) pp. 940–943.

[12] D. C. Anastasiu and G. Karypis, *L2knng: Fast exact k-nearest neighbor graph construction with l2-norm pruning,* in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management* (2015) pp. 791–800.

[13] S. Zhu, J. Wu, H. Xiong, and G. Xia, *Scaling up top-k cosine similarity search,* Data & Knowledge Engineering **70**, 60 (2011).

[14] J. Wang and Y. Xia, *Fast graph construction using auction algorithm,* arXiv preprint arXiv:1210.4917 (2012).

[15] A. Andoni and P. Indyk, *Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions,* in *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)* (IEEE, 2006) pp. 459–468.

[16] Y. Kalantidis and Y. Avrithis, *Locally optimized product quantization for approximate nearest neighbor search,* in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014) pp. 2321–2328.

[17] Y. Avrithis, I. Z. Emiris, and G. Samaras, *High-dimensional approximate nearest neighbor: kd generalized randomized forests,* arXiv preprint arXiv:1603.09596 (2016).

[18] M. Aumüller, E. Bernhardsson, and A. Faithfull, *Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms,* in *International Conference on Similarity Search and Applications* (Springer, 2017) pp. 34–49.

[19] C. Fu, C. Xiang, C. Wang, and D. Cai, *Fast approximate nearest neighbor search with the navigating spreading-out graph,* arXiv preprint arXiv:1707.00143 (2017).

[20] A. Munteanu and C. Schwiegelshohn, *Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms,* KI-Künstliche Intelligenz **32**, 37 (2018).

[21] B. Yang and S. Chen, *Sample-dependent graph construction with application to dimensionality reduction,* Neurocomputing **74**, 301 (2010).

[22] M. K. Alewiwi, *Efficient and secure document similarity search cloud utilizing mapreduce*, Ph.D. thesis (2015).

[23] E. Cohen and D. D. Lewis, *Approximating matrix multiplication for pattern recognition tasks,* Journal of Algorithms **30**, 211 (1999).

[24] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, *Evaluating collaborative filtering recommender systems,* ACM Transactions on Information Systems (TOIS) **22**, 5 (2004).

[25] C. C. Aggarwal *et al.*, *Recommender systems*, Vol. 1 (Springer, 2016).

[26] J. Shi and J. Malik, *Normalized cuts and image segmentation,* IEEE Transactions on pattern analysis and machine intelligence **22**, 888 (2000).

[27] N. Tremblay and A. Loukas, *Approximating spectral clustering via sampling: a review,* in *Sampling Techniques for Supervised or Unsupervised Tasks* (Springer, 2020) pp. 129–183.

[28] P. Drineas, A. M. Frieze, R. Kannan, S. S. Vempala, and V. Vinay, *Clustering in large graphs and matrices.* in *SODA*, Vol. 99 (Citeseer, 1999) pp. 291–299.

[29] S. Lloyd, *Least squares quantization in pcm,* IEEE transactions on information theory **28**, 129 (1982).

[30] D. Cai and X. Chen, *Large scale spectral clustering via landmark-based sparse representation,* IEEE transactions on cybernetics **45**, 1669 (2014).

[31] N. Tremblay, G. Puy, R. Gribonval, and P. Vandergheynst, *Compressive spectral clustering,* in *International Conference on Machine Learning* (2016) pp. 1002–1011.

# 3

# DIAMOND SAMPLING BASED SIMILARITY GRAPH CONSTRUCTION

Given a data matrix, our goal is to leverage the *diamond sampling* to construct a data-driven similarity graph without computing the matrix product. Thus, it is efficient and feasible for the large dimensions. In this chapter, we introduce the diamond sampling based $\epsilon$-N and $k$NN graph constructions. We also show the smooth property of the $\epsilon$-N graph by reviewing the work in [1]. Then, we study the errors in the constructed $\epsilon$-N graph. Lastly, we discuss improvements on graph construction by reviewing the *butterfly counting* task for bipartite graphs. We consider cosine similarity throughout the chapter.

## 3.1. $\epsilon$-BALL NEIGHBOR GRAPH CONSTRUCTION

The $\epsilon$-ball neighbor graph construction preserves the edges with a similarity score larger than a predefined threshold $\epsilon$ and makes the rest zero. This procedure preserves the top-$T$ (predefined, corresponding to $\epsilon$) edges in the similarity matrix. Thus, using diamond sampling is straightforward. It consists of replacing the direct matrix product $\mathbf{A}\mathbf{A}^\top$ by diamond sampling as shown in Algorithm 5.

The two input matrices to diamond sampling [cf. Algorithm 3] should be $\mathbf{A}^\top$, since we want to build a similarity matrix $\mathbf{D}$ of dimensions $M \times M$. After obtaining the indicator matrix $\mathbf{X}$, we symmetrize it as in Line 2. We also assume that the data matrix is normalized, which can be done by storing the data, otherwise, normalization requires additional efforts. After normalization, the pairwise cosine similarity measure of the data can be represented as matrix product $\mathbf{A}\mathbf{A}^\top$ as in (2.2). Lastly, we translate the edge thresholding by $\epsilon$ into a task of looking for the top-$T$ entries.

Providing an appropriate threshold parameter $\epsilon$ is usually difficult. Instead, we search top-$T$ strongly correlated node pairs [1, 2]. This is also more intuitive when processing big data and indicates the sparsity rate of the similarity matrix, i.e., $T/M^2$. For instance, in a $10^3 \times 10^3$ item-item pairwise similarity graph, there can be in total $10^6$ pairwise relations, but we search for and preserve top-5% strongly correlated ones. In the following,

**Algorithm 5** Diamond sampling based cosine similarity $\epsilon$-N graph construction.

**Input:** Data matrix $\mathbf{A} \in \mathbb{R}^{M\times N}$ (normalized in the row direction), threshold $\epsilon$, corresponding to the top-$T$, number of samples $s$.
**Output:** A similarity matrix $\mathbf{D} \in \mathbb{R}^{M\times M}$.
1: Input matrix $\mathbf{A}^\top$, number of samples $s$, and $T$ to Algorithm 3.
2: Obtain the indicator matrix $\mathbf{X}$, then $\mathbf{X} \leftarrow (\mathbf{X}+\mathbf{X}^\top)/2$.
3: Fine the indices of the top-$T$ entries in $\mathbf{X}$ as in Algorithm 2.
4: Obtain the approximate cosine similarity matrix $\mathbf{D}$ of the $\epsilon$-N graph, by extracting the top-$T$ similarity scores.

we study the properties and the accuracy of the constructed $\epsilon$-N graph.

### 3.1.1. SMOOTHNESS GUARANTEE

This section analyzes the $\epsilon$-N graph from a graph topology learning perspective based on [1]. The work in [1] learns the graph topology from data under a smoothness prior [3].

Given a data matrix $\mathbf{A} \in \mathbb{R}^{M\times N}$ , we can learn a sparse graph from a complete one, in which each node is connected to every other node with the number of edges $|\mathscr{E}| = r = M(M-1)/2$. We aim to determine a subgraph by choosing a subset of edges, $\mathscr{E}_S \subset \mathscr{E}$. Denote the graph Laplacian matrix of the complete graph as $\mathbf{L} \in \mathbb{S}^{M\times M}$, where the main diagonal entries are all $M-1$, and the off-diagonal are $-1$. The Laplacian can also be expressed in terms of the incidence matrix, $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_r] \in \mathbb{R}^{M\times r}$ as

$$\mathbf{L} = \mathbf{B}\mathbf{B}^\top = \sum_{p=1}^{r} \mathbf{b}_p \mathbf{b}_p^\top, \tag{3.1}$$

where the $p$-th column, $\mathbf{b}_p$ denotes a length-$m$ edge vector with entries $[\mathbf{b}_p]_i = 1$, and $[\mathbf{b}_p]_j = -1$, and zeros elsewhere, if an edge $p$ leaves node $j$ and enters node $i$.

Consider a subgraph with $T$ edges and $|\mathscr{E}_S| = T \ll r$. We refer to this subgraph as a $T$-sparse graph. We can connect this graph to $\mathbf{L}$ through a *sparse edge selection* vector $\boldsymbol{\omega} = [\omega_1, \omega_2, \ldots, \omega_r]^\top \in \{0,1\}^r$, where $\|\boldsymbol{\omega}\|_0 = T$, and $\omega_p = 1$ if $p \in \mathscr{E}_S$, or $\omega_p = 0$ otherwise. Finally we can write the Laplacian of this $T$-sparse graph as a function of $\boldsymbol{\omega}$, i.e.,

$$\mathbf{L}_S(\boldsymbol{\omega}) = \sum_{p=1}^{r} \omega_p \mathbf{b}_p \mathbf{b}_P^\top. \tag{3.2}$$

We can learn the graph Laplacian, i.e., the graph topology under the prior information of the data matrix $\mathbf{A}$, which can be seen as $N$ graph signals (i.e., the signal sit on this graph), has smooth variations on the resulting graph. Mathematically, we can formulate the latter as

$$\underset{\boldsymbol{\omega}\in\mathscr{W}}{\arg\min} \quad \mathrm{tr}\{\mathbf{A}^\top \mathbf{L}_S(\boldsymbol{\omega})\mathbf{A}\}, \tag{3.3}$$

where $\mathscr{W} = \{\boldsymbol{\omega} \in \{0,1\}^r | \|\boldsymbol{\omega}\|_0 = T\}$. This is because the cost in (3.3) measures the signal variation [4]. By inserting (3.2), we can express the cost function in (3.3) as a linear func-

tion in $\boldsymbol{\omega}$, i.e.,

$$\mathrm{tr}\{\mathbf{A}^\top L_S(\boldsymbol{\omega})\mathbf{A}\} = \sum_{p=1}^{r} \omega_p \mathrm{tr}\{\mathbf{A}^\top \mathbf{b}_p \mathbf{b}_p^\top \mathbf{A}\}. \tag{3.4}$$

Then, by introducing a length-$r$ vector $\mathbf{c} = [c_1, c_2, \ldots, c_r]^\top$, with $c_p = \mathrm{tr}\{\mathbf{A}^\top \mathbf{b}_p \mathbf{b}_p^\top \mathbf{A}\}$, we can rewrite the optimization problem (3.3) as

$$\underset{\boldsymbol{\omega}\in\{0,1\}^r}{\arg\min} \quad \mathbf{c}^\top \boldsymbol{\omega} \quad \text{s.to} \quad \|\boldsymbol{\omega}\|_0 = T. \tag{3.5}$$

The above Boolean linear programming problem can be solved by finding the $T$ smallest entries of $\mathbf{c}$, and the solution $\boldsymbol{\omega}$ will have entries equal to one at indices corresponding to the $T$ smallest entries of $\mathbf{c}$, and zero otherwise. We can now link this procedure with our work as follows.

**Proposition 1.** *Consider a full similarity matrix $\mathbf{C} = \mathbf{A}\mathbf{A}^\top$, with the given data $\mathbf{A} \in \mathbb{R}^{M\times N}$, where the set $[M]$ can be seen as the node set of a similarity graph. The solution to* (3.5) *selects $T$ edges between those nodes having the highest cross-correlation. We can also express the cost function in* (3.3) *as*

$$tr\{\mathbf{A}^\top \mathbf{L}_S(\boldsymbol{\omega})\mathbf{A}\} = tr\{\mathbf{L}_S(\boldsymbol{\omega})\mathbf{C}\} = \sum_{p=1}^{r} \omega_p (\mathbf{b}_p^\top \mathbf{C}\mathbf{b}_p), \tag{3.6}$$

*where $\mathbf{C}$ is the full data similarity matrix. As the definition of $\mathbf{b}_p$ in* (3.1)*, it is easy to see that $(\mathbf{b}_p^\top \mathbf{C}\mathbf{b}_p) = [\mathbf{C}]_{i,i} + [\mathbf{C}]_{j,j} - 2[\mathbf{C}]_{i,j}$ is small if the $i$-th and $j$-th nodes are most similar.*

*Proof.* See in [1, Sec.3] □

Proposition 1 describes the validity of constructing a graph by connecting nodes that are most similar, i.e., selecting the top-$T$ entries in the full similarity matrix. The work in [1] is restricted to learn the graph topology not the edge weights. The above shows that an $\epsilon$-N graph guarantees smoothness, which is preferred since the signals often vary slowly over the graph. The latter also shows our method can be seen as an alternative to build large-scale graphs for smooth signals,

### 3.1.2. ERROR ANALYSIS IN DIAMOND SAMPLING BASED $\epsilon$-N GRAPH CONSTRUCTION

In this part, we analyze the errors that could happen in the $\epsilon$-N graph built by the random diamond sampling.

**Theorem 1.** *Given a data matrix $\mathbf{A} \in \mathbb{R}^{M\times N}$, where entry $a_{ij} \in [0, K]$ for all $i, j$. Suppose the oracle $\epsilon$-ball Neighbor graph corresponds to the top-$T$ nonzero entries in the full similarity matrix $\mathbf{C} = \mathbf{A}\mathbf{A}^\top$ with entries $c_{ij}$. Denote the set of indices within the top-$T$ entries by $\mathscr{T}$ and the total set of indices by $\mathscr{C}$. With $s$ being the number of diamond sampling steps, we can approximately build an $\epsilon$-N similarity matrix $\mathbf{D}$ with $T$ nonzero entries. The number of errors (entries smaller than $\epsilon$) in the entries of $\mathbf{D}$ satisfies*

$$\#(errors\ in\ \mathbf{D}) < \exp(-2\epsilon^4 s/(K\|\mathbf{W}\|_1)^2)\cdot(M^2 - T) + \exp(-2(min(c_{ij}^2) - \epsilon^2)^2 s/(K\|\mathbf{W}\|_1)^2)\cdot T \tag{3.7}$$

*where $\mathbf{W}$ is the weight matrix computed in diamond sampling Algorithm 3.*

*Proof.* See Appendix A.1. □

From Theorem 1, the errors can happen in two parts. In the top-$T$ entries and the remaining entries. The errors will be reduced as the number of samples $s$ increases. The increase of the maximal value $K$ in $\mathbf{A}$ and the absolute sum of the weight matrix $\mathbf{W}$ will increase the error probability. Theorem 1 indicates the number of errors in the diamond sampling $\epsilon$-N similarity matrix is limited.

## 3.2. $k$NN GRAPH CONSTRUCTION

The general idea of diamond sampling $k$NN graph construction is similar to $\epsilon$-N graph with cosine similarity. The key step in $k$NN graph construction is the nearest neighbor search for each node. Given again the data matrix $\mathbf{A} \in \mathbb{R}^{M\times N}$ with cosine similarity, building a $k$NN graph implies looking for the top-$k$ entries in each vector-matrix multiplication $\mathbf{a}_i\mathbf{A}^\top$ for all $i$, where $\mathbf{a}_i$ is $i$-th row of $\mathbf{A}$. The computational cost is $\mathcal{O}(N^2)$ plus the cost of ordering.

- When dealing with big data ($N$ is large while $M$ is of medium size), this strategy is not practical and we should rely on diamond sampling.
- When $M$ is also large, even with diamond sampling by solving the exhaust of the $k$NN search per node, the loop over the large number of nodes hurdles building a $k$NN graph.

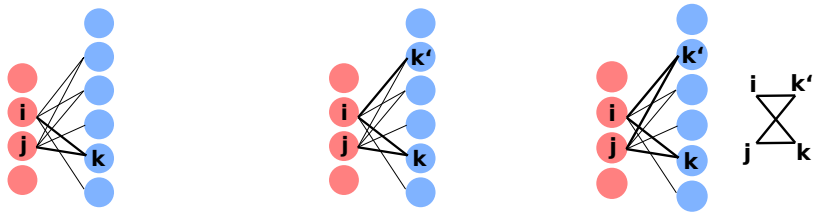
In the numerical experiments on a big dataset, we conduct the $k$NN search for a relatively small number of the total nodes, which vindicates the first case. More development of the diamond sampling based $k$NN graph construction is left for future work.

## 3.3. BUTTERFLY SAMPLING

Diamond sampling approximates the largest entries of two general matrix product. However, in the cosine similarity graph construction, we need to compute the self-product of the data matrix $\mathbf{A}$, i.e., $\mathbf{C} = \mathbf{A}\mathbf{A}^\top$. In this sense, besides storing one less big matrix, we can represent the matrix product by a bipartite graph instead of a tripartite one, which brings additional benefits. The diamond sampling approximation technique can be understood by a butterfly sampling procedure. From the illustration in Figure 3.1, the transition is rather intuitive and follows diamond sampling, so we skip the details and deliver the bipartite graph representation and algorithms directly.

### 3.3.1. BIPARTITE GRAPH REPRESENTATION OF COSINE SIMILARITY MATRIX

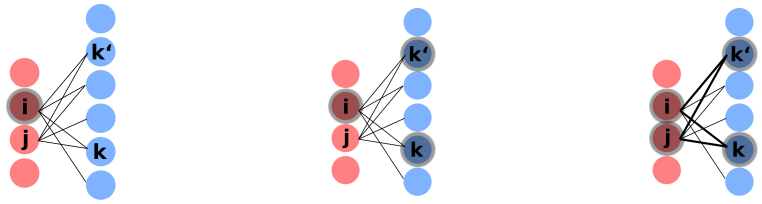
From the tripartite graph representation of the binary matrices product ($\mathbf{C} = \mathbf{A}\mathbf{B}^\top$) in Figure 2.2, the number of the wedges and diamonds connecting two nodes is proportional to the matrix product entry itself and its square, respectively. However, when the two matrices are equal, i.e., $\mathbf{B} = \mathbf{A}$, we can represent the product ($\mathbf{C} = \mathbf{A}\mathbf{A}^\top$) in a bipartite graph and the corresponding wedge, three-path and diamond, called a butterfly, are shown in Figure 3.1. The wedge, three-path, and Butterfly in a bipartite graph can be seen by mirroring the corresponding ones in a tripartite graph.




(a) Wedge $(i, k, j)$. (b) Three-path $(k', i, k, j)$. (c) Butterfly $(i, k, j, k')$.

Figure 3.1: Bipartite graph representation of the matrix product $\mathbf{AA}^\top$.

### 3.3.2. BUTTERFLY SAMPLING




(a) Sample node $i$ from the row indices of $\mathbf{A}$. (b) Sample two nodes from the neighbor of $i$ uniformly. (c) Butterfly $(i, k, j, k')$.

Figure 3.2: Butterflying sampling in a bipartite graph (the sampled objects are shadowed).

From the bipartite graph representation, we sample a butterfly from to look for the largest entries in the similarity matrix. Consider the binary case first. Algorithm 6 is a small variant of the diamond sampling in Algorithm 1, aiming at finding the indicator matrix which reveals the final product matrix ($\mathbf{C} = \mathbf{AA}^\top \in \mathbb{R}^{M\times M}$). Instead of following the exact diamond sampling procedure, i.e., "edge, three-path, diamond", we consider the node-based butterfly sampling as in Figure 3.2. First, sample one node from left side, then uniformly sample two of its neighbors from the right side, lastly, we sample the last node from left side and check if a butterfly is complete.

The butterfly sampling, in Algorithm 6, can speed up the convergence procedure for approximating the similarity graph. In Line 7, we find all potential butterflies that contain nodes $i, k, k'$, which incorporate a deterministic procedure. This is different from the diamond sampling where at each sampling procedure, one can find at most one diamond. These modifications can also be applied to the general case diamond sampling in Algorithm 3 as well. In the following, we discuss how to compute the similarity matrix exactly by counting the butterflies in the bipartite graph.

### 3.3.3. RELATION WITH EXACT BUTTERFLY COUNTING

Inspired by the butterflying counting problem in the bipartite graph [5, 6], we propose Algorithm 7 to compute the similarity graph *exactly* based on butterflying counting. The ground is on the bipartite representation of the similarity graph construction, and that each wedge connecting nodes $(i, j)$ in the bipartite graph representing an increment in the entry $c_{ij}$. The time complexity of exact butterfly counting for exact similarity ma-

**Algorithm 6** Butterfly sampling with binary input data matrix $A$, vertex-based

**Input:** Data matrix $\mathbf{A} \in \{0,1\}^{M\times N}$, number of samples $s$, all zeros matrix $\mathbf{X} \in \mathbb{R}^{M\times M}$.
**Output:** Indication matrix $\mathbf{X} \in \mathbb{R}^{M\times M}$.

**for** $i = 1,\ldots,M$, #rows in $A$ **do**
  $w_i \leftarrow \binom{d_i}{2}$, where $d_i$ is the degree of the node $i$, i.e., $\|\mathbf{a}_{i*}\|_1$, the $i$-th row sum of matrix $\mathbf{A}$
**end for**
**for** $l = 1,\ldots,s$ **do**
  sample node $i$ with probability $w_i/\|\mathbf{W}\|_1$, from the row indices of $\mathbf{A}$
  sample two distinct $k, k'$ from $\mathscr{N}_i$ uniformly at random
  extract the common neighbors of $k$ and $k'$ as set $\mathscr{J} \leftarrow \mathscr{N}_k \cap \mathscr{N}_{k'}$
  **for** $j \in \mathscr{J}$ and $j \neq i$ **do**
    $x_{ij} \leftarrow x_{ij} + 1$
  **end for**
**end for**
postprocessing (see Algorithm 2)

trix construction is not reduced, but it converts the matrix computation as a counting problem on graphs. Lastly, we note that the algorithm is for binary matrix only. As the butterfly counting approximation techniques are extended to large scale graphs [5, 6], we can apply them in the similarity graph construction problem in future.

**Algorithm 7** Exact butterfly counting for similarity matrix computation

**Input:** Binary data matrix $\mathbf{A} \in 0,1^{M\times N}$
**Output:** Similarity matrix $\mathbf{C} = \mathbf{A}\mathbf{A}^\top \in \mathbb{R}^{M\times M}$

**if** $M < N$ **then**
  set $\mathscr{I} \leftarrow \{1,\ldots,M\}$, i.e., the row indices of $\mathbf{A}$
**end if**
**for** $i \in \mathscr{I}$, $k \in \mathscr{N}_i$ and $j \in \mathscr{N}_k$ **do**
  **if** $j < i$ **then**
    $c_{ij} \leftarrow c_{ij} + 1$
  **end if**
**end for**

# REFERENCES

[1] S. P. Chepuri, S. Liu, G. Leus, and A. O. Hero, *Learning sparse graphs under smoothness prior,* in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, 2017) pp. 6508–6512.

[2] S. Zhu, J. Wu, H. Xiong, and G. Xia, *Scaling up top-k cosine similarity search,* Data & Knowledge Engineering **70**, 60 (2011).

[3] G. Mateos, S. Segarra, A. G. Marques, and A. Ribeiro, *Connecting the dots: Identifying*

*network structure via graph signal processing,* IEEE Signal Processing Magazine **36**, 16 (2019).

[4] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, *The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains,* IEEE Signal Processing Magazine **30**, 83 (2013).

[5] S.-V. Sanei-Mehri, A. E. Sariyuce, and S. Tirthapura, *Butterfly counting in bipartite networks,* in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2018) pp. 2150–2159.

[6] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang, *Efficient butterfly counting for large bipartite networks,* arXiv preprint arXiv:1812.00283 (2018).

# 4

# NUMERICAL RESULTS AND CONCLUSION

In this chapter, we corroborate diamond sampling graph construction. We consider several different baseline dataset to build the $\epsilon$-ball and $k$NN similarity graph. We also test the performance in recommender system and spectral clustering. Lastly, we draw the conclusions of Part I.

In the numerical experiments, we use the diamond sampling library [1]. It has Matlab and C implementations. The C implementations rely on the mex interface to Matlab. For comparison, Matlab and CSparse library [2] based exact similarity matrix computations are included.

## 4.1. $\epsilon$-N AND $k$NN GRAPHS

In the first experiment, we use diamond sampling to construct an $\epsilon$-N similarity graph. This graph is equivalent to find the top-$T$ entries in the full similarity graph. We compare the accuracy of an $\epsilon$-N graph constructed by diamond sampling with the exact computation. This is measured by hit rate, defined as follows

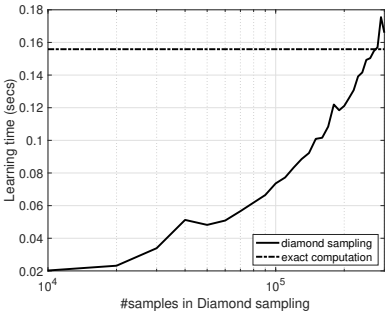
$$\text{Hit rate} = \frac{\#\text{correct entry indices found by diamond sampling}}{T}, \tag{4.1}$$

which measures how many relations are correctly found by diamond sampling. We also compare the time to learn the similarity graph with different number of samples.

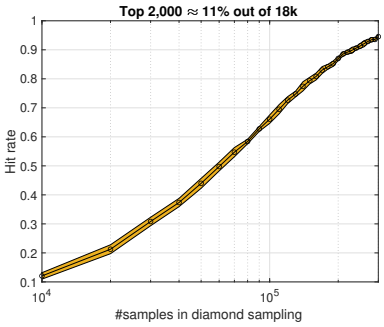
### 4.1.1. AMAZON AUTOMOTIVE REVIEWS

The first dataset is a subset from the Amazon automotive product reviews with ratings. The number of users is 2918 and items is 1835. The number of total ratings is 17545. To build an $\epsilon$-N similarity for items, we keep the top-2000 entries, which is of sparsity 11.1% in the full item-item similarity matrix.

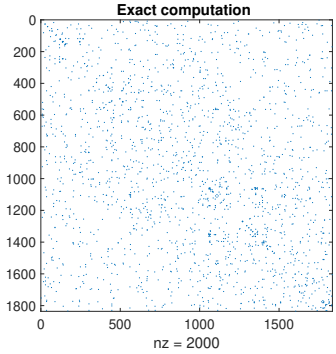
In Figure 4.1, we present the results of the similarity matrix learned from diamond sampling with respect to the exact one for different number of samples in $\{1 \times 10^4, 2 \times$



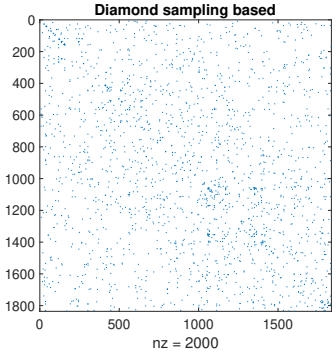

(a) Learning time w.r.t. number of samples.




(b) Hit rate of the similarity matrix learned by diamond sampling w.r.t. the exact one (The yellow shadow area is the standard deviation).




(c) Exact computation.




(d) Diamond sampling with $1.5 \times 10^5$ samples.

Figure 4.1: $\epsilon$-N graph construction performance on Amazon automotive product reviews data.

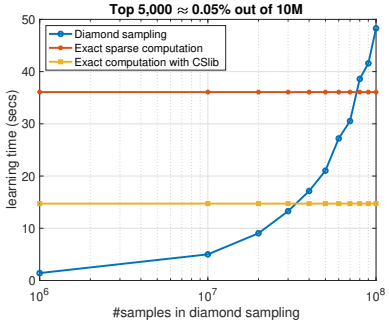
$10^4, \ldots, 30 \times 10^4\}$. From Figure 4.1a, when the number of samples is smaller than 30K, diamond sampling takes less time than direct computation. If a hit rate of 80% is required as in Figure 4.1b, diamond sampling takes 50% less time. When the number of samples increases, the standard deviation, i.e., the yellow shadow area reduces, which vindicates that the variance of the approximation results studied by Lemma 4. Lastly, in Figures 4.1c and 4.1d, we show the similarity graph sparsity pattern learned by the exact computation and diamond sampling.

### 4.1.2. MOVIELENS 10 MILLIONS

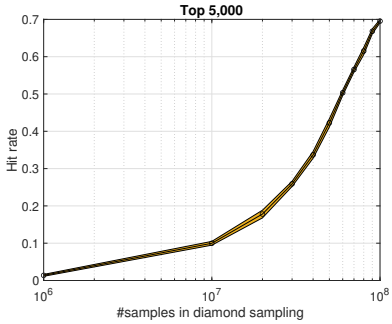
In this section, we construct an $\epsilon$-N and $k$NN similarity graph on a larger dataset, MovieLens 10M [3]. This dataset contains more than $10^7$ ratings on 65,133 movies by 71,567 users. The ratings are made on a 5-star scale with half-star increments. All users have rated at least 20 movies. We build an item-item $\epsilon$-N and $k$NN similarity graph based on diamond sampling.

### $\epsilon$-N GRAPH

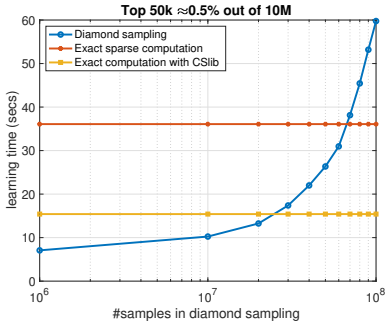
The construction performance of the $\epsilon$-N graph is shown in Figure 4.2. Figures 4.2a, 4.2b, 4.2c and 4.2d show the learning time and hit rate by preserving the top-5k and top-50k, similarity scores in the graph, respectively. When only 5k top entries are preserved, diamond sampling takes 20% less time to achieve a 58% hit rate, while to keep the top-50k, it is14% less time taken to achieve a hit rate of 65%. If we require a hit rate of 50%, diamond sampling takes half the time needed by exact computation. We also observe that the randomness of diamond sampling is negligible as the number of the samples is large.



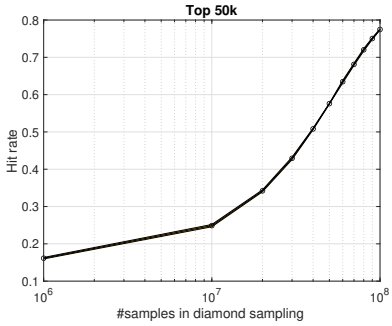

(a) Learning time of the diamond sampling with different number of samples.




(b) Hit rate of the $\epsilon$-ball graph search learned by diamond sampling w.r.t. the exact one.




(c) Learning time of the diamond sampling with different number of samples.




(d) Hit rate of the $\epsilon$-ball graph search learned by diamond sampling w.r.t. the exact one.

Figure 4.2: $\epsilon$-N graph construction performance on MovieLens 10M dataset [3].

### $k$NN GRAPH

We now do a $k$NN search for each item to build an item-item similarity matrix. We set $k$ to be 20. Since this item-by-item search involves 65,133 matrix-vector multiplications exactly or by diamond sampling, is not efficient. Thus, we only search the $k$NN for the first 100 items, which is sufficient to validate diamond sampling based $k$NN graph con-

struction. The hit rate is defined as

$$\text{Hit rate} = \frac{\#\text{correctly identified } k\text{NN}}{k}. \tag{4.2}$$

The hit rate and the consumed time are averaged over 100 items.

Table 4.1: Time comparison for 20NN search per user on ML 10M data

| Method | Time (seconds) | | | | | |
|---|---|---|---|---|---|---|
| | Number of samples in diamond sampling | | | | | |
| Diamond sampling | 200 | 800 | 3,200 | 12,800 | 51,200 | 204,800 |
| | 0.1493 | 0.1495 | 0.1572 | 0.1649 | 0.1840 | 0.2434 |
| Direct computation | 17.9449 | | | | | |
| Sparse computation | 0.0463 | | | | | |

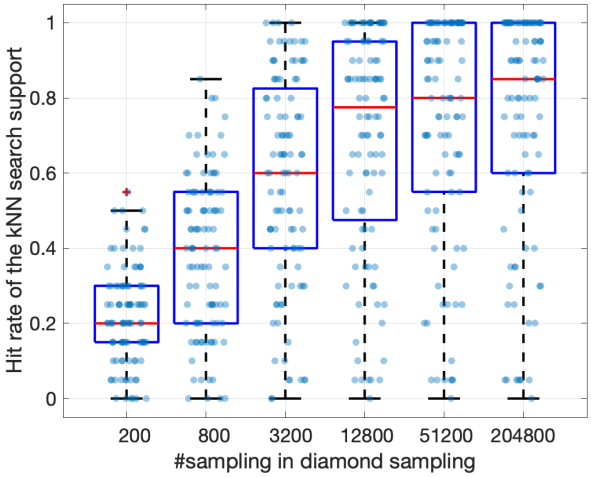



Figure 4.3: Boxplot of hit rate for 20NN search for first 100 items on ML 10M data [3]. On each box, the central mark indicates the median, and the bottom and top edges of the box indicate the 25th and 75th percentiles, respectively. The whiskers extend to the most extreme data points. The outliers are plotted individually using the '+' symbol.

The hit rate result is reported in Figure 4.3 by boxplot, where the NN found by diamond sampling of each individual item out of the first 100 items is dotted. We can see that as the number of samples increase, the accuracy of 20NN is improved. In Table 4.1, the mean learning time of 20NN search for each user is compared between diamond sampling with different number of samples, direct matrix-vector multiplication, and its sparse computation run in Matlab 2019b. We see that the sparse computation takes the least time, and diamond sampling in general takes 100 less time compared to direct computation. If a median hit rate of 80% is required, diamond sampling takes around 51,200

samples with a time cost of 0.1840 per NN search, which is around 100 times faster than direct computation, but 5 times longer than sparse computation. Diamond sampling can bring a significant improvement on the computational cost when the data is not sparse, but when the data is sparse, the simple sparse computation is still outperforming.

## 4.2. PERFORMANCE IN RATING PREDICTION

In this section, we test the performance of the similarity matrix built from diamond sampling in recommender systems. For simplicity, we only test the diamond sampling in the context of user-based collaborative filtering (CF), which is the most commonly used method.

The data we are using is MovieLens 100K [3], which is of medium size but sufficient to validate the similarity graph constructed from diamond sampling for rating prediction. The dataset consists of 100K ratings (1-5) from 943 users on 1682 movies. Each user has rated at least 20 movies. In user-based CF, after the user-user similarity graph is constructed, there are different ways of make recommendations or rating predictions. We adapt two schemes in below two subsecitons.

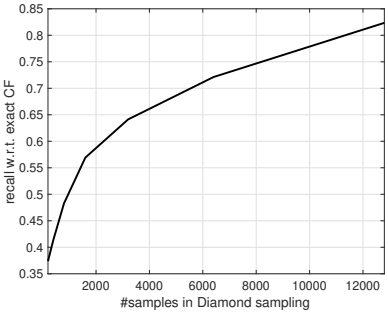
### 4.2.1. MOVIE RECOMMENDATION

First, we make recommendations for the target user based on the fully rated movies by its nearest neighbors [4]. This is a practical setting and the merchant does not necessarily looking for specific rating values. We find the similarity graph first, then for the target user, we find its top-$k$ most similar users, and lastly, we make recommendations lists as the movies rated by the neighbors with a star-5.

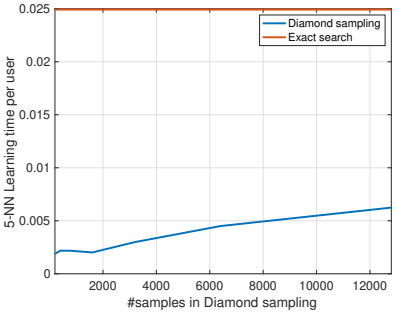
In this experiment, the whole dataset is 80%/20% split into training and testing data. From the training data, we use diamond sampling to approximately compute the user-user 5NN similarity graph. The recommendation performance is compared with respect to the recommendation results from exact 5NN similarity graph.

To build the 5-NN similarity matrix based on diamond sampling, we set the number of samples in the set $\{1\times10^3, 1.5\times10^3, \ldots, 15\times10^3\}$. We look for the top-10 entries for each user, then keep the top-5 by removing the target user itself. In Figure 4.4a, we show the recall of recommendation by diamond sampling w.r.t. the exact similarity matrix, which is defined as the number of movies in both recommendation lists divided by the number of recommendations based on exact 5NN graph. The results are averaged over all users. From the hit rate curve, we see the performance by diamond sampling can be quite close to that of computing the exact similarity matrix. This is because increasing the number of samples in diamond sampling can achieve a better approximation of the similarity matrix.

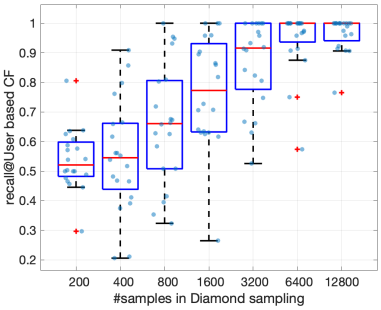
From Figure 4.4b, we observe that the 5NN search learning time based on diamond sampling is much smaller than the exact search. The box-plots in Figure 4.4c and 4.4d show the recall for the first 20 and 100 users with different number of samples. Lastly, note that diamond sampling has potential in rating prediction, because its computational cost increases linearly with the number of samples, but its performance increases rapidly. For example, when the number of samples increases from 1,000 to 10,000, the



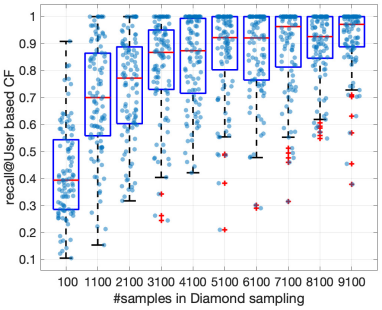

(a) Recall of the recommendations, diamond sampling CF w.r.t. exact user CF, averaged of all users.




(b) Learning time comparison per user NN search, averaged over all users.




(c) Relative recall of the first 20 users recommendations.




(d) More details of recall of the first 100 users recommendations.

Figure 4.4: Recall of simple user-based collaborative filtering recommendation generated by diamond sampling w.r.t. exact computation.

performance increases from 53% to 78%, but the time required for each NN search increases from 0.002 to 0.0052 secs.

### 4.2.2. RATING PREDICTION

In user-based collaborative filtering, we can complement the user-item rating matrix by predicting the unknown rating values. From the prediction results, we can make the top-$N$ movie recommendations [4].

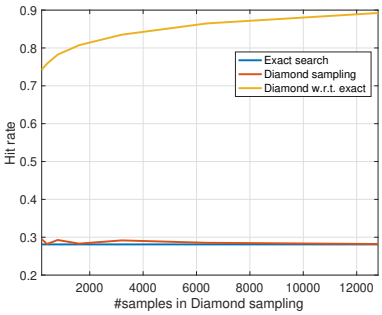
Let $P_u(j)$ denote the set of the top-$k$ similar users, for which the ratings of item $j$ have been observed, of the target user $u$. The predicted rating $\hat{r}_{uj}$ of target user $u$ for item $j$ is estimated as

$$\hat{r}_{uj} = \bar{r}_u + \frac{\sum_{v \in P_u(j)} c_{uv}(r_{vj} - \bar{r}_v)}{\sum_{v \in P_u(j)} |c_{uv}|}, \tag{4.3}$$

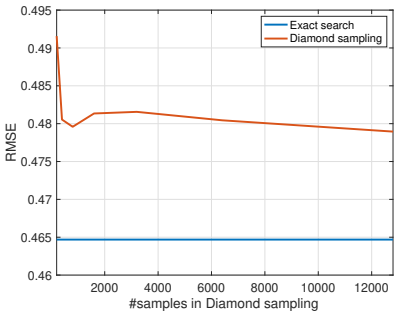
where $c_{uv}$ is the cosine similarity score between user $u$ and $v$, $\bar{r}_u$ is the average rating score of user $u$.

We consider the dataset ML 100K [3], which is again split into a training set and a

testing set in which there are exactly 10 ratings per user. With the training data, we built a 20NN similarity graph based on diamond sampling and from exact computation, then recommend the top-30 items based on (4.3). We compute the hit rate as the ratio of the number of users whose recommendation lists contain at least one of the movies from the testing data, and the total number of users [5]. Also, we compute the relative hit rate of the recommendations based on diamond sampling with respect to the one based on exact computation, i.e., the number of movies that are in both lists divided by 30.




(a) (Relative) Hit rate of the recommendations based on diamond sampling.




(b) RMSE comparison of the predicted ratings based on two methods.

Figure 4.5: Rating prediction based on user collaborative filtering by diamond sampling constructed and exact 20NN similarity graph.

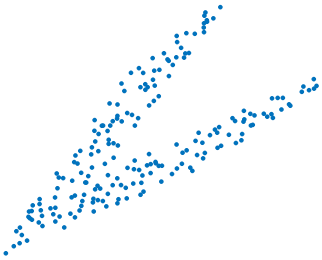
Figure 4.5 shows the accuracy of the rating prediction of diamond sampling with respect to the exact computation. Figure 4.5a shows the hit rates of both diamond sampling and the exact computation, and the relative one, which are averaged over all users. In Figure 4.5b, the RMSE between the rating predication and the test data is shown, defined as

$$\text{RMSE} = \sqrt{\frac{1}{\#\text{user}\ \#\text{item}} \sum_{(u,i)} (\hat{r}_{ui} - z_{ui})}, \tag{4.4}$$

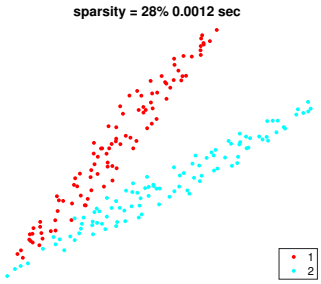
where #user is the total number of users, #item the number of items and $\hat{r}_{ui}$ and $z_{ui}$ are the predicted rating and the rating from test data, of user $u$ on item $i$, respectively. We observe that when the number of samples increases, the RMSE will decrease, which has small difference with the exact results. Regarding the learning time of each $k$NN search, it has the similar results as above in movie recommendations.

## 4.3. SPECTRAL CLUSTERING

Lastly, we conduct a synthetic experiment to find out if the similarity matrix built with diamond sampling can perform well in spectral clustering. The goal here is to validate diamond sampling, instead of its efficiency. We generate 200 data points in the 2D plane, where each of them is along with a specific direction, $30^o$, or $60^o$, respectively, but contaminated by random noise, indicating that we have two clusters of data points. Our goal is to partition them without supervision.

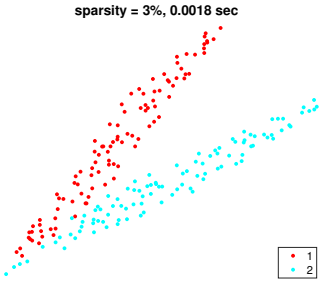


(a) 200 data points.




(b) Results from exact similarity matrix with top-28% entries.

(c) Similarity graph constructed by diamond sampling.



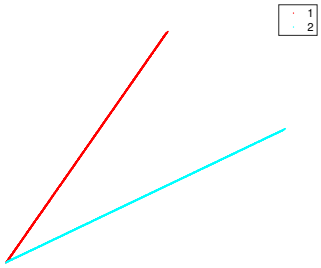

(d) Diamond sampling based similarity matrix with top-3% entries.

Figure 4.6: Toy experiments on spectral clustering.

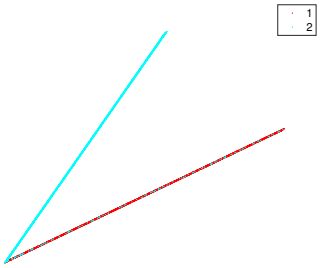
We follow the spectral clustering method as in section 2.4 by first constructing the similarity matrix from the data and we use the "*spectralcluster*" function in Matlab [6–8]. Specifically, we sparsify the exact similarity graph adjacency matrix by keeping the top-28% entries, while for the one built by diamond sampling, only the top-3% is preserved. This is tested to get a small number of samples as much as possible while preserving the clustering performance. For diamond sampling, we set the number of samples the same as the number of the nonzeros entries. This procedure takes the similar time complexity as exact computation, which is not significantly improved, because the data matrix dimension is of $200 \times 2$, too small to show the efficiency of diamond sampling.

The clustering results are shown in Figure 4.6, from which we observe that both similarity matrices are able to cluster the data points well. Their performance is similar except for two data points that are partitioned differently. Note that here we only showed an instance of the diamond sampling based result, which is random and has clustering variations on the middle fused nodes.

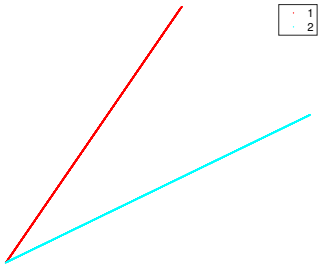
In Figure 4.7, we show the spectral clustering results based on exact similarity graph and diamond sampling constructed similarity graph for 5K and 10K data points. From



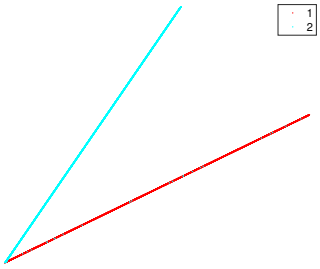

(a) Results from exact similarity matrix for 5K points.




(b) Results from diamond sampling for 5K points.




(c) Results from exact similarity matrix for 10K points.




(d) Results from diamond sampling for 10K points.

Figure 4.7: Spectral clustering with 5K and 10K data points.

the experimental results, for 5K data points, it takes 30% less time to build the similarity graph by diamond sampling (1.3882 and 0.9637 seconds, respectively), with an accuracy of 96.26%. For 10K data points, it takes 60% less time to build the similarity graph (12.2968 and 4.69 seconds, respectively), with an accuracy of 99.73%. We point out that diamond sampling is potential to reduce the computational cost in spectral clustering, but other acceleration methods to speed up the other steps are needed as well.

## 4.4. CONCLUSION

In Part I, we discussed how to improve the efficiency of similarity graph construction from data. This is an important task in many application, e.g., recommender systems, and spectral embedding/clustering. We managed to leverage the fast matrix product approximation technique, *diamond sampling*, to achieve a much faster graph construction method.

In chapter 2, we reviewed and developed *diamond sampling* for approximating the maximum product search in large matrices. In chapter 3 we have proposed how to use diamond sampling to construct cosine similarity based graph. In both chapters, a the-

oretical study is provided as well as the background information regarding the experiments. A big improvement of diamond sampling based fast graph construction algorithms is in no need of high performance computing tools, compared to the state-of-the-art. However, high performance computing can also be incorporated, as each major sampling step in diamond sampling is independent of each other. In chapter 4, with experiments on showing the performance of diamond sampling in cosine similarity based $\epsilon$-N or $k$NN graph construction. Diamond sampling can improve the efficiency compared to the direct computation, while preserving an acceptable performance. This is needed especially in the big data processing.

In the future, the extension of other similarity measures can be carried out. The ground rule is to translate other measures into a matrix product. Regarding the $k$NN similarity graph construction, so far we only achieved the NN search per node efficiently. When the graph is too large, this still cannot resolve the high computational cost problem. Finally, as indicated in section 3.3, research is needed to further reduce the computational cost and storage cost for diamond sampling.

# REFERENCES

[1] G. Ballard, T. G. Kolda, A. Pinar, and C. Seshadhri, *Diamond sampling for approximate maximum all-pairs dot-product (mad) search,* 2015 IEEE International Conference on Data Mining (2015), 10.1109/icdm.2015.46.

[2] T. A. Davis, *Direct methods for sparse linear systems* (SIAM, 2006).

[3] F. M. Harper and J. A. Konstan, *The movielens datasets: History and context,* Acm transactions on interactive intelligent systems (tiis) **5**, 1 (2015).

[4] C. C. Aggarwal *et al.*, *Recommender systems*, Vol. 1 (Springer, 2016).

[5] X. Ning and G. Karypis, *Slim: Sparse linear methods for top-n recommender systems,* in *2011 IEEE 11th International Conference on Data Mining* (IEEE, 2011) pp. 497–506.

[6] J. Shi and J. Malik, *Normalized cuts and image segmentation,* IEEE Transactions on pattern analysis and machine intelligence **22**, 888 (2000).

[7] A. Y. Ng, M. I. Jordan, and Y. Weiss, *On spectral clustering: Analysis and an algorithm,* in *Advances in neural information processing systems* (2002) pp. 849–856.

[8] U. Von Luxburg, *A tutorial on spectral clustering,* Statistics and computing **17**, 395 (2007).

# II

# GRAPH SIGNAL RECONSTRUCTION

# 5

# BACKGROUND

In second part of this thesis, we focus on the signal reconstruction over graphs. This chapter provides the background knowledge of topic, Graph signal processing (GSP) [1],[2]. First, we review the previous work in graph signal reconstruction. Then, we provide the graph signal processing basic knowledge to prepare for later chapters, e.g., graph signals, graph Fourier transform, graph filtering and so on. Finally, we conclude the chapter by outlining the work in Part II.

## 5.1. INTRODUCTION

In this part, we study graph signal reconstruction. As in classical signal processing, the task of *estimating the underlying signal from noisy observations. i.e., signal denoising,* is a critical task in GSP. This is a fundamental problem and can be useful in many other problems, for instance, graph signal sampling, graph-based semi-supervised learning [1–4]. The most commonly used method is graph Laplacian denoising based on Tikhonov regularization [1, 5].

### 5.1.1. LITERATURE REVIEW

Over the last few years, a large amount of related research has been focused on the topic of graph signal reconstruction [1, 2, 5–16]. Within this body of work, most of the approaches rely on solving a regularized least squares problem under a smoothness assumption and Tikhonov denoising is the most common one [1],[5].

Different variants of regularized signal recovery methods find a solution at a lower complexity cost by means of iterative and distributed methods [6–12]. For instance, [6] proposed two local-set-based iterative methods relying on frame theory, while [8] put forth a distributed implementation for time-varying signals. In contrast, in [7], a least squares problem is regularized to suppress the high frequency components. Furthermore, the work in [9] generalized the classical least mean squares and recursive least squares methods to adaptively estimate the graph signal. In [11], the problem was regularized using the signal total variation and it was solved distributively by a primal dual

hybrid gradient method. The authors of [12] proposed the Chebyshev polynomial approximation to solve the same problem distributively. Similarly, in [13–16], the conventional smoothness measure was generalized by means of graph kernels. These graph kernels, defined as polynomial functions of the matrix representation of the graph, allow to capture different structures present in the data. Finally, regarding the choice of the regularization parameter in Tikhonov-based denoising, [5] has studied the bias-variance trade-off controlled by this parameter as well as its optimal value, which will be reviewed in detail later.

Although state-of-the-art successfully address the signal denoising problem, almost all proposed methods adopt a single parameter to control the performance of the method; that is, its fitting error and regularization cost. However, a single regularization parameter is insufficient to reflect the penalty locally over the graph. To see this, consider an instance of graph Laplacian denoising, which penalizes the error fitting term with signal smoothness measure. A scalar regularization weight can only penalize the global signal smoothness, instead of local penalties, which are important for signals without global smoothness, such as piecewise-smooth, piecewise-constant signals [17, 18]. This choice limits the denoising performance and motivates us to improve the dimension of regularization parameters, i.e., the degrees of freedom (DoFs).

Some initial work to increase the DoFs in the design is for instance [11]. This work proposes to minimize the signal total smoothness by separately regularizing the fitting error of each individual node. Despite that this approach can be considered as a multi-parameter based regularization, it focuses on a measurement-specific regularization, similar to a weighted least squares approach. It also does not consider the possible heterogeneity at node level, i.e., the parameters are related to the different measurements rather than to the different nodes. In this part, we consider a multi-parameter regularization scheme to penalize the reconstruction problem with the local graph signal smoothness, so to obtain an improved performance.

### 5.1.2. OUTLINE OF PART II

This part is outlined as follows.

- In chapter 6, we propose *node-adaptive (NA)* or *node-varying* graph signal regularization. Then, we study its bias-variance trade-off and implementation. Afterwards, we look into how to design the regularization parameters. Finally, we conduct several experiments to validate the NA signal regularization over graphs on the synthetic and real data.

- Graph signal sampling is strongly related to graph signal reconstruction. As a reconstruction method, NA regularization is promising in sampling and interpolation problems. In chapter 7, we first review the development in graph signal sampling problem. Then, we analyze the Tikhonov regularization based method. Finally, we apply NA regularization in sampling problems and improve the interpolation performance.

- In chapter 8, we conclude the second part of this thesis and propose several future research statements.

## 5.2. GRAPH SIGNAL PROCESSING

Consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \ldots, N\}$ is the set of $N$ nodes and $\mathcal{E}$ the set of $M$ edges such that if nodes $i$ and $j$ are connected, then $(i, j) \in \mathcal{E}$. The neighborhood set of node $i$ is $\mathcal{N}_i = \{j | (i, j) \in \mathcal{E}\}$. The graph can be represented by its adjacency matrix $\mathbf{A}$ with entry $A_{ij} \geq 0$ if $(i, j) \in \mathcal{E}$ and $A_{ij} = 0$, otherwise. Alternatively, the graph can also be represented by its graph Laplacian matrix $\mathbf{L} = \text{diag}(\mathbf{A1}) - \mathbf{A}$. The graph Laplacian is a positive semi-definite matrix and is one of the candidates of the graph shift operator matrix $\mathbf{S}$. Other candidates for $\mathbf{S}$ are adjacency matrix, normalized graph Laplacian.

### 5.2.1. GRAPH SIGNAL VARIATION

On the vertices of $\mathcal{G}$, we define a *graph signal* $\mathbf{x} = [x_1, \ldots, x_N]^\top$ whose $i$-th element, $x_i$, is the signal value on node $i$. The signal variation, with respect to the graph, can be measured by the *graph Laplacian quadratic form*

$$S_2(\mathbf{x}) = \frac{1}{2} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} A_{ij} (x_i - x_j)^2 = \mathbf{x}^\top \mathbf{L} \mathbf{x}. \tag{5.1}$$

If the signal $\mathbf{x}$ is *smooth* over $\mathcal{G}$, then $S_2(\mathbf{x})$ is small [1]. This measure is used as a regularizer to recover smooth graph signals from noisy measurements [1, 2, 5], which is so-called *graph Laplacian denoising*, or *Tikhonov regularization*.

### 5.2.2. GRAPH SHIFT OPERATOR

One of the central operations in GSP is shifting a graph signal $\mathbf{x}$ over the graph as follows

$$\mathbf{x}^{(1)} = \mathbf{S}\mathbf{x}, \tag{5.2}$$

where $\mathbf{x}^{(1)}$ represents the one step shifted version of signal $\mathbf{x}$ by $\mathbf{S}$. The physical meaning of shifting is that each node collects the information from its neighbors. Similarly, higher order shifts can be computed recursively as

$$\mathbf{x}^{(k)} = \mathbf{S}\mathbf{x}^{(k-1)} = \mathbf{S}^k \mathbf{x}, \tag{5.3}$$

where each node can aggregate the information from its $k$-hop neighbors.

### 5.2.3. GRAPH FOURIER TRANSFORM

The graph Fourier transform (GFT) relies on the spectral decomposition of the graph shift operator. In this thesis, we consider graph Laplacian, which can be decomposed as

$$\mathbf{L} = \mathbf{U} \mathbf{\Lambda} \mathbf{U}^\top, \tag{5.4}$$

where $\mathbf{U} = (\mathbf{u}_0, \ldots, \mathbf{u}_{N-1})$ is an $N \times N$ orthonormal matrix containing eigenvectors of $\mathbf{L}$, and $\mathbf{\Lambda} = \text{diag}(\lambda_0, \ldots, \lambda_{N-1})$ is the eigenvalue matrix. We order the eigenvalues as $0 = \lambda_0 \leq \lambda_1 \leq \lambda_2 \leq \ldots, \leq \lambda_{N-1}$, where the number of zero eigenvalue equals to the number of connected components of the graph. We then define the following.

**Definition 1.** *Graph Fourier transform (GFT): The GFT* $\tilde{\mathbf{x}}$ *of a graph signal* $\mathbf{x}$ *sitting on the graph* $\mathcal{G}$ *with graph Laplacian* $\mathbf{L}$ *is defined as*

$$\tilde{\mathbf{x}} = \mathbf{U}^\top \mathbf{x}, \tag{5.5}$$

*where* $\mathbf{U}$ *is the eigenvector matrix of* $\mathbf{L}$.

**Definition 2.** *Inverse graph Fourier transform (IGFT): The IGFT of* $\tilde{\mathbf{x}}$ *is defined as*

$$\mathbf{x} = \mathbf{U}\tilde{\mathbf{x}}, \tag{5.6}$$

*where* $\mathbf{U}$ *is the eigenvector matrix of* $\mathbf{L}$.

**Definition 3.** *Graph frequencies: The frequencies of a graph are the eigenvalues* $\lambda_0, \ldots, \lambda_{N-1}$ *of the graph Laplacian.*

The GFT of a graph signal $\mathbf{x}$ expands itself in the eigenspace of the graph Laplacian. The GFT coefficients $\tilde{\mathbf{x}}$ measure the expansion weight of the signal over each eigenvector. The graph Laplacian eigenvectors act as the oscillating modes of the graph [19]. In this sense, the GFT is a generalization of the temporal Fourier transform. Instead, the GFT maps the signal from vertex domain to graph spectral domain. Likewise, the IGFT maps signal from graph spectral domain to vertex domain.

### 5.2.4. GRAPH SIGNAL BANDWIDTH

A graph signal $\mathbf{x}$ is said to be low-pass if it is concentrated in the frequencies with slow-varying eigenvectors. Similarly, a high-pass graph signal has most energy concentrated in the graph frequencies corresponding to the fast-varying eigenvectors.

A *bandlimited* graph signal on $\mathcal{G}$ with bandwidth $|\mathcal{F}| \leq N$ is and only if its GFT $\tilde{\mathbf{x}}$ is nonzeros on the limited set $\mathcal{D}$ of graph frequencies, i.e., the set $\mathcal{F}$ consists of

$$\mathcal{F} = \{\lambda_i | \tilde{x}_i \neq 0, \text{ for } i \in \{0, \ldots, N-1\}\}. \tag{5.7}$$

### 5.2.5. GRAPH FILTERING

Similar to traditional signal processing, filtering can be done in GSP by weighting the GFT of each frequency differently. For instance, if a low-pass graph signal is perturbed by a high-pass noise, then a low-pass filtering can be used to denoise.

**Definition 4.** *Graph filtering: A linear shift-invariant graph filter is an operation on the graph signal with graph frequency domain output*

$$\tilde{\mathbf{y}} = h(\mathbf{\Lambda})\tilde{\mathbf{x}}, \tag{5.8}$$

*where* $h(\mathbf{\Lambda})$ *is the graph filter response, defined as a function over the graph frequencies* $\mathbf{\Lambda}$ *to the real number set, and assigns a particular value* $h(\lambda_n)$ *to each graph frequency* $\lambda_n$. *In vertex domain, the filtering output is*

$$\mathbf{y} = \mathbf{H}\mathbf{x} = \mathbf{U}h(\mathbf{\Lambda})\mathbf{U}^\top \mathbf{x}, \tag{5.9}$$

*with graph filter* $\mathbf{H} = \mathbf{U}h(\mathbf{\Lambda})\mathbf{U}^\top$.

**Definition 5.** *FIR graph filters: A popular choice of graph filter* $\mathbf{H}$ *is a polynomial of the graph shift operator*[1]*, i.e.,*

$$\mathbf{H} \triangleq \sum_{k=0}^{K} \phi_k \mathbf{S}^k, \tag{5.10}$$

*where* $\phi_k$ *is the graph filter coefficient. The corresponding graph filter response* $h(\lambda)$ *is*

$$h(\lambda) = \sum_{k=0}^{K} \phi_k \lambda^k, \tag{5.11}$$

*where* $\lambda$ *is a graph frequency.*

When an FIR filter is applied to a graph signal $\mathbf{x}$, the nodes can compute locally the $k$the shift of $\mathbf{x}$ from the former $(k-1)$th shift, since $\mathbf{S}^k\mathbf{x} = \mathbf{S}(\mathbf{S}^{k-1}\mathbf{x})$,. An Fir filter of order $K$ requires $K$ local exchanges between neighbors and amounts to a computation and communication complexity of $\mathcal{O}(MK)$ [19, 20].

In [21], an ARMA recursion on graphs to implement distributively rational graph filtering, i.e., a filtering operation characterized by a rational frequency response.

**Definition 6.** *ARMA graph filters. The building block of this filter is the so-called ARMA graph filter of order one (*$ARMA_1$*). This filter is obtained as the steady-state of the first-order recursion*

$$\mathbf{y}_t = \psi \mathbf{S}\mathbf{y}_{t-1} + \phi\mathbf{x}, \tag{5.12}$$

*with arbitrary* $\mathbf{y}_0$ *and scalar coefficients* $\psi$ *and* $\phi$*. The frequency response of the* $ARMA_1$ (5.12) *is*

$$h(\lambda) = \frac{r}{\lambda - p}, \text{ subject to } |p| > \rho \tag{5.13}$$

*with residual* $r = -\phi/\psi$*, pole* $p = 1/\psi$ *and* $\rho$ *the spectral radius bound of* $\mathbf{S}$*.*

Given the condition of $|\psi\rho| < 1$, the recursion (5.12) achieves the frequency response (5.13). The operation (5.12) is a distributed recursion on graphs, where neighbors now exchange their former output $\mathbf{y}_{t-1}$ rather than the input $\mathbf{x}$. The per-iteration complexity of such a recursion of $\mathcal{O}(M)$.

### 5.2.6. GRAPH LAPLACIAN DENOISING, TIKHONOV REGULARIZATION

Through Part II, we consider the following signal model

$$\mathbf{y} = \mathbf{x}^* + \mathbf{n}, \tag{5.14}$$

where $\mathbf{y}$ is the noisy signal measurement, $\mathbf{x}^* \in \mathbb{R}^N$ is the smooth graph signal and $\mathbf{n} \in \mathbb{R}^N$ is zero-mean noise with covariance matrix $\mathbf{\Sigma}$.

The graph Laplacian quadratic form is used to regularize to recover a smooth graph signal. Specifically, if $\mathbf{x}^*$ is smooth, we can reconstruct it from the noisy measurements by solving the Tikhonov regularized problem

$$\hat{\mathbf{x}}(\omega_0) = \underset{\mathbf{x}\in\mathbb{R}^N}{\arg\min} \ \|\mathbf{y}-\mathbf{x}\|_2^2 + \omega_0 \mathbf{x}^\top \mathbf{L}\mathbf{x} \tag{5.15}$$

[1] In this thesis, we consider the choice of the graph Laplacian $\mathbf{L}$

where $\omega_0 > 0$ is a scalar regularization parameter. The first term in (5.15) forces the estimate $\hat{\mathbf{x}}(\omega_0)$ to be close to the observed signal $\mathbf{y}$ (fitting term), while the second term promotes smoothness. The trade-off between these two quantities is governed by the scalar weight $\omega_0$. The closed-formed solution of (5.15) is

$$\hat{\mathbf{x}}(\omega_0) = (\mathbf{I} + \omega_0 \mathbf{L})^{-1} \mathbf{y} := \mathbf{H}(\omega_0)\mathbf{y} \tag{5.16}$$

where we defined $\mathbf{H}(\omega_0) \triangleq (\mathbf{I} + \omega_0 \mathbf{L})^{-1}$ for convenience. Note that from Definition 6, recursion (5.12) is an ARMA graph filter of order one.

BIAS-VARIANCE TRADE-OFF

The smooth regularizer in (5.15) biases the estimator $\hat{\mathbf{x}}(\omega_0)$ in (5.16). The bias $\mathbf{b}(\omega_0)$ is given by

$$\mathbf{b}(\omega_0) = \mathbb{E}(\hat{\mathbf{x}}(\omega_0)) - \mathbf{x}^* = ((\mathbf{I} + \omega_0 \mathbf{L})^{-1} - \mathbf{I})\mathbf{x}^* \tag{5.17}$$

which is controlled by $\omega_0$. Also the variance over all nodes is controlled by $\omega_0$ and can be expressed as

$$\text{var}(\omega_0) = \mathbb{E}(\|\hat{\mathbf{x}}(\omega_0) - \mathbb{E}(\hat{\mathbf{x}}(\omega_0))\|^2) = \text{tr}(\mathbf{H}^2(\omega_0)\mathbf{\Sigma}). \tag{5.18}$$

By combining the bias and the variance, we can quantify the performance of the estimator (5.16) by its MSE

$$\begin{aligned} \text{mse}(\omega_0) &= \mathbb{E}(\|\hat{\mathbf{x}}(\omega_0) - \mathbf{x}^*\|_2^2) = \|\mathbf{b}(\omega_0)\|_2^2 + \text{var}(\omega_0) \\ &= \text{tr}((\mathbf{I} - \mathbf{H}(\omega_0))^2 \mathbf{x}^* \mathbf{x}^{*\top}) + \text{tr}(\mathbf{H}^2(\omega_0)\mathbf{\Sigma}). \end{aligned} \tag{5.19}$$

The MSE exhibits the bias-variance trade-off imposed by the smoothness regularizer. If the scalar $\omega_0$ is reduced, we achieve a lower bias but a higher variance, and vice-versa. A good value for $\omega_0$ can be obtained by some traditional parameter selection methods such as the discrepancy principle [22, 23], the $L$-curve criterion [24] and the generalized cross-validation [25]. In GSP, a natural optimal parameter selection criterion is based on the MSE, which is investigated in [5]. Parameter $\omega_0$ is found by equating the bias contribution to the variance. This is because the MSE expression is not a convex function w.r.t. $\boldsymbol{\omega}_0$. By matching the bias and the variance term, an upper-bound of MSE can be minimized [5].

However, since the balance between the bias and the variance is controlled from a scalar $\boldsymbol{\omega}_0$, it is often required to substantially pay in one metric for gaining little on the other metric. Starting from the next chapter, we show how to improve this by changing problem (5.15) to a node-adaptive regularization problem on graphs.

## REFERENCES

[1] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, *The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains,* IEEE Signal Processing Magazine **30**, 83 (2013).

[2] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst, *Graph signal processing: Overview, challenges, and applications,* Proceedings of the IEEE **106**, 808 (2018).

[3] J. Jia, M. T. Schaub, S. Segarra, and A. R. Benson, *Graph-based semi-supervised & active learning for edge flows,* in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019) pp. 761–771.

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

[5] P. Chen and S. Liu, *Bias-variance tradeoff of graph laplacian regularizer,* IEEE Signal Processing Letters **24**, 1118 (2017).

[6] X. Wang, P. Liu, and Y. Gu, *Local-set-based graph signal reconstruction,* IEEE Transactions on Signal Processing **63**, 2432 (2015).

[7] S. K. Narang, A. Gadde, E. Sanou, and A. Ortega, *Localized iterative methods for interpolation in graph structured data,* in *2013 IEEE Global Conference on Signal and Information Processing* (2013) pp. 491–494.

[8] X. Wang, M. Wang, and Y. Gu, *A distributed tracking algorithm for reconstruction of graph signals,* IEEE Journal of Selected Topics in Signal Processing **9**, 728 (2015).

[9] P. Di Lorenzo, P. Banelli, E. Isufi, S. Barbarossa, and G. Leus, *Adaptive graph signal processing: Algorithms and optimal sampling strategies,* IEEE Transactions on Signal Processing **66**, 3584 (2018).

[10] E. Isufi, P. Di Lorenzo, P. Banelli, and G. Leus, *Distributed wiener-based reconstruction of graph signals,* in *2018 IEEE Statistical Signal Processing Workshop (SSP)* (2018) pp. 21–25.

[11] P. Berger, G. Hannak, and G. Matz, *Graph signal recovery via primal-dual algorithms for total variation minimization,* IEEE Journal of Selected Topics in Signal Processing **11**, 842 (2017).

[12] D. I. Shuman, P. Vandergheynst, D. Kressner, and P. Frossard, *Distributed signal processing via chebyshev polynomial approximation,* IEEE Transactions on Signal and Information Processing over Networks **4**, 736 (2018).

[13] A. Venkitaraman, S. Chatterjee, and P. Händel, *Predicting graph signals using kernel regression where the input signal is agnostic to a graph,* IEEE Transactions on Signal and Information Processing over Networks , 1 (2019).

[14] V. N. Ioannidis, M. Ma, A. N. Nikolakopoulos, G. B. Giannakis, and D. Romero, *Kernel-based Inference of Functions over Graphs,* arXiv e-prints , arXiv:1711.10353 (2017), arXiv:1711.10353 [stat.ML] .

[15] D. Romero, M. Ma, and G. B. Giannakis, *Kernel-based reconstruction of graph signals,* IEEE Transactions on Signal Processing **65**, 764 (2017).

[16] A. Venkitaraman, S. Chatterjee, and P. Händel, *Multi-kernel Regression For Graph Signal Processing,* arXiv e-prints , arXiv:1803.04196 (2018), arXiv:1803.04196 [stat.ML] .

[17] Y.-X. Wang, J. Sharpnack, A. J. Smola, and R. J. Tibshirani, *Trend filtering on graphs,* The Journal of Machine Learning Research **17**, 3651 (2016).

[18] S. Chen, R. Varma, A. Singh, and J. Kovačević, *Representations of piecewise smooth signals on graphs,* in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, 2016) pp. 6370–6374.

[19] E. Isufi, *Graph-time signal processing: Filtering and sampling strategies,* (2019).

[20] M. Coutino, E. Isufi, and G. Leus, *Advances in distributed graph filtering,* IEEE Transactions on Signal Processing **67**, 2320 (2019).

[21] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, *Autoregressive moving average graph filtering,* IEEE Transactions on Signal Processing **65**, 274 (2017).

[22] O. Scherzer, *The use of morozov's discrepancy principle for tikhonov regularization for solving nonlinear ill-posed problems,* Computing **51**, 45 (1993).

[23] S. W. Anzengruber and R. Ramlau, *Morozov's discrepancy principle for tikhonov-type functionals with nonlinear operators,* Inverse Problems **26**, 025001 (2009).

[24] P. C. Hansen and D. P. O'Leary, *The use of the l-curve in the regularization of discrete ill-posed problems,* SIAM Journal on Scientific Computing **14**, 1487 (1993), https://doi.org/10.1137/0914086 .

[25] G. H.Golub and U. von Matt, *Generalized cross-validation for large-scale problems,* Journal of Computational and Graphical Statistics **6**, 1 (1997), https://amstat.tandfonline.com/doi/pdf/10.1080/10618600.1997.10474725 .

# 6

# NODE-ADAPTIVE GRAPH SIGNAL REGULARIZATION

Graph Laplacian denoising (5.15) penalizes the error fitting term with global signal smoothness measure weighted by a scalar regularization weight $\omega_0$. Since all nodal values are weighted equally, we refer to problem (5.15) as a node-invariant (NI) regularization. In this chapter, we propose the node-adaptive (NA) graph regularization.

If the signal is globally smooth over the underlying graph, the NI regularization will result in a good signal reconstruction performance. However, it focuses only on the big picture, ignoring local signal detail and ultimately leading to an unsatisfactory performance. This is particularly emphasized when the signal is not strictly smooth globally over the underlying graph such as piecewise-smooth or piecewise-constant signals [1, 2]. In most situations, graph signals are smooth locally. This is a more general and relaxed assumption on the signal compared to the global smoothness and is adaptive. To enhance the role of local detail and improve the estimator MSE, we propose a graph signal regularization strategy that replaces the global penalty term $S_2(\mathbf{x})$ with a local penalty for each node. This strategy allows each node $i$ to weigh its own signal $x_i$ with an individual scalar $\omega_i$. With the enhanced DoFs, we expect a better denoising performance without increasing the complexity.

It is, however, unclear what is the new trade-off imposed by said regularizer and how to design these node-adaptive weights $\omega_1, \ldots, \omega_N$ to balance such a trade-off. Thus, in this chapter, three main questions are proposed i) *what is the structure of the solution of NA Tikhonov-based denoising;* ii) *how does the bias-variance trade-off affects the performance of this solution;* iii) *how to leverage this solution to optimally design node-dependent weights.*

Aiming to answer these questions, our main contributions are the following.

– We formulate the NA Tikhonov denoising problem under a deterministic signal model (5.14), derive its closed-form solution, and study its bias-variance trade-off. We also state two theoretical results supporting the improved performance of NA

weights compared to the NI weight. Specifically, we provide conditions for the NA weights that allow for a lower mean-squared error (MSE) and variance compared to NI Tikhonov-based denoising.

– Regarding the choice of the NA weights, we propose three methods based on minimizing the MSE. The first two methods leverage the famous Prony's method from classical signal processing as well as convex relaxation techniques. The third method addresses the case where we have no access to the true signals, and only know their variation bounds. We use a *minimax* strategy to design the weights minimizing the worst-case performance of the regularizer. The latter method addresses scenarios where only upper- and lower-bounds for the node signals are available.

For the sake of exposition, we solely focus on *NA Tikhonov-based denoising*. However, our findings can be generalized to other regularization penalties, such as any graph shift operator based ridge regression penalty [3, 4].

## 6.1. NODE-ADAPTIVE REGULARIZER

Consider a vector of parameters $\boldsymbol{\omega} = [\omega_1, \ldots, \omega_N]^\top \in \mathbb{R}^N$ and define the node-adaptive Laplacian operator as

$$\mathbf{S}(\boldsymbol{\omega}) \triangleq \text{diag}(\boldsymbol{\omega})\mathbf{L}\text{diag}(\boldsymbol{\omega}) = \boldsymbol{\omega}\boldsymbol{\omega}^\top \odot \mathbf{L} \tag{6.1}$$

where $\odot$ is the element-wise Hadamard product. For any value of $\boldsymbol{\omega}$, the parametric shift operator matrix $\mathbf{S}(\boldsymbol{\omega})$ is positive semi-definite (see Lemma 9 in Appendix B.1) and has the same support as the graph Laplacian $\mathbf{L}$ –properties that will result useful in the sequel. The parameterized graph Laplacian quadratic form (6.1) has the form [cf.(5.1)]

$$\begin{aligned} S_2(\mathbf{x}, \boldsymbol{\omega}) &= \mathbf{x}^\top \mathbf{S}(\boldsymbol{\omega})\mathbf{x} = (\text{diag}(\boldsymbol{\omega})\mathbf{x})^\top \mathbf{L}(\text{diag}(\boldsymbol{\omega})\mathbf{x}) \\ &= \frac{1}{2}\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i} A_{ij}(\omega_i x_i - \omega_j x_j)^2. \end{aligned} \tag{6.2}$$

This quadratic form can be seen in a two-way step: first it applies a pre-weighting to each entry $x_i$ of $\mathbf{x}$ using parameter $w_i$, i.e., $\text{diag}(\boldsymbol{\omega})\mathbf{x}$; then computes the regular quadratic measure in (5.1) w.r.t. Laplacian $\mathbf{L}$. We can use (6.2) to recover a graph signal $\mathbf{x}^*$ from noisy measurements $\mathbf{y}$ by solving the convex problem

$$\hat{\mathbf{x}}(\boldsymbol{\omega}) = \underset{\mathbf{x}\in\mathbb{R}^N}{\arg\min}\ \|\mathbf{y}-\mathbf{x}\|_2^2 + \mathbf{x}^\top \text{diag}(\boldsymbol{\omega})\mathbf{L}\text{diag}(\boldsymbol{\omega})\mathbf{x}, \tag{6.3}$$

where the trade-off between the fitting error and the regularization term is now controlled by the $N$ parameters in $\boldsymbol{\omega}$.

To see the impact of problem (6.3) on capturing local detail, suppose we are interested to recover a graph signal that is smooth only on a block of connected nodes $\mathcal{B} \subseteq \mathcal{V}$. We can then set the respective entries in $\boldsymbol{\omega}$ for all in $\mathcal{B}$ to be a shared value, $\omega_i = \sqrt{\omega_{\mathcal{B}}}$, and set all remaining entries to zero. Problem (6.3) will then seek for a graph signal that is locally smooth on $\mathcal{B}$ while it fits $\mathbf{y}$ on the remaining nodes $\mathcal{V}\setminus\mathcal{B}$. Problem (5.15) is the particular case of (6.3) for $\mathcal{B} = \mathcal{V}$ and $\omega_{\mathcal{B}} = \omega_0$. This indicates our more general local signal smoothness priors, compared to the global smoothness used in Tikhonov

regularization. In the sequel, we will design this parameter $\boldsymbol{\omega}$ to learn the local signal smoothness priors adaptively.

Optimization (6.3) forms our node-adaptive graph signal denoising problem and has the optimal solution

$$\hat{\mathbf{x}}(\boldsymbol{\omega}) = (\mathbf{I} + \mathbf{S}(\boldsymbol{\omega}))^{-1}\mathbf{y} := \mathbf{H}(\boldsymbol{\omega})\mathbf{y} \tag{6.4}$$

where we again defined $\mathbf{H}(\boldsymbol{\omega}) \triangleq (\mathbf{I} + \mathbf{S}(\boldsymbol{\omega}))^{-1}$ to simplify notation. Despite the similarity with (5.16), the optimal solution in (6.4) is now governed by the vector of parameters $\boldsymbol{\omega}$ and not anymore by the scalar $\omega_0$. The latter changes the bias-variance trade-off as we will discuss in the next section.

The optimal regularized solutions (5.16) and (6.4) can also be interpreted as graph filtering operations [5]. In particular, (5.16) consists of filtering the measurements $\mathbf{y}$ with an autoregressive graph filter with denominator coefficients $(1;\omega_0)$ which is common for all nodes [6]. The node-adaptive solution (6.4) consists of filtering $\mathbf{y}$ with an autoregressive edge varying filter with edge varying coefficients $(\mathbf{I};\boldsymbol{\omega}\boldsymbol{\omega}^\top)$ [7]. The latter allows for a direct efficient and distributed implementation of these approaches.

## 6.2. BIAS-VARIANCE TRADE-OFF

Similar to (5.19) in graph Laplacian denoising, the MSE for estimator (6.4) has the form

$$\text{mse}(\boldsymbol{\omega}) = \|\mathbf{b}(\boldsymbol{\omega})\|_2^2 + \text{var}(\boldsymbol{\omega}) = \text{tr}((\mathbf{I} - \mathbf{H}(\boldsymbol{\omega}))^2\mathbf{x}^*\mathbf{x}^{*\top}) + \text{tr}(\mathbf{H}(\boldsymbol{\omega})^2\boldsymbol{\Sigma}) \tag{6.5}$$

with bias

$$\mathbf{b}(\boldsymbol{\omega}) = ((\mathbf{I} + \text{diag}(\boldsymbol{\omega})\mathbf{L}\text{diag}(\boldsymbol{\omega}))^{-1} - \mathbf{I})\mathbf{x}^* \tag{6.6}$$

and variance

$$\text{var}(\boldsymbol{\omega}) = \text{tr}((\mathbf{I} + \text{diag}(\boldsymbol{\omega})\mathbf{L}\text{diag}(\boldsymbol{\omega}))^{-2}\boldsymbol{\Sigma}). \tag{6.7}$$

As it follows from (6.5)-(6.7), the bias-variance trade-off is now controlled by vector $\boldsymbol{\omega}$. If all entries of $\boldsymbol{\omega}$ are close to zero the bias is low and the MSE is governed by a high variance. If all entries of $\boldsymbol{\omega}$ are far from zero, the bias is large and governs the MSE, while the variance is small. The enhanced DoFs of the NA regularizer compared to the NI one allow us to identify an interval for $\boldsymbol{\omega}$ that guarantees a smaller reconstruction variance. This is confirmed by the following lemma.

**Lemma 5.** *Consider the node-invariant and node-adaptive estimates* $\hat{\mathbf{x}}(\omega_0)$ *and* $\hat{\mathbf{x}}(\boldsymbol{\omega})$ *in* (5.16) *and* (6.4)*, respectively. Consider also the respective variances over all nodes* $var(\omega_0)$ *in* (5.18) *and* $var(\boldsymbol{\omega})$ *in* (6.7)*. If all node-adaptive weights* $\boldsymbol{\omega} = [\omega_1, \ldots, \omega_N]^\top$ *satisfy*

$$\omega_0 \leq \omega_i^2, \quad \text{for } i = 1, 2, \ldots, N \tag{6.8}$$

*then* $var(\boldsymbol{\omega}) \leq var(\omega_0)$.

*Proof.* See Appendix B.2. □

While a reduced variance is useful for signal recovery, it often comes at the expense of an increased bias. To see how sensitive the changes in the two quantities are, we illustrate in Figure 6.1 the bias, the variance, and the MSE for the NI and NA regularizers. We can see that there exists a region for the NA weights where both the variance and the MSE of the NA regularizer are lower compared to those of the NI one without increasing significantly the bias. In Theorem 2, we provide sufficient conditions on $\boldsymbol{\omega}$ to identify this region.

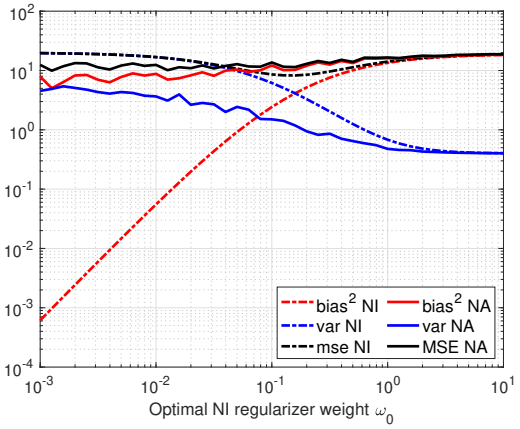



Figure 6.1: Bias-variance trade-off for recovering a graph signal with NA and NI regularizer over and Erdos-Renyì graph and SNR = 0 dB. The detailed settings can be found in section 6.5.1. The node adaptive weights are chosen randomly to satisfy the result in Lemma 5.

**Theorem 2.** *Consider the measurements* $\mathbf{y} = \mathbf{x}^* + \mathbf{n}$ *with desired graph signal* $\mathbf{x}^*$ *and noise* $\mathbf{n}$ *with covariance matrix* $\boldsymbol{\Sigma}$. *Let* $\mathbf{L}$ *be the graph Laplacian with maximum eigenvalue* $\lambda_{\max}(\mathbf{L})$. *Further, define the rank one matrix* $\mathbf{P} := \mathbf{x}^* \mathbf{x}^{*\top} \boldsymbol{\Sigma}^{-1}$ *and let* $\rho$ *be its only non-zero eigenvalue. Define also the rank one matrix* $\boldsymbol{\Gamma} = \mathbf{P}(\mathbf{I}+\mathbf{P})^{-1}$ *and let* $\gamma = \rho(1+\rho)^{-1} \in (0,1)$ *be its only non-zero eigenvalue. Let also* $\mathrm{mse}(\omega_0)$ *[cf.* (5.19)*] and* $\mathrm{mse}(\boldsymbol{\omega})$ *[cf.* (6.5)*] be the mean squared error of the node-invariant estimate* $\hat{\mathbf{x}}(\omega_0)$ *in* (5.16) *and node-adaptive estimate* $\hat{\mathbf{x}}(\boldsymbol{\omega})$ *in* (6.4)*, respectively. Then, if all node-adaptive parameters* $\boldsymbol{\omega} = [\omega_1, \ldots, \omega_N]^\top$ *satisfy*

$$\omega_0 \leq \omega_i^2, \text{ for } i = 1, 2, \ldots, N \tag{6.9a}$$

$$2\gamma \leq \frac{1}{1+\omega_0 \lambda_{max}(\mathbf{L})} + \frac{1}{1+max\{\omega_i^2\}\lambda_{max}(\mathbf{L})} \tag{6.9b}$$

*both the variance and the mean squared error of the node-adaptive regularizer are smaller than those of the node-invariant one; i.e.,* $var(\boldsymbol{\omega}) \leq var(\omega_0)$ *and* $mse(\boldsymbol{\omega}) \leq mse(\omega_0)$.

*Proof.* See Appendix. B.3. □

Condition (6.9b) is easier satisfied when the eigenvalue $\gamma \to 0$, i.e., the signal-to-noise ratio (SNR) is low, or $\rho \to 0$. This indicates that the NA is more powerful in harsher

scenarios. In contrast, when $\gamma \to 1$ and, thus $\rho \to \infty$, i.e., the SNR is high, the condition for the NA regularization to outperform NI one is hard to satisfy. This behavior is intuitively satisfying because we want to use regularizers when dealing with high noise level. When the noise is low, therefore, the variance. A regularizer will only bias the estimate and degrade the reconstruction MSE.

**Corollary 1.** *Under the same settings of Theorem 2, condition*

$$max\{\omega_i^2\} \leq (\rho \lambda_{max}(\mathbf{L}))^{-1}, \text{ for } i = 1, 2, \ldots, N, \tag{6.10a}$$

*guarantees that* (6.9b) *is satisfied.*

*Proof.* See Appendix. B.4. □

Corollary 1 provides an easier condition for the NA parameters $\boldsymbol{\omega}$ compared to those of Theorem 2. In addition, condition (6.10) provides a clearer link between $\boldsymbol{\omega}$ and the SNR. If $\rho \to 0$, hence $\gamma \to 0$, the noise is high and the upper bound for $\omega_i^2$ increases, implying a larger weight on local smoothness is needed. If $\rho \to \infty$, hence $\gamma \to 1$, the noise vanishes and the upper bound for $\omega_i^2$ goes to zero, implying regularization should have little effect. We shall corroborate these observations in section 6.5.1 with numerical results.

## 6.3. IMPLEMENTATION

The matrix inversion in the expression of the optimal estimate $\hat{\mathbf{x}}(\boldsymbol{\omega})$ [cf. (6.4)] makes the node-adaptive regularizer challenging to be implemented on large graphs or in a distributed manner. Fortunately, the inverse can be approximated with a linear cost iteration w.r.t. the number of graph edges $\mathcal{O}(M)$, by leveraging the graph filtering equivalence of (6.4); see, e.g., [6–8]. The key to such a linear cost lies in the sparsity of the node-adaptive operator $\mathbf{S}(\boldsymbol{\omega})$, which coincides with the sparsity of the graph [cf. (6.1)]. Because of this sparsity, the graph signal shifting operation $\mathbf{x}^{(1)} = \mathbf{S}(\boldsymbol{\omega})\mathbf{x} = (\boldsymbol{\omega}\boldsymbol{\omega}^\top \odot \mathbf{L})\mathbf{x}$ has a cost equal to the number of edges $M$. Moreover, this operation is local over the graph, and the $i$-th signal value is given by

$$x_i^{(1)} = \omega_i \sum_{j \in \mathcal{N}_i} A_{ij}(\omega_i x_i - \omega_j x_j). \tag{6.11}$$

By exploring (6.11), we detail next how the NA filter can be implemented using the conjugate gradient method [9] and distributed graph filters [6, 10].

**Centralized.** To solve (6.4) efficiently, we first rephrase it as a linear system

$$(\mathbf{I} + \mathbf{S}(\boldsymbol{\omega}))\hat{\mathbf{x}}(\boldsymbol{\omega}) = \mathbf{y} \tag{6.12}$$

and then employ conjugate gradient [9] to obtain $\hat{\mathbf{x}}(\boldsymbol{\omega})$. For completeness, Algorithm 8 summarizes the required steps. Identifying (6.11) in Steps 7 and 9 of Algorithm 8 and running the conjugate gradient method for $T$ iterations yield a cost of order $\mathcal{O}(TM)$.

**Distributed.** To implement (6.4) distributively with graph filters, we start with a random initialization $\hat{\mathbf{x}}_0$ for estimate $\hat{\mathbf{x}}(\omega)$. At iteration $\tau$, the distributed estimate follows the recursion

$$\hat{\mathbf{x}}_\tau(\boldsymbol{\omega}) = -\mathbf{S}(\boldsymbol{\omega})\hat{\mathbf{x}}_{\tau-1}(\boldsymbol{\omega}) + \mathbf{y}. \tag{6.13}$$

**Algorithm 8** Conjugate gradient method for solving (6.12)

**Input:** $\hat{\mathbf{x}}_{(0)}$, node-adaptive regularizer weights $\boldsymbol{\omega}$, accuracy $\epsilon$, number of iterations $T$
**Initialization:**
$\mathbf{S}(\boldsymbol{\omega}) = \text{diag}(\boldsymbol{\omega})\mathbf{L}\text{diag}(\boldsymbol{\omega})$
$\mathbf{b}_{(0)} = \mathbf{r}_{(0)} = \mathbf{y} - (\mathbf{I} + \mathbf{S}(\boldsymbol{\omega}))\hat{\mathbf{x}}_{(0)}$
$d_{(0)} = d_{new} = \mathbf{r}_{(0)}^{\top}\mathbf{r}_{(0)}$
**while** $\tau < T$ and $d_{new} > \epsilon^2 d_{(0)}$
$c_{(\tau)} = \frac{d_{new}}{\mathbf{b}_{(\tau)}^{\top}(\mathbf{I}+\mathbf{S}(\boldsymbol{\omega}))\mathbf{b}_{(\tau)}}$
$\hat{\mathbf{x}}_{(\tau+1)} = \hat{\mathbf{x}}_{(\tau)} + c_{(\tau)}\mathbf{b}_{\tau}$
$\mathbf{r}_{(\tau+1)} = \mathbf{r}_{(\tau)} - c_{(\tau)}(\mathbf{I} + \mathbf{S}(\boldsymbol{\omega}))\mathbf{b}_{(\tau)}$
$d_{old} = d_{new}$, $d_{new} = \mathbf{r}_{(\tau+1)}^{\top}\mathbf{r}_{(\tau+1)}$
$\mathbf{b}_{(\tau+1)} = \mathbf{r}_{(\tau+1)} + \frac{d_{new}}{d_{old}}\mathbf{b}_{(\tau)}$
$\tau = \tau + 1$
**Output:** $\hat{\mathbf{x}}(\boldsymbol{\omega}) = \hat{\mathbf{x}}_{(\tau+1)}$

Observing (6.11), it is seen that the term $\mathbf{S}(\boldsymbol{\omega})\hat{\mathbf{x}}_{\tau-1}(\boldsymbol{\omega})$ implies that nodes communicate with their neighbors and exchange information about the previous estimate $\hat{\mathbf{x}}_{\tau-1}(\boldsymbol{\omega})$, which has a cost of order $\mathcal{O}(M)$.

When $\boldsymbol{\omega}$ satisfies the spectral norm stability $\|\mathbf{S}(\boldsymbol{\omega})\| < 1$, recursion (6.13) leads to the steady-state ($\tau \to \infty$) estimate

$$\hat{\mathbf{x}}(\boldsymbol{\omega}) \triangleq \lim_{\tau\to\infty} \hat{\mathbf{x}}_{\tau}(\boldsymbol{\omega}) = \sum_{\tau=0}^{\infty} (-\mathbf{S}(\boldsymbol{\omega}))^{\tau}\mathbf{y} = (\mathbf{I} + \mathbf{S}(\boldsymbol{\omega}))^{-1}\mathbf{y}, \tag{6.14}$$

which matches the optimal solution (6.4). Halting (6.13) in $T$ iterations leads to a distributed communication and computational cost of order $\mathcal{O}(TM)$.

## 6.4. WEIGHT DESIGN

In this section, we design the NA weights $\boldsymbol{\omega}$ in a minimum MSE (MMSE) sense. The estimation error between the optimal estimate $\hat{\mathbf{x}}(\boldsymbol{\omega})$ [cf. (6.4)] and the true value $\mathbf{x}^*$ is

$$\mathbf{e} \triangleq \hat{\mathbf{x}}(\boldsymbol{\omega}) - \mathbf{x}^*. \tag{6.15}$$

We can then formulate the optimal design of $\boldsymbol{\omega}$ as solving the following optimization problem

$$\min_{\boldsymbol{\omega} \in \mathbb{R}^N} \quad \mathbb{E}\left\{\|\hat{\mathbf{x}}(\boldsymbol{\omega}) - \mathbf{x}^*\|_2^2\right\}, \tag{6.16}$$

where $\hat{\mathbf{x}}$ is defined as in (6.4).

The inverse relation in $\hat{\mathbf{x}}(\boldsymbol{\omega})$ renders problem (6.16) challenging to solve in its original form. To overcome this challenge, we first propose two methods to solve (6.16) for $\boldsymbol{\omega}$ with true signal information. Each solution relies on adopting different relaxation techniques. Then we use the *min-max* strategy to adapt these two methods to a scenario where only signal bounds are available.

### 6.4.1. ENHANCED PRONY'S METHOD

Given the estimate $\hat{\mathbf{x}}(\boldsymbol{\omega}) = (\mathbf{I} + \mathbf{S}(\boldsymbol{\omega}))^{-1}\mathbf{y}$ and the error $\mathbf{e}$ in (6.15), a typical approach to design parameters in inverse relationships is to consider Prony's modified error [11]

$$\mathbf{e}' \triangleq \mathbf{y} - (\mathbf{I} + \mathbf{S}(\boldsymbol{\omega}))\mathbf{x}^*, \tag{6.17}$$

which is obtained by multiplying both sides of (6.15) by $(\mathbf{I} + \mathbf{S}(\boldsymbol{\omega}))$. Albeit not equivalent to the true error $\mathbf{e}$, minimizing the modified error $\mathbf{e}'$ is easier due to the linear relationship in $\boldsymbol{\omega}$ and the resulting performance is often satisfactory [11]. The NA weights that minimize the error $\mathbf{e}'$ in (6.17) can be obtained by solving

$$\begin{aligned} \min_{\boldsymbol{\omega} \in \mathbb{R}^N} \quad & \mathbb{E}\left\{\|\mathbf{y} - (\mathbf{I} + \mathbf{S}(\boldsymbol{\omega}))\mathbf{x}^*\|_2^2\right\} \\ \text{s.t.} \quad & \omega_i^2 \geq \omega_0^*, \text{ for } i = 1, \ldots, N \end{aligned} \tag{6.18}$$

where the constraint imposes that all entries of $\boldsymbol{\omega}$ satisfy the theoretical result of Lemma 5. Besides guaranteeing a smaller variance, we have observed that the constraints result in a lower MSE.

The quadratic relation in the optimization variable $\boldsymbol{\omega}$ in $\mathbf{S}(\boldsymbol{\omega})$ [cf. (6.1)] makes problem (6.18) non-convex. To obtain $\boldsymbol{\omega}$, we follow a two step approach. First we define a positive semi-definite rank-one matrix $\boldsymbol{\Omega} \triangleq \boldsymbol{\omega}\boldsymbol{\omega}^\top$ and solve (6.18) w.r.t the new variable $\boldsymbol{\Omega}$. Then, we find a vector estimate $\hat{\boldsymbol{\omega}}$ by performing a rank-one approximation of the obtained matrix [12].

Rewriting (6.18) w.r.t. the new variable $\boldsymbol{\Omega}$, we obtain

$$\begin{aligned} \min_{\boldsymbol{\Omega} \in \mathscr{S}_+^{N \times N}} \quad & \operatorname{tr}\left\{(\boldsymbol{\Omega} \odot \mathbf{L})^2 \mathbf{x}^* \mathbf{x}^{*\top}\right\} \\ \text{s.t.} \quad & \Omega_{ii} \geq \omega_0^*, \text{ for } i = 1, \ldots, N \end{aligned} \tag{6.19}$$

where the derivation of the cost function is reported in Appendix B.5. In (6.19), we follow [12] and drop the non-convex constraint $\operatorname{rank}(\boldsymbol{\Omega}) = 1$ from its definition. Then, problem (6.19) becomes convex and solvable with off-the-shelf tools [13, 14]. We observe that in our experiments, shown later, in all instances the returned solution from (6.19) is a rank-one matrix.

Given then $\boldsymbol{\Omega}^*$ from (6.19), the node-adaptive weight vector $\boldsymbol{\omega}^*$ is equal to the eigenvector of $\boldsymbol{\Omega}^*$ with the largest eigenvalue, multiplied with the square root of the eigenvalue. However, more sophisticated rank-one approximations are also possible [12].

### 6.4.2. SEMI-DEFINITE RELAXATION

Despite its simplicity, Prony's method does not directly relate to the true error $\mathbf{e}$ in (6.15). Working with the modified error might be viable when the signal-to-noise ratio (SNR) on $\mathbf{y}$ is high but it might lead to a degraded performance when the SNR is low since the uncertainty in $\mathbf{y}$ increases. To overcome the latter, we here propose an optimization problem relying on semi-definite relaxation when minimizing the true error as in (6.16). This approach follows again a two-step procedure: first, we formulate (6.16) w.r.t. the matrix variable $\mathbf{H}(\boldsymbol{\Omega}) = (\mathbf{I} + \boldsymbol{\Omega} \odot \mathbf{L})^{-1}$ with $\boldsymbol{\Omega} := \boldsymbol{\omega}\boldsymbol{\omega}^\top$, and then we obtain $\boldsymbol{\Omega}$ from $\mathbf{H}(\boldsymbol{\Omega})$

by inversion. Finally, $\boldsymbol{\omega}^*$ can be directly extracted from $\boldsymbol{\Omega}^*$ by rank-one approximation [12].

To start, let us recall the node-adaptive estimate $\hat{\mathbf{x}}(\boldsymbol{\omega}) = (\mathbf{I}+\boldsymbol{\omega}\boldsymbol{\omega}^\top \odot \mathbf{L})^{-1}\mathbf{y}$ and rewrite (6.16) w.r.t. $\mathbf{H}(\boldsymbol{\Omega})$ as

$$\begin{aligned} \min_{\boldsymbol{\Omega},\, \mathbf{H}(\boldsymbol{\Omega}) \in \mathscr{S}_+^{N\times N}} \quad & \mathbb{E}\left\{\|\mathbf{H}(\boldsymbol{\Omega})\mathbf{y}-\mathbf{x}^*\|_2^2\right\} \\ \text{s.t.} \quad & \mathbf{H}(\boldsymbol{\Omega}) = (\mathbf{I}+\boldsymbol{\Omega}\odot\mathbf{L})^{-1}, \\ & \text{rank}(\boldsymbol{\Omega}) = 1., \\ & \Omega_{ii} \geq \omega_0^*, \text{ for } i = 1,\ldots,N. \end{aligned} \tag{6.20}$$

where the last constraint is again following Lemma 5. The cost function in (6.20) can be expanded further as (cf. Appendix. B.6)

$$\mathbb{E}\left\{\|\mathbf{H}(\boldsymbol{\Omega})\mathbf{y}-\mathbf{x}^*\|_2^2\right\} = \text{tr}\{(\mathbf{H}^2(\boldsymbol{\Omega})) - 2\mathbf{H}(\boldsymbol{\Omega})+\mathbf{I})\mathbf{x}^*\mathbf{x}^{*\top} + \mathbf{H}^2(\boldsymbol{\Omega})\boldsymbol{\Sigma}\},$$

which depends on both the signal and noise covariance. Problem (6.20) presents two non-convex constraints to be addressed: the inverse relationship $\mathbf{H}(\boldsymbol{\Omega}) = (\mathbf{I}+\boldsymbol{\Omega}\odot\mathbf{L})^{-1}$ and the rank-one constraint $\text{rank}(\boldsymbol{\Omega}) = 1$. We address the former, by leveraging *semi-definite relaxation* and writing the constraint in its positive semi-definite convex form [12]

$$\mathbf{H}(\boldsymbol{\Omega}) - (\mathbf{I}+\boldsymbol{\Omega}\odot\mathbf{L})^{-1} \succeq \mathbf{0}. \tag{6.21}$$

Further, since $\mathbf{L}$, $\boldsymbol{\Omega}$, and $\mathbf{I}+\boldsymbol{\Omega}\odot\mathbf{L}$ are positive semi-definite matrices, we can use the Schur complement (see Appendix.B.1) to reformulate (6.21) into a convex linear matrix inequality

$$\begin{pmatrix} \mathbf{I}+\boldsymbol{\Omega}\odot\mathbf{L} & \mathbf{I} \\ \mathbf{I} & \mathbf{H}(\boldsymbol{\Omega}) \end{pmatrix} \succeq \mathbf{0}.$$

Regarding the non-convexity of the rank-one constraint, we can relax it as in (6.19). With these relaxation techniques, we can rewrite problem (6.20) as a convex problem

$$\begin{aligned} \min_{\boldsymbol{\Omega},\, \mathbf{H}(\boldsymbol{\Omega}) \in \mathscr{S}_+^{N\times N}} \quad & \mathbb{E}\left\{\|\mathbf{H}(\boldsymbol{\Omega})\mathbf{y}-\mathbf{x}^*\|_2^2\right\} \\ \text{s.t.} \quad & \begin{pmatrix} \mathbf{I}+\boldsymbol{\Omega}\odot\mathbf{L} & \mathbf{I} \\ \mathbf{I} & \mathbf{H}(\boldsymbol{\Omega}) \end{pmatrix} \succeq \mathbf{0}, \\ & \Omega_{ii} \geq \omega_0^*, \text{ for } i = 1,\ldots,N. \end{aligned} \tag{6.22}$$

After obtaining $\mathbf{H}^*(\boldsymbol{\Omega})$ from (6.22), we can find $\boldsymbol{\Omega}$ based on its inverse relation

$$\mathbf{H}^*(\boldsymbol{\Omega})(\mathbf{I}+\boldsymbol{\Omega}\odot\mathbf{L}) = (\mathbf{I}+\boldsymbol{\Omega}\odot\mathbf{L})\mathbf{H}^*(\boldsymbol{\Omega}) = \mathbf{I},$$

by solving the following least squares problem

$$\min_{\boldsymbol{\Omega} \in \mathscr{S}_+^{N\times N}} \quad \|\mathbf{H}^*(\boldsymbol{\Omega})(\mathbf{I}+\boldsymbol{\Omega}\odot\mathbf{L})-\mathbf{I}\|_2^2 + \|(\mathbf{I}+\boldsymbol{\Omega}\odot\mathbf{L})\mathbf{H}^*(\boldsymbol{\Omega})-\mathbf{I}\|_2^2. \tag{6.23}$$

Given then the matrix $\boldsymbol{\Omega}^*$ from (6.23), we extract $\boldsymbol{\omega}^*$ by a rank-one approximation of the matrix [12]. With the obtained NA weights $\boldsymbol{\omega}^*$, we subsequently build the filter

$\mathbf{H}(\boldsymbol{\omega}^*) = (\mathbf{I}+\boldsymbol{\omega}^*\boldsymbol{\omega}^{*\top}\odot\mathbf{L})^{-1}$ and obtain the estimate $\hat{\mathbf{x}}(\boldsymbol{\omega}^*)$ as in (6.4). The main advantage of the semidefinite relaxation (SDR) approach (6.22)-(6.23) is that it focuses directly on the true error (6.15) rather than the modified error (6.17). However, semidefinite relaxation techniques are only applicable to medium-sized graphs, i.e., graphs up to a few hundreds of nodes.

### 6.4.3. *min-max* METHOD

As it follows from (6.18) and (6.20) both Prony's method and the SDR method require knowledge of signal $\mathbf{x}^*$ to design the parameter vector $\boldsymbol{\omega}$. This is possible in a data-driven fashion under the condition that the test data has a similar distribution as the training data used for designing $\boldsymbol{\omega}$. In this section, we depart from this assumption and propose a design method for $\boldsymbol{\omega}$ that is independent of $\mathbf{x}^*$ and only requires some side information about the signal such as bounds on the signal evolution. The latter is simpler to get from a small set of data or physical considerations.

Consider signal $\mathbf{x}^*$ has an evolution bounded in the interval $[\mathbf{x}_\mathrm{l},\mathbf{x}_\mathrm{u}]$. Here, $\mathbf{x}_\mathrm{l}$ gives an entry-wise lower bound and $\mathbf{x}_\mathrm{u}$ an entry-wise upper bound. We can then design the parameter vector $\boldsymbol{\omega}$ for the estimator $\hat{\mathbf{x}}(\boldsymbol{\omega})$ [cf. (6.4)] as the one that minimizes the MSE of the worst-case scenario, i.e.,

$$\begin{aligned} \min_{\boldsymbol{\omega}} \quad & \max_{\mathbf{x}^*} \mathbb{E}\left\{\|\hat{\mathbf{x}}(\boldsymbol{\omega})-\mathbf{x}^*\|_2^2\right\} \\ \text{s.t.} \quad & \hat{\mathbf{x}}(\boldsymbol{\omega}) = (\mathbf{I}+\mathbf{S}(\boldsymbol{\omega}))^{-1}\mathbf{y}, \\ & \mathbf{x}_\mathrm{l} \le \mathbf{x}^* \le \mathbf{x}_\mathrm{u}. \end{aligned} \tag{6.24}$$

Problems of the form (6.24) are known as *min-max* problems. The inner maximization seeks for a signal $\mathbf{x}^*$ that leads to the worst MSE performance, while the outer minimization finds the parameter $\boldsymbol{\omega}$ that minimizes the worst MSE among all possible choices. The two constraints basically impose $\boldsymbol{\omega}$ to be a node-adaptive regularizer and the signal to be bounded.

Differently from problems (6.18) and (6.20), signal $\mathbf{x}^*$ is now an optimization variable in (6.24) and only $\mathbf{x}_\mathrm{l}$ and $\mathbf{x}_\mathrm{u}$ are needed. There are efficient methods to solve min-max problems of the form (6.24) such as iterative first order methods [15] or gradient descent-ascent[16]. In the sequel, we detail how (6.24) specializes to Prony's and SDR designs.

Following the same rationale as in (6.19), we can write the min-max enhanced Prony's method as

$$\begin{aligned} \min_{\boldsymbol{\Omega}\in\mathscr{S}_+^{N\times N}} \quad & \max_{\mathbf{x}^*} \operatorname{tr}\{(\boldsymbol{\Omega}\odot\mathbf{L})^2\mathbf{x}^*\mathbf{x}^{*\top}\} \\ \text{s.t.} \quad & \mathbf{x}_\mathrm{l} \le \mathbf{x}^* \le \mathbf{x}_\mathrm{u}, \\ & \Omega_{ii} \ge \omega_0^*, \text{ for } i=1,\ldots,N. \end{aligned} \tag{6.25}$$

Since $\boldsymbol{\Omega}\odot\mathbf{L}$ is positive semidefinite, the cost function is quadratic (convex) over the inner optimization variable $\mathbf{x}^*$. Further, it can be shown that the maximizer of the inner problem is at the boundaries, either $\mathbf{x}^*=\mathbf{x}_\mathrm{l}$ or $\mathbf{x}^*=\mathbf{x}_\mathrm{u}$. To solve the outer minimization problem w.r.t. $\boldsymbol{\Omega}$, we proceed similarly as for problem (6.19). Moreover, we can rephrase and solve the min-max version of the SDR problem (6.22) since the cost function for the latter is also convex in $\mathbf{x}^*$. Here again, we optimize the true error [cf. (6.15)] at the price of a higher computational complexity.

## 6.5. NUMERICAL RESULTS

We compare the performance of the proposed design schemes with state-of-the-art alternatives and illustrate the different trade-offs inherent to node-adaptive regularization through synthetic and real data from the Molene[1] and the NOAA[2] data sets. We measure the recovering accuracy between and estimate $\hat{\mathbf{x}}$ and the true signal $\mathbf{x}^*$ through the normalized mean squared error NMSE $= \|\hat{\mathbf{x}} - \mathbf{x}^*\|_2^2 / \|\mathbf{x}^*\|_2^2$. In these simulations, we used the GSP box [17] and CVX [13].

### 6.5.1. SYNTHETIC DATA

In the first set of experiments, we consider synthetic Erdos-Renyì graphs of $N = 50$ nodes and a link formation probability of 0.5. We generated a synthetic graph signal $\mathbf{x}^*$, whose graph Fourier transform is one in the first 20 coefficients and zero elsewhere [5]. This signal is called a bandlimited graph signal and varies smoothly over the graph; hence, it fits the Tikhonov regularization problem (5.15). We corrupted the signal with a zero-mean Gaussian noise with variance $\sigma_n^2$, i.e., with a signal-to-noise ratio

$$\text{SNR} = \|\mathbf{x}^*\|_2^2 / (N\sigma_n^2).$$

We average the performance over 50 different graphs and 100 noise realizations leading to a total of 5000 Monte-Carlo runs. This scenario is also the one used to produce the results in Figure 6.1, where we showed that the node-adaptive regularizer can achieve both a lower variance and MSE.

We first evaluate Prony's method and the SDR method when the true signal is known. Our rationale is to avoid biases induced by a training set or a performance degradation due to selecting the worst-case scenarios. We address the latter in the subsequent section for real data. The specific approaches we consider are:

i) The benchmark node-invariant regularizer with optimal weight $\omega_0^* = \mathcal{O}(\sqrt{\frac{\theta}{\lambda_2 \lambda_N}})$, where $\theta = \sqrt{\frac{1}{\text{SNR}}}$, and $\lambda_2$, $\lambda_N$ are the smallest and the largest non-zero eigenvalues of $\mathbf{L}$, respectively [18].

ii) The naive node-adaptive regularizer where the weight vector $\boldsymbol{\omega}$ is chosen randomly to satisfy Lemma 5; i.e., $\omega_i = \sqrt{\omega_0^* + \omega_0^* \cdot c_i}$ for $i = 1, \ldots, N$, where $c_i$ is uniformly distributed in $[0, 1]$.

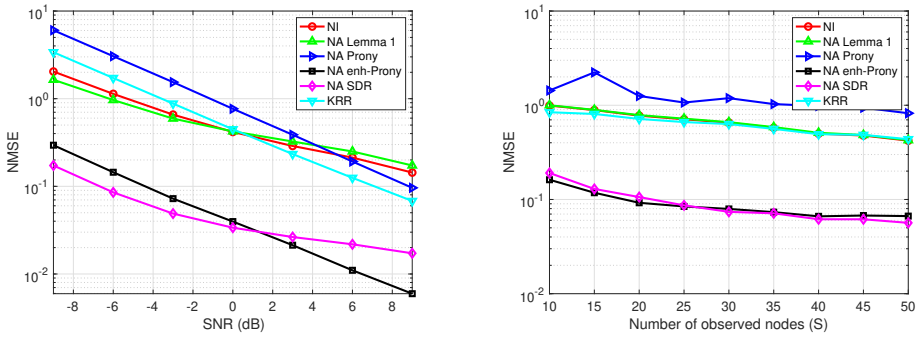
iii) Prony's node-adaptive design, where we did not enforce the constraint of Lemma 5 in problem (6.19).

iv) The enhanced Prony method [cf. (6.19)].

v) The SDR node-adaptive design [cf. (6.22), (6.23)].

[1] Raw data available at
https://donneespubliques.meteofrance.fr/donneeslibres/Hackathon/RADOMEH.tar.gz

[2] Raw data available at
https://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/climate-normals/1981-2010-normals-data




(a) Denoising.

(b) Interpolation.

Figure 6.2: (a) NMSE of different methods as a function of the SNR over an Erdos-Renyì graph. The true graph signal is bandlimited to the first 20 graph frequencies. (b) Interpolation performance of the different methods as a function of number of observed nodes with SNR = 0 dB over an Erdos-Renyì graph. The true graph signal is bandlimited to the first 20 graph frequencies.

vi) The diffusion kernel ridge regression (KRR) with best-performing parameters $\sigma^2_{\text{KRR}} = 1$ and $\mu_{\text{KRR}} = 10^{-4}$ [3].

Figure 6.2a shows the NMSE recovery performance of the different methods as a function of the SNR from $-9$ dB to 9 dB. First, we observe that the proposed enhanced Prony design and the SDR reduce the NMSE by one order of magnitude compared to the optimal node-invariant approach and the KRR. In fact, as we anticipated in Figure 6.1, even the naive random node-adaptive regularizer achieves a comparable performance with these competitive alternatives, ultimately, highlighting the potential of the NA regularizer for graph signal recovery. Finally, we remark the need of the theoretical Lemma 5 result in Prony's problem (6.19), which reduces the MSE by one order of magnitude. We attribute the latter to the fact that Prony's approaches focus on the modified error rather than the true one. This is also evidenced by the comparison with the SDR technique, where the enhanced Prony's method has a worse NMSE for lower SNRs; i.e., where considering the true error is more effective to deal with large noise.

Next, we evaluate the performance of the different methods for interpolating missing values. We consider noisy observations from the random subset $\mathcal{M} \in \{10, 15, 20, \ldots, 50\}$ with an SNR of 0 dB to show the resistance of node-adaptive regularization. From Figure 6.2b, we observe again the superior performance of the enhanced Prony and SDR method and both of them perform close to each other. This is due to the relaxation loss in SDR method and modified error in enhanced Prony's method. Under different SNR settings, their performance vary slightly. On the contrary, the naive weight setting and the simple Prony method offer a similar performance as the benchmark NI regularization. When the number of observed nodes is full, the NMSE results correspond to the denoising results.

In the sequel, we experiment over data collected from real scenarios. We omit the experimental results of the node-adaptive design based on ground truth signal, since they

behave the same as for the synthetic data. Instead, we consider the earlier mentioned data-driven and *min-max* scenarios.

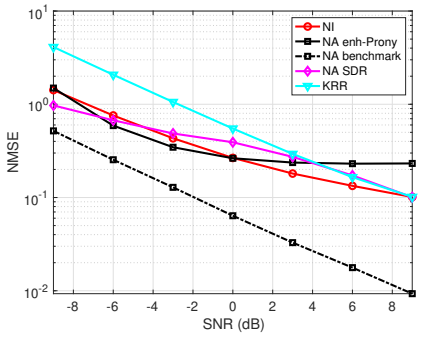
### 6.5.2. MOLENE DATA SET

The Molene data set comprises $T = 744$ hourly temperature measurements collected in January 2014 from $N = 32$ weather stations in the region of Brest, France. We treat each weather station as a node of a graph and build a geometric distance graph in which each node is connected to five nearest neighbors. The weight of edge $(i, j)$ is $A_{ij} = \exp\{-5d^2(i, j)\}$, where $d(i, j)$ is the Euclidean distance between stations $i$ and $j$. After removing the mean across space and time, we can view every temporal snapshot of the temperature as a graph signal. Among the six approaches listed in the former section, we omit the naive node-adaptive regularizer and the simple Prony's method. This avoids overcrowded plots since their performance trend is similar to that observed with synthetic data. For the KRR, we set $\sigma^2_{\text{KRR}} = 5$ to have the best performance for this method. The performance criterion is again the NMSE averaged over all the 744 graph signals. The measurements are assumed to be the true signal and we artificially add noise. We consider 50 noise realizations per signal.

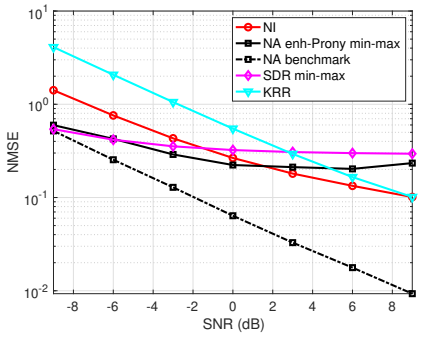
To show how the different methods behave in the ideal and in a more practical setting, we considered two scenarios. First, a data-driven scenario, where half of the temporal data are used to learn the node adaptive weights. Specifically, we used those data to compute the average of $\mathbf{x}^*\mathbf{x}^{*\top}$ which can be used in the enhanced Prony design [cf. (6.19)] and to solve the SDR problem [cf.(6.22)]. The remaining half graph signals are used for testing. Note that in the SDR method of this scenario, to make use of the noise covariance matrix in the objective function of (6.22), as derived in B.6, we have to design the parameters for each of the rest testing data, since we have to compute the noise level with given SNR for each data. To avoid doing this design for each daily data, we use a one-day sample of the whole data to compute the noise covariance matrix needed in the SDR method, which shall give worse results as it might achieve. Second, the min-max method was tested where only signal lower and upper bounds are known from the data. These are selected as the lowest and highest temperature records in this dataset. For both scenarios, we consider the NMSE denoising performance as a function of the SNR in the interval $[-9\text{ dB}, 9\text{ dB}]$. Figure 6.3 shows the results. The enhanced Prony's method with true signals is used as a benchmark for comparison.

From Figures 6.3a, we see that the NA methods are not superior compared with NI method. We contribute this to the inappropriate training strategy. Specifically, we observe that the enhanced Prony method degrades substantially and achieves an NMSE similar to the NI regularizer. In the large SNR regime, it gets worse than the NI regularizer. This is also because when there is not too much noise present, the NA regularizer will instead bias the estimate without the true signal information.

When the signal variation bound is available, the performance of the NA approaches degrades by approximately one order of magnitude. Figure 6.3b further shows that the NA algorithms still perform better than the state-of-the-art in the low SNR regime although the only information is the signal variation range. We attribute the saturation in the high SNR regime to the lack of information needed for designing the NA weights; i.e., the NA regularizer will impose a stronger bias on the solution that is not needed to

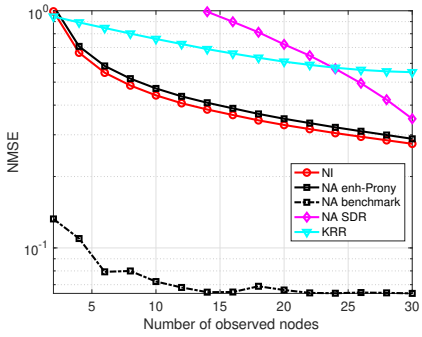



(a) Data-driven.

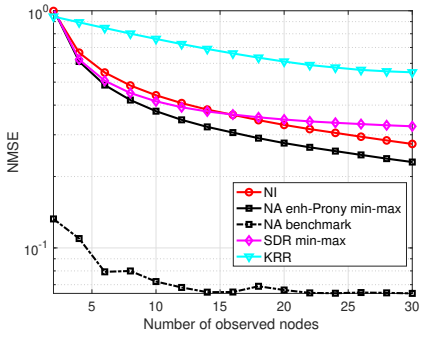



(b) Min-max.

Figure 6.3: NMSE denoising performance of the different methods as a function of the SNR in the Molene data set. (a) Data-driven scenario. Half of the data are used to estimate the node-adaptive parameters. (b) Min-max scenario. Only the signal evolution bounds are needed.




(a) Data-driven.




(b) Min-max.

Figure 6.4: NMSE interpolation performance of the different methods as a function of the number of sampled nodes in the Molene data set. The signal-to-noise ratio is SNR = 0 dB. (a) Data-driven scenario. Half of the data are used to estimate the signal covariance matrix. (b) Min-max scenario. Only the signal evolution bounds are needed.

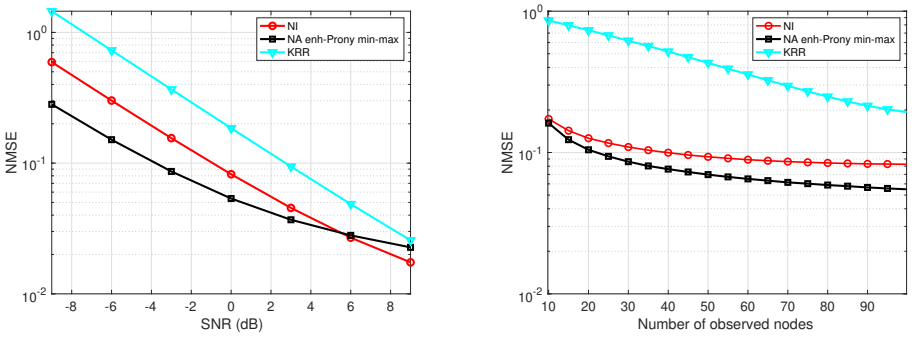
denoise the signal in the high SNR regime. Note that in the *min-max* strategy results of Figure 6.3b, we report only the better result out of the two generated by using lower and upper bounds separately.

We then evaluate the interpolation performance of different methods. We consider noisy observations at the nodes $\mathscr{M} \in \{2,4,6,\ldots,28,30\}$ and SNR = 0 dB. The results are shown in Figure 6.4. For the data-driven case in Figure 6.4a, Prony's method degrades significantly with a performance worse than the NI regularizer. This could be caused by the inappropriate training and over-fitting of the training data. On the other hand, the SDR is not obtaining satisfactory results unless the observations are collected from all of

the nodes. This is because the obtained estimate of $\mathbf{x}^*\mathbf{x}^{*\top}$ does not apply on the SDR method for the interpolation case. In the *min-max* scenario, shown in Figure 6.4b, we only make use of the signal bounds and a consistently better performance is obtained by Prony's method, while the SDR method behaves close to the NI regularizer.

In practical situations, with a proper prior information of graph signals, we recommend to design the NA regularizer using the *min-max* strategy. From the above experimental results, we see that in harsher situations with a lot of noise present, based on the *min-max* strategy, the NA regularizer, for denoising and interpolation, will behave better than its NI and KRR counterparts. In the next section, we corroborate the ability of the *min-max* strategy on another real-world dataset.

### 6.5.3. NOAA DATA SET




(a) Denoising.

(b) Interpolation.

Figure 6.5: (a) Denoising performance of the enhanced Prony's method as a function of SNR with different methods in *min-max* scenario. (b) Interpolation performance as a function of the number of observed nodes. The signal-to-noise ratio is SNR = 0 dB.

The NOAA data set comprises $T = 8759$ hourly temperature measurements collected across the continental U.S. in 2010 from $N = 109$ weather stations. Following [19], we treated each station as a node of a seven nearest neighbor graph based on geographical distances. We treat the signals in the same way as we did before. We measure again the performance with the NMSE averaged over all the 8759 signals and 50 noise realizations per signal. The parameters of all methods are the same as in the former section.

Figure 6.5 shows directly the results for the min-max method applied to denoising and interpolation. These results correspond to the experiments conducted on Molene weather data. We see again, from Figure 6.5a, the improved performance of the proposed NA regularizer over other alternatives. In specific, we see a 3 dB SNR improvement for a fixed NMSE in the enhanced Prony method, which is even larger with low SNR values. For interpolation purposes, the number of the observed nodes are set as $\mathscr{M} \in \{10, 15, 20, \ldots, 90, 95\}$. From Figure 6.5b, we observe that with the NA regularizer, we consistently have smaller NMSE compared to the NI regularizer and KRR. This improved performance gets more noticeable when the number of observed nodes is larger, since

the node-adaptive regularizer has more information to exploit the local signal behavior to find the missing values.

## REFERENCES


[1] Y.-X. Wang, J. Sharpnack, A. J. Smola, and R. J. Tibshirani, *Trend filtering on graphs,* The Journal of Machine Learning Research **17**, 3651 (2016).

[2] S. Chen, R. Varma, A. Singh, and J. Kovačević, *Representations of piecewise smooth signals on graphs,* in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, 2016) pp. 6370–6374.

[3] D. Romero, M. Ma, and G. B. Giannakis, *Kernel-based reconstruction of graph signals,* IEEE Transactions on Signal Processing **65**, 764 (2017).

[4] A. Venkitaraman, S. Chatterjee, and P. Händel, *Multi-kernel Regression For Graph Signal Processing,* arXiv e-prints , arXiv:1803.04196 (2018), arXiv:1803.04196 [stat.ML] .

[5] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, *The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains,* IEEE Signal Processing Magazine **30**, 83 (2013).

[6] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, *Autoregressive moving average graph filtering,* IEEE Transactions on Signal Processing **65**, 274 (2017).

[7] M. Coutino, E. Isufi, and G. Leus, *Advances in distributed graph filtering,* IEEE Transactions on Signal Processing **67**, 2320 (2019).

[8] J. Liu, E. Isufi, and G. Leus, *Filter design for autoregressive moving average graph filters,* IEEE Transactions on Signal and Information Processing over Networks **5**, 47 (2019).

[9] J. R. Shewchuk, *An introduction to the conjugate gradient method without the agonizing pain,* (August 4, 1994).

[10] E. Isufi, *Graph-time signal processing: Filtering and sampling strategies,* (2019).

[11] M. H. Hayes, *Statistical Digital Signal Processing and Modeling,* 1st ed. (John Wiley & Sons, Inc., New York, NY, USA, 1996).

[12] Z. Luo, W. Ma, A. M. So, Y. Ye, and S. Zhang, *Semidefinite relaxation of quadratic optimization problems,* IEEE Signal Processing Magazine **27**, 20 (2010).

[13] M. Grant and S. Boyd, *CVX: Matlab software for disciplined convex programming, version 2.1,* `http://cvxr.com/cvx` (2014).

[14] D. P. Bertsekas, *Convex optimization theory* (Athena Scientific, 2009).

[15] M. Nouiehed, M. Sanjabi, T. Huang, J. D. Lee, and M. Razaviyayn, *Solving a class of non-convex min-max games using iterative first order methods, Advances in Neural Information Processing Systems 32*, , 14905 (2019).

[16] T. Lin, C. Jin, and M. I. Jordan, *On gradient descent ascent for nonconvex-concave minimax problems,* ArXiv **abs/1906.00331** (2019).

[17] N. Perraudin, J. Paratte, D. Shuman, L. Martin, V. Kalofolias, P. Vandergheynst, and D. K. Hammond, *GSPBOX: A toolbox for signal processing on graphs,* , 1 (2016), arXiv:arXiv:1408.5781v2 .

[18] P. Chen and S. Liu, *Bias-variance tradeoff of graph laplacian regularizer,* IEEE Signal Processing Letters **24**, 1118 (2017).

[19] J. Mei and J. M. F. Moura, *Signal processing on graphs: Causal modeling of unstructured data,* IEEE Transactions on Signal Processing **65**, 2077 (2017).

# 7

# NODE-ADAPTIVE REGULARIZATION BASED GRAPH SIGNAL SAMPLING

Graph signal sampling is a critical task in GSP, the goal of which is to only sample measurements from a subset (with a fixed cardinality of $K$) of graph nodes such that the graph signal can be reconstructed with a good performance. The optimal design of this subset is the main work in graph signal sampling. This is important in many applications, e.g., signal interpolation over graphs where the number of nodes to observe is limited, how to deploy sensors in a sensor networks, or which weather stations to observe across the country.

This chapter focuses on graph nodes subset selection based on NA graph signal regularization. The idea is to explore the property of the NA regularizer, and then design a greedy sampling strategy. This chapter presents an application of NA graph signal regularization in the graph signal sampling.

## 7.1. INTRODUCTION

In this section, we first review the recent development in graph signal sampling. Then, we provide the knowledge required for the following chapters.

### 7.1.1. LITERATURE REVIEW

Since graph signal sampling is strongly related to the sensor selection problem [1], traditional convex optimization based methods can be used in graph signal sampling. These methods focus on dealing with the Boolean and nonconvex node selection vector. Usually, they first relax the Boolean selection constraints to make the problem convex, then build the selection vector by different rounding approaches, for example, typically taking the largest $K$ values [1, 2], or Fedorov's exchange algorithm [3].

Depends on the requirements, the cost functions can be different experimental design criteria, such as the MSE, the worst case variance or log-det of the error covariance matrix (which are called A, E, D-optimality), and so on [2, 4]. Some authors use

the so-called frame potential as the objective function [5, 6]. In [7], randomized techniques by pipaging or proportional rounding are proposed to improve the optimality for D-experimental design. Authors of [8] propose a greedy swapping for rounding for rigorous objectives, which can achieve a near-optimality comparable or slightly worse than greedy method in a more efficient way.

To reduce the high computational cost of the convex relaxation based algorithms, greedy methods based sampling techniques are proposed. The idea is to sample one node from the available nodes greedily at each iteration so to achieve the biggest performance improvement. The work in [9] provided the near-optimal guarantee for greedy based graph signal sampling, where a bandlimited graph signal model is considered, requiring the eigendecomposition of the graph Laplacian. In [10], a Tikhonov solution based greedy sampling scheme is used for semi-supervised learning task over graphs, which does not require an eigendecomposition. The submodularity of the Tikhonov estimate MSE expression is well studied, which is critical for greedy algorithm. In [11], stochastic version of greedy methods is used to speed up the greedy sampling algorithms. This is achieved by considering a random subset of the remaining nodes at each iteration. In general, greedy sampling can achieve a suboptimality of at least 67% when the cost function is submodular [9].

Beyond the above approaches, some researchers leverage graph properties to scale sampling strategies to large scales. In [12], the author first partition the graph, and then do graph signal sampling in each small clusters, which allows to extend the sampling to a large scale. Since the most traditional signal model for sampling rely on the eigendecomposition of the graph Laplacian, the authors in [13] proposed to use the Tikhonov solution to interpolate the graph signal, then use the techniques of truncated Neumann series to further approximate the inverse operation involved. Later, the same authors also used Gershgorin theorem to convert the Tikhonov solution based sampling problem as a set cover problem [14], which significantly improved the performance and reduce the computational complexity [15]. Both methods avoid the eigen-decomposition of the graph Laplacian. The work in [16] provide a recent survey of graph signal sampling.

In this chapter, we are also interested in avoiding the eigen-decomposition, and applying our NA regularization to replace the Tikhonov regularization. In addition, we will consider greedy based sampling, since it has acceptable computational cost compared to convex relaxations.

### 7.1.2. BASICS

Consider a graph signal $\mathbf{x} \in \mathbb{R}^N$ defined over a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The task is to estimate $\mathbf{x}$ from its sampled noisy version $\mathbf{y}_{\mathcal{S}} \in \mathbb{R}^K$, where $\mathcal{S} \subseteq \mathcal{V}$ is the sampling subset. We consider the linear measurement model

$$\mathbf{y}_{\mathcal{S}} = \mathbf{C}\mathbf{x} + \mathbf{n}, \tag{7.1}$$

where $\mathbf{C}$ is the binary sampling matrix of size $K \times N$ and $\mathbf{n}$ is the noise with covariance matrix $\mathbf{\Sigma}_n$. Note that $\mathbf{C}^\top\mathbf{C} = \text{diag}(\mathbf{c})$, where $c_i = 1$ if node $i$ is sampled; $c_i = 0$ otherwise, and we call the vector $\mathbf{c}$ as the sampling vector. We assume that the noise and the signal are independent.

If one is required to sample only $K$ nodes, the graph signal sampling has the following general problem formulation

$$\begin{aligned} &\text{find } \mathbf{c} \text{ from } f(\mathbf{c}), \\ &\text{subject to} \quad \|\mathbf{c}\|_0 = K, \\ &\qquad\qquad \mathbf{c} \in \{0,1\}^K. \end{aligned} \tag{7.2}$$

The objective function $f(\mathbf{c})$ can have different forms under different models. For example, as done in [1], the objective functions based on the sparse graph spectral model require eigendecomposition of the graph Laplacian, which is not efficient when the graph is large. In this chapter, we consider the graph signal estimate e.g., Tikhonov estimate, based sampling under deterministic and random signal cases, i.e., the Bayesian and non-Bayesian frameworks.

In the following, we introduce two definitions of the set functions which are used in the greedy algorithms.

**Definition 7.** *A set function $f: 2^{\mathscr{X}} \to \mathbb{R}$ is monotone decreasing if $f(\mathscr{S}) < f(\mathscr{T})$ for all $\mathscr{S} \subseteq \mathscr{T} \subset \mathscr{X}$.*

**Definition 8** (Submodular function)**.** *A set function $f: 2^{\mathscr{X}} \to \mathbb{R}$ is submodular if*

$$f(\mathscr{S} \cup j) - f(\mathscr{S}) > f(\mathscr{T} \cup j) - f(\mathscr{T}) \tag{7.3}$$

*for all subsets $\mathscr{S} \subseteq \mathscr{T} \subset \mathscr{X}$ and $j \in \mathscr{X} \backslash \mathscr{T}$.*

If a set function follows $-f$ is submodular, we say $f$ is supermodular. Submodularity is a diminishing return property where adding an element to a smaller set gives a larger gain than adding it to a greater set. The maximization of monotone increasing submodular functions is still NP-hard, but as shown in [17], greedy heuristic can be used to obtain a solution that is proven to have an approximation ratio of $1-1/e \approx 63\%$.

**Definition 9** (Stieltjes matrix)**.** *A Stieltjes matrix is a real symmetric positive definite matrix with non-positive off-diagonal entries.*

The graph Laplacian is a Stieltjes matrix, which will be used in later the supermodularity study of the functions we use.

### 7.1.3. INTERPOLATION VIA TIKHONOV REGULARIZATION

The Tikhonov regularization in interpolation problems can be formulated as

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\mathbf{y}_{\mathscr{S}} - \mathbf{C}\mathbf{x}\|^2 + \omega_0 \mathbf{x}^\top \mathbf{L}\mathbf{x}. \tag{7.4}$$

The analytical solution can be found through setting its gradient as zero, i.e.,

$$\hat{\mathbf{x}} = (\mathbf{C}^\top \mathbf{C} + \omega_0 \mathbf{L})^{-1} \mathbf{C}^\top \mathbf{y}_{\mathscr{S}}. \tag{7.5}$$

When the number of samples $K \geq 1$, term $\mathbf{C}^\top\mathbf{C} + \omega_0\mathbf{L}$ is invertible and positive definite, which can be seen as follows. For the graph Laplacian $\mathbf{L}$ of a connected graph, the all-one vector leads to $\mathbf{1}^\top\mathbf{L}\mathbf{1} = 0$, and $\|\mathbf{C}\mathbf{1}\|_2 > 0$ for any $\mathscr{S} \subseteq \mathscr{V}$ with $K \geq 1$. Further, for any nonzero vectors $\mathbf{u} \neq \mathbf{1}$ that are not equal to all-one vector, we have $\mathbf{u}^\top\mathbf{L}\mathbf{u} > 0$ and $\|\mathbf{C}\mathbf{u}\|_2 \geq 0$. Thus, for any $\mathbf{u} \neq \mathbf{0}$, we have $\mathbf{u}^\top(\mathbf{C}^\top\mathbf{C} + \omega_0\mathbf{L})\mathbf{u} = \|\mathbf{C}\mathbf{u}\|_2^2 + \omega_0\mathbf{u}^\top\mathbf{L}\mathbf{u} > 0$. Hence, term $\mathbf{C}^\top\mathbf{C} + \omega_0\mathbf{L}$ is invertible and positive definite [15].

## 7.2. GREEDY GRAPH SIGNAL SAMPLING

The ground rule of greedy sampling is to work with a submodular cost function in problem (7.2), since it preserves a good suboptimality for the greedy algorithms [17]. In the following, we discuss the submodularity of the Tikhonov estimate based cost function (7.5), then we provide the greedy sampling algorithm based on it.

### 7.2.1. COST FUNCTIONS DISCUSSION BASED ON TIKHONOV SOLUTION

The advantage of Tikhonov regularization solution (7.5) based graph signal sampling is that there is no need of eigendecomposition, like what has been done based on sparse graph Fourier coefficients model [1]. Also, it does better in terms of the tasks like signal reconstruction, interpolation, semi-supervised learning, etc [10, 15]. We discuss the general cost functions based on Tikhonov solution (7.5) in the both deterministic and stochastic cases.

- Assume graph signal $\mathbf{x}$ is deterministic. On the one hand, we can use a scalar function of covariance matrix to design the sampling set with a greedy method, which can be related to the ratio of two submodular functions [18]. The objective function can be the trace or determinant operation, of the covariance matrix of the estimate (7.5) as follows

$$\text{Cov}\{\hat{\mathbf{x}}\} = (\mathbf{C}^\top\mathbf{C} + \omega_0\mathbf{L})^{-1}\mathbf{C}^\top\mathbb{E}(\mathbf{y}_{\mathscr{S}}\mathbf{y}_{\mathscr{S}}^\top)\mathbf{C}(\mathbf{C}^\top\mathbf{C} + \omega_0\mathbf{L})^{-1}. \tag{7.6}$$

  On the other hand, we can also use the MSE of (7.5) as the objective function to design the sampling set, but in general the MSE expression is not well-behaved in terms of convexity, as we have seen in subsection 5.2.6. The work in [15] proposed to maximize the minimal eigenvalue of matrix $\mathbf{C}^\top\mathbf{C} + \omega_0\mathbf{L}$, which is shown to be equivalent to minimize the upper bound of MSE between the true signal and Tikhonov estimate [15, Proposition 1].

  **Proposition 2** ([15])**.** *Maximizing* $\lambda_{min}(\mathbf{C}^\top\mathbf{C} + \omega_0\mathbf{L})$ *minimizes the upper bound of MSE between original signal* $\mathbf{x}$ *and the reconstructed signal* $\hat{\mathbf{x}}$ (7.5)*.*

  *Proof.* See proof in [15, Proposition 1]. □

- Assume graph signal $\mathbf{x}$ is stochastic and follows a multivariate Gaussian distribution

$$p(\mathbf{x}) \propto \exp(-\frac{1}{2}\mathbf{x}^\top\Omega_0\mathbf{x}), \tag{7.7}$$

  where $\Omega_0$ can be graph Laplacian $\mathbf{L}$. Under this setting, (7.5) is a maximum a posterior (MAP) estimator, as well as an MMSE estimator. Then we minimize the MSE, which is the trace of the Tikhonov filter $(\mathbf{C}^\top\mathbf{C} + \omega_0\mathbf{L})^{-1}$ [19]. If we *minimize* this MSE expression as the graph signal sampling objective function

$$\text{tr}(\mathbf{C}^\top\mathbf{C} + \omega_0\mathbf{L})^{-1}, \tag{7.8}$$

  then we will have a supermodular objective function which can be minimized by greedy method with a provable suboptimality [9, 10, 17]. Under this setting, [10] had a thorough study in the context of semi-supervised learning. We will provide the results in the following for convenience.

- Assume graph signal $\mathbf{x}$ is random but not following a Gaussian distribution like above. Instead, we assume it follows a general distribution, with a covariance matrix $\mathbf{L}$. This setting can be understood from the work in [20], which is not detailed in this thesis. Then we will have a Bayesian estimator of form (7.5). Furthermore, it again has the same MSE expression as (7.8) that can be used as the objective function due to its promising properties for greedy algorithms introduced in the following.

In conclusion, the objective function $\mathrm{tr}(\mathbf{C}^\top\mathbf{C}+\omega_0\mathbf{L})^{-1}$ can be thought as a generic choice for all the cases, which is obvious for the random cases. For deterministic cases, maximization of the minimal eigenvalue of $\mathbf{C}^\top\mathbf{C}+\omega_0\mathbf{L}$ is a good choice, or equivalently the minimization of $\lambda_{\max}((\mathbf{C}^\top\mathbf{C}+\omega_0\mathbf{L}))^{-1}$, which is a proxy of (7.8). Thus, we can unify the cost function of the Tikhonov estimate based greedy sampling for graph signal. In the following, we study the *monotonic and submodular* property of (7.8).

### 7.2.2. TIKHONOV ESTIMATE BASED GREEDY GRAPH SIGNAL SAMPLING [10]

By substituting the cost function of $\mathrm{tr}\{(\mathbf{C}^\top\mathbf{C}+\omega_0\mathbf{L})^{-1}\}$ in the general greedy method framework (7.2), we have the following optimization problem

$$\begin{aligned} \arg\min_{\mathbf{c}} \quad & \mathrm{tr}\{(\mathbf{C}^\top\mathbf{C}+\omega_0\mathbf{L})^{-1}\} \\ \text{subject to} \quad & \|\mathbf{c}\|_0 = K, \\ & \mathbf{c}\in\{0,1\}^K. \end{aligned} \tag{7.9}$$

In the following, we aim to show the non-increasing monotonicity and supermodularity of the objective function in problem (7.9), which guarantee the sub-optimality of the greedy sampling.

We first study about the change of several critical term in the objective expression by adding a sample node into the sample subset, i.e., $\Delta_v(\mathscr{S})$.

- **Term $\mathbf{C}^\top\mathbf{C}+\omega_0\mathbf{L}$:**

$$\begin{aligned} \Delta_v(\mathscr{S}) &= (\mathbf{C}^\top_{\mathscr{S}\cup\{v\}}\mathbf{C}_{\mathscr{S}\cup\{v\}}+\omega_0\mathbf{L}) - (\mathbf{C}^\top_{\mathscr{S}}\mathbf{C}_{\mathscr{S}}+\omega_0\mathbf{L}) \\ &= \mathrm{diag}(\mathbf{c}_{\mathscr{S}\cup\{v\}}) - \mathrm{diag}(\mathbf{c}_{\mathscr{S}}) \\ &= \mathrm{diag}(\mathbf{c}_{\{v\}}) \end{aligned} \tag{7.10}$$

- **Term $\mathbf{H}=(\mathbf{C}^\top\mathbf{C}+\omega_0\mathbf{L})^{-1}$:**

$$\begin{aligned} \Delta_v(\mathscr{S}) &= \mathbf{H}_v(\mathscr{S}\cup\{v\}) - \mathbf{H}_v(\mathscr{S}) \\ &= (\mathbf{C}^\top_{\mathscr{S}\cup\{v\}}\mathbf{C}_{\mathscr{S}\cup\{v\}}+\omega_0\mathbf{L})^{-1} - (\mathbf{C}^\top_{\mathscr{S}}\mathbf{C}_{\mathscr{S}}+\omega_0\mathbf{L})^{-1} \\ &= (\mathbf{C}^\top_{\mathscr{S}}\mathbf{C}_{\mathscr{S}}+\mathbf{c}_v^\top\mathbf{c}_v+\omega_0\mathbf{L})^{-1} - (\mathbf{C}^\top_{\mathscr{S}}\mathbf{C}_{\mathscr{S}}+\omega_0\mathbf{L})^{-1}, \end{aligned} \tag{7.11}$$

  where $\mathbf{c}_v$ is a $1\times N$ dimension row vector with one at the $v$-th entry. Adding a node results in a rank-one matrix update $\mathbf{c}_v^\top\mathbf{c}_v$, and by using the matrix inversion

**Algorithm 9** Greedy method of Tikhonov estimate based graph signal sampling[10]

**Input:** size of sampling set $K$; Tikhonov regularization parameter $\omega_0$; graph Laplacian $\mathbf{L}$, #iterations $T$

**Output:** Sampling subset $\mathscr{S}$

1: **Initialization** : $\mathscr{S} = \{\emptyset\}$
2: **while** $|\mathscr{S}| < K$ **do**
3: Find $v^* = \arg\min_{v \in \mathscr{V}/\mathscr{S}} f(\mathscr{S} \cup \{v\})$ where $f(\mathscr{S}) = \mathrm{tr}\{(\mathbf{C}^\top\mathbf{C} + \omega_0\mathbf{L})^{-1}\}$
4: $\mathscr{S} \leftarrow \mathscr{S} \cup \{v^*\}$
5: **end while**

lemma[1] in the first term $\mathbf{H}_v(\mathscr{S} \cup \{v\})$, we have

$$\begin{aligned}\Delta_v(\mathscr{S}) &= \mathbf{H}(\mathscr{S}) - \mathbf{H}(\mathscr{S})\mathbf{c}_v^\top(1 + \mathbf{c}_v\mathbf{H}(\mathscr{S})\mathbf{c}_v^\top)^{-1}\mathbf{c}_v\mathbf{H}(\mathscr{S}) - \mathbf{H}(\mathscr{S}) \\ &= -\frac{\mathbf{H}(\mathscr{S})\mathbf{c}_v^\top\mathbf{c}_v\mathbf{H}(\mathscr{S})}{(1 + \mathbf{c}_v\mathbf{H}(\mathscr{S})\mathbf{c}_v^\top)}.\end{aligned} \tag{7.12}$$

Define the objective function $f(\mathscr{S}) = \mathrm{tr}\{(\mathbf{C}^\top\mathbf{C} + \omega_0\mathbf{L})^{-1}\}$, where the subset $\mathscr{S}$ is indicated by sampling matrix $\mathbf{C}$. We can study the decrease in $f(\mathscr{S})$ due to adding a new node $v$ to $\mathscr{S}$, which we define as $\delta_v(\mathscr{S})$

$$\begin{aligned}\delta_v(\mathscr{S}) &= f(\mathscr{S}) - f(\mathscr{S} \cup \{v\}) \\ &= \mathrm{tr}\{(\mathbf{C}_\mathscr{S}^\top\mathbf{C}_\mathscr{S} + \omega_0\mathbf{L})^{-1}\} - \mathrm{tr}\{(\mathbf{C}_{\mathscr{S}\cup\{v\}}^\top\mathbf{C}_{\mathscr{S}\cup\{v\}} + \omega_0\mathbf{L})^{-1}\} \\ &= \mathrm{tr}\{(\mathbf{C}_\mathscr{S}^\top\mathbf{C}_\mathscr{S} + \omega_0\mathbf{L})^{-1}\} - \mathrm{tr}\{(\mathbf{C}_\mathscr{S}^\top\mathbf{C}_\mathscr{S} + \mathbf{c}_v^\top\mathbf{C}_v + \omega_0\mathbf{L})^{-1}\}.\end{aligned} \tag{7.13}$$

As we have studied in (7.12), adding a node results in a rank-one update, and by using the matrix inversion lemma on the second term above, we have

$$\begin{aligned}\delta_v(\mathscr{S}) &= \mathrm{tr}\{\mathbf{H}(\mathscr{S})\} - \mathrm{tr}\{\mathbf{H}(\mathscr{S}) - \mathbf{H}(\mathscr{S})\mathbf{c}_v^\top(1 + \mathbf{c}_v\mathbf{H}(\mathscr{S})\mathbf{c}_v^\top)^{-1}\mathbf{c}_v\mathbf{H}(\mathscr{S})\} \\ &= \mathrm{tr}\{(\mathbf{H}(\mathscr{S})\mathbf{c}_v^\top\mathbf{c}_v\mathbf{H}(\mathscr{S}))/(1 + \mathbf{c}_v\mathbf{H}(\mathscr{S})\mathbf{c}_v^\top)\} \\ &= \|\mathbf{H}(\mathscr{S})\mathbf{c}_v^\top\|_2^2/(1 + \mathbf{c}_v\mathbf{H}(\mathscr{S})\mathbf{c}_v^\top).\end{aligned} \tag{7.14}$$

From this, we see that adding a new node $v$ into the sect $\mathscr{S}$ leads to a positive decrease in the objective function. Thus, the objective function $f(\mathscr{S}) = \mathrm{tr}\{(\mathbf{C}^\top\mathbf{C} + \omega_0\mathbf{L})^{-1}\}$ is monotonically decreasing (non-increasing).

From the analysis in [10, Theorem 3], we know that the objective function $f(\mathscr{S})$ in (7.9) is supermodular since the graph Laplacian $\mathbf{L}$ is a *Stieltjes* matrix. Therefore, the objective function in problem (7.9) is *monotonically non-increasing and supermodular*, which satisfies the sub-optimal conditions for greedy heuristics [17]. The greedy method based graph signal sampling is shown in Algorithm 9.

However, the objective function in Algorithm 9 involves with a matrix inverse, which has a cubic order of complexity. So, we exploit the properties investigated above to simplify the algorithm and reduce the costs. Adding a new node into the sampling set leads

[1] $(\mathbf{A} + \mathbf{UCV})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{C} + \mathbf{VA}^{-1}\mathbf{U})^{-1}\mathbf{VA}^{-1}$

**Algorithm 10** Greedy method of graph signal sampling with refinement

**Input:** size of sampling set $K$; Tikhonov regularization parameter $\omega_0$; graph Laplacian $\mathbf{L}$, #iterations $T$

**Output:** Sampling subset $\mathcal{S}$

1: **Initialization** : $\mathcal{S} = \{\emptyset\}$
2: **while** $|\mathcal{S}| < K$ **do**
3: Find $v^* = \arg\max_{v \in \mathcal{V}/\mathcal{S}} \delta_v(\mathcal{S})$ where $\delta_v(\mathcal{S}) = \|\mathbf{H}(\mathcal{S})\mathbf{C}_v^\top\|_2^2/(1+\mathbf{c}_v\mathbf{H}(\mathcal{S})\mathbf{c}_v^\top)$
4: $\mathcal{S} \leftarrow \mathcal{S} \cup \{v^*\}$
5: $\mathbf{H}(\mathcal{S}) \leftarrow \mathbf{H}(\mathcal{S}) - \mathbf{H}(\mathcal{S})\mathbf{c}_v^\top(1+\mathbf{c}_v\mathbf{H}(\mathcal{S})\mathbf{c}_v^\top)^{-1}\mathbf{c}_v\mathbf{H}(\mathcal{S})$
6: **end while**

to a rank-one update and the decrease in the objective function is $\delta_v(\mathcal{S})$ in (7.14). To minimize the objective function at each loop is equivalent to maximize the decrease in the objective function by adding a new node [9]. Therefore, we propose the following Algorithm 10. The similar cost reduction method is also used in [9].

## 7.3. NODE-ADAPTIVE ESTIMATE BASED GREEDY GRAPH SIGNAL SAMPLING

The application of NA regularization in graph signal sampling is straightforward. Consider the node-adaptive graph signal regularization for the sampling signal model

$$\hat{\mathbf{x}}_{NA} = \arg\min_{\mathbf{x}} \|\mathbf{y}_{\mathcal{S}} - \mathbf{C}\mathbf{x}\|^2 + \mathbf{x}^\top\tilde{\mathbf{L}}\mathbf{x}, \tag{7.15}$$

where $\tilde{\mathbf{L}} = \mathrm{diag}(\boldsymbol{\omega})\mathbf{L}\mathrm{diag}(\boldsymbol{\omega})$ is positive semi-definite and has rank $n-1$, and it is also a valid *Stieltjes matrix*. The solution of (7.15) is

$$\hat{\mathbf{x}}_{NA} = (\mathbf{C}^\top\mathbf{C} + \tilde{\mathbf{L}})^{-1}\mathbf{y}_{\mathcal{S}}, \tag{7.16}$$

and we can define the NA filter $\mathbf{H}_{NA}(\mathcal{S}) = (\mathbf{C}^\top\mathbf{C} + \tilde{\mathbf{L}})^{-1} = (\mathbf{C}^\top\mathbf{C} + \mathrm{diag}(\boldsymbol{\omega})\mathbf{L}\mathrm{diag}(\boldsymbol{\omega}))^{-1}$.

In the subset sampling problem based on the NA regularization, the cost function in all deterministic and random model cases can be unified to be

$$\mathrm{tr}\{(\mathbf{C}^\top\mathbf{C} + \tilde{\mathbf{L}})^{-1}\}, \tag{7.17}$$

which is again *monotonically non-decreasing and supermodular*. By replacing the Tikhonov estimate (7.5) by NA estimate (7.16), we simply realize the NA estimate based greedy graph signal sampling, shown in Algorithms 11 and 12.

## 7.4. NUMERICAL EXPERIMENTS

In this section, we present the experimental results on the graph signal sampling based on Tikhonov regularization and node-adaptive regularization, respectively. We compare the performance random sampling[2] and the greedy method we proposed, respectively.

[2] The sampling set is randomly generated.

**Algorithm 11** Greedy method of NA estimate based graph signal sampling

**Input:** size of sampling set $K$; NA regularization parameter $\boldsymbol{\omega}$; graph Laplacian $\mathbf{L}$, #iterations $T$

**Output:** Sampling subset $\mathcal{S}$

1: **Initialization** : $\mathcal{S} = \{\emptyset\}$
2: **while** $|\mathcal{S}| < K$ **do**
3: Find $v^* = \arg\min_{v \in \mathcal{V}/\mathcal{S}} f(\mathcal{S} \cup \{c\})$ where $f(\mathcal{S}) = \mathrm{tr}\{(\mathbf{C}^\top \mathbf{C} + \tilde{\mathbf{L}})^{-1}\}$, and $\tilde{\mathbf{L}} = \mathrm{diag}(\boldsymbol{\omega})\mathbf{L}\mathrm{diag}(\boldsymbol{\omega})$
4: $\mathcal{S} \leftarrow \mathcal{S} \cup \{v^*\}$
5: **end while**

**Algorithm 12** NA estimate based Greedy method of graph signal sampling with refinement

**Input:** size of sampling set $K$; NA regularization parameter $\boldsymbol{\omega}$; graph Laplacian $\mathbf{L}$, #iterations $T$

**Output:** Sampling subset $\mathcal{S}$

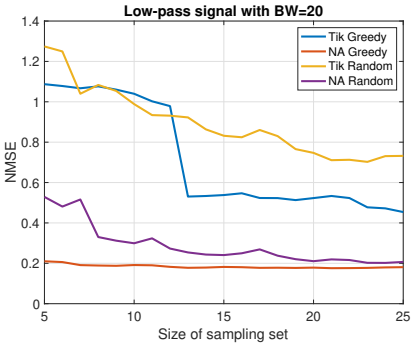
1: **Initialization** : $\mathcal{S} = \{\emptyset\}$
2: **while** $|\mathcal{S}| < K$ **do**
3: Find $v^* = \arg\max_{v \in \mathcal{V}/\mathcal{S}} \delta_v(\mathcal{S})$ where $\delta_v(\mathcal{S}) = \|\mathbf{H}_{NA}(\mathcal{S})\mathbf{c}_v^\top\|_2^2 / (1 + \mathbf{c}_v \mathbf{H}_{NA}(\mathcal{S})\mathbf{c}_v^\top)$
4: $\mathcal{S} \leftarrow \mathcal{S} \cup \{v^*\}$
5: $\mathbf{H}_{NA}(\mathcal{S}) \leftarrow \mathbf{H}_{NA}(\mathcal{S}) - \mathbf{H}_{NA}(\mathcal{S})\mathbf{c}_v^\top (1 + \mathbf{c}_v \mathbf{H}_{NA}(\mathcal{S})\mathbf{c}_v^\top)^{-1} \mathbf{c}_v \mathbf{H}_{NA}(\mathcal{S})$
6: **end while**

Regrading the choice of NA regularization weight vector $\boldsymbol{\omega}$, we consider the enhanced Prony's method from subsection 6.4.1.

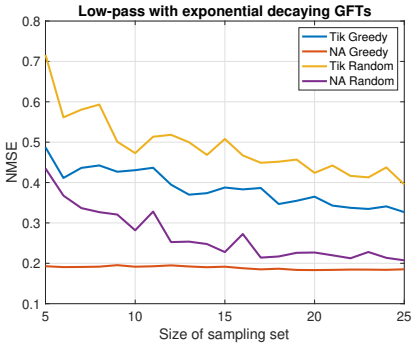
In the experiment setting, we consider a synthetic Erdős–Rényi graph with 50 nodes and 0.5 edge connection probability. We artificially add the noise to generate a 0dB SNR. we consider the following different graph signals

- Low-pass smooth signals with its GFT is constantly one in the first 20 graph frequencies, i.e., $\tilde{x}_i = 1$, for $i = 1, \ldots, 20$, and the rest are zeros.
- Low-pass smooth signals with decaying GFT in the first 20 graph frequencies, i.e., $\tilde{x}_i = \frac{1}{1+\lambda_i}$, for $i = 1, \ldots, 20$, and the rest are zeros.
- High-pass signals with constant GFT in the last 20 graph frequencies, i.e., $\tilde{x}_i = 1$, for $i = 31, \ldots, 50$, and the rest are zeros.
- Random graph signals follow a Gaussian distribution $\mathcal{N}(0, \mathbf{L}^\dagger)$.

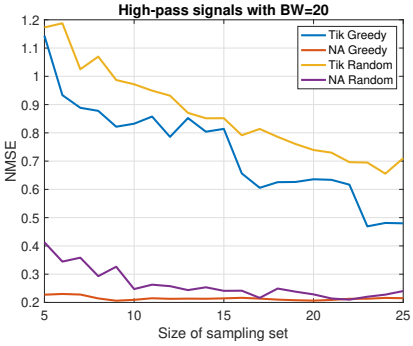
The sampling set size is chosen to be in the set $\mathcal{M} = \{5, 6, 7, \ldots, 43, 44, 45\}$. We generate 100 noisy realizations for each type of signals. For random sampling set selection, the sampling set is generated randomly over 50 realizations to average the performance. For Tikhonov and NA estimate based greedy methods, the sampling sets are determined greedily based on Algorithm 9 and 11, respectively. With the sampling sets built by random sampling and greedy sampling based on Tikhonov estimate and NA estimate, we average the reconstruction NMSE results over 10 rounds.



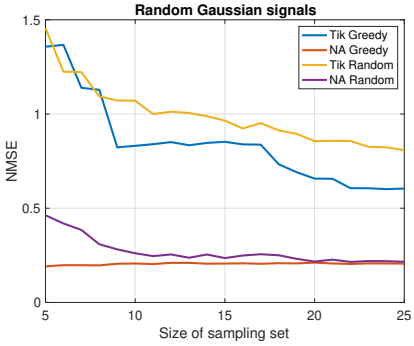

(a) Low-pass signals with constant GFT in the bandwidth of 20.




(b) Low-pass signal with exponential decaying GFT.




(c) High-pass signals with constant GFT in the bandwidth of 20.




(d) Random Gaussian signals following $\mathcal{N}(\mathbf{0}, \mathbf{L}^{\dagger})$.

Figure 7.1: The sampling performance comparison between Tikhonov estimate and NA estimate based greedy graph signal sampling.

From the experiment results in Figure 7.1, we have the following observations:

- The performance of graph signal reconstruction from partial observations of the noisy signals based on our NA regularization is much better than the typical Tikhonov regularization, with either random scheme or greedy scheme. This is not surprising, since in chapter 6 we have already seen the superiority of the NA regularization, which captures the local signal information.
- Sampling the subset with greedy approach in general can perform better than the random sampling. The reason is because at each greedy sampling step, a node is sampled to minimize the cost function related to MSE, while random sampling does not optimize.
- In Low-pass signal and Random the greedy method of Tikhonov regularization behaves not well. This is because the Tikhonov filter matrix is close to singular or

badly scaled numerically.

- The interpolation performance has shown the similar patterns for both non-Bayesian and Bayesian settings though we use the same cost functions (7.9) and (7.17), respectively. This supports our unified greedy sampling framework for both deterministic and random signals (cf. subsection 7.2.1).

## 7.5. CONCLUSION AND DISCUSSIONS

In this chapter, we first develop the Tikhonov estimate based greedy algorithms. It is partly investigated in [10], and we generalize it to both the Bayesian and non-Bayesian setting. Different from the work in [9] where a bandlimited graph signal model is considered, we consider a Tikhonov estimate based model which does not require eigendecomposition of the graph Laplacian with a high computational cost. Then, we shift to NA estimate based sampling, simply by replacing Tikhonov estimate with NA estimate. With experiments on synthetic signals under both Bayesian and non-Bayesian settings, we showed the significant improvement of the performance by NA estimate based greedy graph signal sampling, compared to the commonly used Tikhonov estimate based.

## REFERENCES

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

[2] S. Joshi and S. Boyd, *Sensor selection via convex optimization,* IEEE Transactions on Signal Processing **57**, 451 (2008).

[3] A. J. Miller and N.-K. Nguyen, *Algorithm as 295: A fedorov exchange algorithm for d-optimal design,* Journal of the royal statistical society. series c (applied statistics) **43**, 669 (1994).

[4] S. P. Chepuri and G. Leus, *Sparsity-promoting sensor selection for non-linear measurement models,* IEEE Transactions on Signal Processing **63**, 684 (2014).

[5] M. Fickus, D. G. Mixon, and M. J. Poteet, *Frame completions for optimally robust reconstruction,* in *Wavelets and Sparsity XIV*, Vol. 8138 (International Society for Optics and Photonics, 2011) p. 81380Q.

[6] G. Ortiz-Jiménez, M. Coutino, S. P. Chepuri, and G. Leus, *Sparse sampling for inverse problems with tensors,* IEEE Transactions on Signal Processing **67**, 3272 (2019).

[7] M. Bouhtou, S. Gaubert, and G. Sagnol, *Submodularity and randomized rounding techniques for optimal experimental design,* Electronic Notes in Discrete Mathematics **36**, 679 (2010).

[8] Z. Allen-Zhu, Y. Li, A. Singh, and Y. Wang, *Near-optimal discrete optimization for experimental design: A regret minimization approach,* Mathematical Programming , 1 (2020).

[9] L. F. Chamon and A. Ribeiro, *Greedy sampling of graph signals,* IEEE Transactions on Signal Processing **66**, 34 (2017).

[10] P.-Y. Chen and D. Wei, *On the supermodularity of active graph-based semi-supervised learning with stieltjes matrix regularization,* in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, 2018) pp. 2801–2805.

[11] A. Hashemi, R. Shafipour, H. Vikalo, and G. Mateos, *Accelerated sampling of bandlimited graph signals,* arXiv preprint arXiv:1807.07222 (2018).

[12] C. Rusu and J. Thompson, *Node sampling by partitioning on graphs via convex optimization,* in *2017 Sensor Signal Processing for Defence Conference (SSPD)* (IEEE, 2017) pp. 1–5.

[13] F. Wang, G. Cheung, and Y. Wang, *Low-complexity graph sampling with noise and signal reconstruction via neumann series,* IEEE Transactions on Signal Processing **67**, 5511 (2019).

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms* (MIT press, 2009).

[15] Y. Bai, F. Wang, G. Cheung, Y. Nakatsukasa, and W. Gao, *Fast graph sampling set selection using gershgorin disc alignment,* IEEE Transactions on Signal Processing **68**, 2419 (2020).

[16] Y. Tanaka, Y. C. Eldar, A. Ortega, and G. Cheung, *Sampling on graphs: From theory to applications,* arXiv preprint arXiv:2003.03957 (2020).

[17] B. Winer, *Statistical Principles in Experimental Design: 2d Ed* (McGraw-Hill, 1971).

[18] M. Coutino, S. P. Chepuri, and G. Leus, *Subset selection for kernel-based signal reconstruction,* in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, 2018) pp. 4014–4018.

[19] S. M. Kay, *Fundamentals of statistical signal processing* (Prentice Hall PTR, 1993).

[20] G. Mateos, S. Segarra, A. G. Marques, and A. Ribeiro, *Connecting the dots: Identifying network structure via graph signal processing,* IEEE Signal Processing Magazine **36**, 16 (2019).

# 8

# CONCLUSION & FUTURE WORK

In conclusion, the second part of our work focuses on the node-adaptive graph signal regularization, which relies on a more general prior on the graph signals. This can be learned by designing the regularization parameters $\boldsymbol{\omega}$ and more adaptive compared to the global smoothness prior, i.e., Tikhonov regularization. With this enhanced degree of freedom, we have shown that indeed NA regularization does better than Tikhonov regularization in signal reconstruction and interpolation over graphs. We also studied the bias-variance trade-off introduced by the regularization parameters. Different methods are proposed to design the NA regularization parameters to minimize the MSE, first based on Prony's method, then some convex relaxation techniques used. To solve the true signal dependency problem, we proposed the *min-max* formulation by only using the signal variation bounds. Afterwards, we applied the NA graph signal regularization in the problem of greedy graph signal sampling, showing its superior performance on different types of graph signals. Extensive experiments on synthetic and real-world data have shown the ability of node-adaptive graph signal regularization and the validity of different regularization parameters design methods.

## FUTURE RESEARCH QUESTIONS AND STATEMENTS

1. *Under stochastic signal model where the graph signals follow certain prior random distribution, e.g.,* $\mathcal{N}(\mathbf{0}, \mathbf{L}^{\dagger})$*, the experiment results indicate that NA regularization performs better than Tikhonov regularization in general which shall be the best in terms of MSE, why?*

   Notice that in the previous experiments in Section C.1, we also compared the performance of our approach on the random signals which do not agree with our underlying deterministic model where the true signal has no prior statistical information.

   Suppose the true graph signal follows a Gaussian distribution with zero mean and $\mathbf{L}^{\dagger}$ as covariance matrix and the noise follows a Gaussian distribution $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$. We can find that Tikhonov regularization solution with penalty weight $\omega_0 = \sigma^2$ is

an MAP (maximum a posterior probability) estimator, an MMSE (Wiener) estimator as well as, since the probability density functions of the signal and the noise are both symmetric. However, the experiments indicate that our NA regularization can still do better than Tikhonov regularization when the parameter is chosen as noise variance.

2. As we have mentioned in Appendix C, in the constant transform equation (cf.(C.2)) we set the constant as 1, which is not always right. When it is too small, the regularization term will be too trivial to influence the fitting term. In our experiments, we did not tune this scalar in the synthetic data, but for the real data ones, we tune it till obtaining a good performance. In practice, this is doable, but it is necessary to have a proper theoretical study regarding to this scalar value level, which shall be related to the data value level and controls the trade-off between the data fitting the regularization term on a high level.

3. In the work of graph signal denoising based on graph neural networks [1], the authors proposed to learn diverse graph signal priors through neural networks, specifically unrolling the iterative algorithms into neural networks. This so-called graph unrolling networks leverage the learning ability of neural networks to adapt the diverse priors of graph signals so to obtain a better denoising performance. Our proposed method does not rely on the neural networks to learn and the constant transform based NA parameters design method does not rely on complex optimizations either.

4. In the numerical experiments of chapter 7, we design the NA weights based on enhanced Prony's method, which requires the true signal information. In future work, we need to carry out the NA weights design method not relying on true signals.

# REFERENCES

[1] S. Chen, Y. C. Eldar, and L. Zhao, *Graph unrolling networks: Interpretable neural networks for graph signal denoising,* arXiv preprint arXiv:2006.01301 (2020).

# A

# APPENDIX OF PART I

## A.1. PROOF OF THEOREM 1

*Proof.* Without loss of generality, an error in the entries of $\mathbf{D}$ indicates the following two cases:

- $d_{ij} > \epsilon, (i,j) \subset \mathscr{C} - \mathscr{T}$ and $c_{ij} < \epsilon$. In contrary, $d(i,j)$ is supposed to be 0. The probability of this event is

$$\begin{aligned} &\Pr(d_{ij} > \epsilon, \text{ where } (i,j) \subset \mathscr{C} - \mathscr{T}) \\ =&\Pr(x_{ij} > \frac{\epsilon^2 s}{\|\mathbf{W}\|_1}) \\ =&\Pr(x_{ij} - \mathbb{E}(x_{ij}) > \frac{\epsilon^2 s}{\|\mathbf{W}\|_1} - \mathbb{E}(x_{ij})) \\ =&\Pr(x_{ij} > \mathbb{E}(x_{ij}) + \delta_1), \end{aligned} \tag{A.1}$$

where $\delta_1 = \frac{\epsilon^2 s}{\|\mathbf{W}\|_1} - \mathbb{E}(x_{ij})$. Note that $\mathbb{E}(x_{ij}) = \frac{c_{ij}^2 s}{\|\mathbf{W}\|_1}$, so we have $\delta_1 = (\epsilon^2 - c_{ij}^2)\frac{s}{\|\mathbf{W}\|_1}$. Following the proof as in [1, Lemma 3], by applying the standard multiplicative Chernoff bound of [2, Theorem 1.1], we have

$$\Pr(x_{ij} > \mathbb{E}(x_{ij}) + \delta_1) < \exp(-2\delta_1^2/(sK^2)). \tag{A.2}$$

Consider a worst case scenario, the minimum of $\delta_1$ is $\min(\delta_1) = \epsilon^2 \frac{s}{\|\mathbf{W}\|_1}$, when $c_{ij} = 0$. Thus, we have

$$\Pr(x_{ij} > \mathbb{E}(x_{ij}) + \delta_1) < \exp(-2\delta_1^2/(sK^2)) < \exp(-2\epsilon^4 s/(K\|\mathbf{W}\|_1)^2). \tag{A.3}$$

The total number of errors in this case is limited by

$$\Pr(x_{ij} > \mathbb{E}(x_{ij}) + \delta_1) \cdot (M^2 - T) < \exp(-2\epsilon^4 s/(K\|\mathbf{W}\|_1)^2) \cdot (M^2 - T). \tag{A.4}$$

- $d_{ij} = 0$, where $(i, j) \subset \mathscr{T}$ and $c_{ij} > \epsilon$. In contrary, $d_{ij}$ is supposed to be $c_{ij}$, within the top-$T$s.
  The probability of this event is

$$\begin{aligned} &\Pr(d_{ij} = 0\,(i.e., d_{ij} < \epsilon), (i, j) \subset \mathscr{T}) \\ =&\Pr(x_{ij} < \frac{\epsilon^2 s}{\|\mathbf{W}\|_1}) \\ =&\Pr(x_{ij} - \mathbb{E}(x_{ij}) < \frac{\epsilon^2 s}{\|\mathbf{W}\|_1} - \mathbb{E}(x_{ij})) \\ =&\Pr(x_{ij} > \mathbb{E}(x_{ij}) - \delta_2), \end{aligned} \tag{A.5}$$

  where $\delta_2 = \mathbb{E}(x_{ij}) - \frac{\epsilon^2 s}{\|\mathbf{W}\|_1} = (c_{ij}^2 - \epsilon^2)\frac{s}{\|\mathbf{W}\|_1}$.
  By following the similar procedure, we have

$$\Pr(x_{ij} > \mathbb{E}(x_{ij}) - \delta_2) < \exp(-2\delta_2^2/(sK^2)). \tag{A.6}$$

  If we consider a worst case scenario, we have $\min(\delta_2) = (\min(c_{ij}^2) - \epsilon^2)\frac{s}{\|\mathbf{W}\|_1}$, for $(i, j) \subset \mathscr{T}$. Thus, we have

$$\Pr(x_{ij} > \mathbb{E}(x_{ij}) - \delta_2) < \exp(-2\delta_2^2/(sK^2)) < \exp(-2(\min(c_{ij}^2) - \epsilon^2)^2 s/(K\|\mathbf{W}\|_1)^2). \tag{A.7}$$

  The total number of errors in this case is limited by

$$\Pr(x_{ij} > \mathbb{E}(x_{ij}) - \delta_2) \cdot T < \exp(-2(\min(c_{ij}^2) - \epsilon^2)^2 s/(K\|\mathbf{W}\|_1)^2) \cdot T. \tag{A.8}$$

Summing the two cases above, we complete the proof. □

## REFERENCES

[1] G. Ballard, T. G. Kolda, A. Pinar, and C. Seshadhri, *Diamond sampling for approximate maximum all-pairs dot-product (mad) search,* 2015 IEEE International Conference on Data Mining (2015), 10.1109/icdm.2015.46.

[2] D. P. Dubhashi and A. Panconesi, *Concentration of measure for the analysis of randomized algorithms* (Cambridge University Press, 2009).

# B

# APPENDICES OF PART II

## B.1. IMPORTANT LEMMAS AND THEOREMS

**Lemma 6** ([1]). *Schur complement lemma : Given any symmetric matrix,* $\mathbf{M} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{C} \end{pmatrix}$, *the following conditions are equivalent:*

- $\mathbf{M} \succeq 0$ *(*$\mathbf{M}$ *is positive semi-definite);*
- $\mathbf{A} \succeq 0, \quad (\mathbf{I} - \mathbf{A}\mathbf{A}^\dagger)\mathbf{B} = 0, \quad \mathbf{C} - \mathbf{B}^\top \mathbf{A}^\dagger \mathbf{B} \succeq 0$
- $\mathbf{C} \succeq 0, \quad (\mathbf{I} - \mathbf{C}\mathbf{C}^\dagger)\mathbf{B} = 0, \quad \mathbf{A} - \mathbf{B}^\top \mathbf{C}^\dagger \mathbf{B} \succeq 0$

**Lemma 7.** *Let* $\mathbf{A}, \mathbf{B}, \mathbf{C}$ *be* $n \times n$ *symmetric matrices, then* $tr\left((\mathbf{A}^2 - \mathbf{B}^2)\mathbf{C}\right) = tr((\mathbf{A}-\mathbf{B})(\mathbf{A}+\mathbf{B})\mathbf{C}) - tr((\mathbf{A}\mathbf{B}-\mathbf{B}\mathbf{A})\mathbf{C}) = tr((\mathbf{A}-\mathbf{B})(\mathbf{A}+\mathbf{B})\mathbf{C})$, *because* $tr(\mathbf{A}\mathbf{B}\mathbf{C}) = tr\left((\mathbf{A}\mathbf{B}\mathbf{C})^\top\right) = tr(\mathbf{C}\mathbf{B}\mathbf{A}) = tr(\mathbf{A}\mathbf{C}\mathbf{B})$[1].

**Lemma 8.** *Let* $\mathbf{A}$ *be an* $n \times n$ *positive semi-definite matrix, and* $\mathbf{B}$ *an* $n \times n$ *negative semi-definite matrix, then* $tr(\mathbf{A}\mathbf{B}) \leq 0$.

*Proof.* Consider an eigenvalue-eigenvector equation of matrix $\mathbf{A}\mathbf{B}$ as follows

$$\mathbf{A}\mathbf{B}\mathbf{x} = \lambda \mathbf{x}, \tag{B.1}$$

where eigenvalue $\lambda$ is a scalar, and eigenvector $\mathbf{x}$ is a vector of length $n$. If we multiple (B.1) by $\mathbf{x}^\top \mathbf{B}$ on both sides, then

$$\mathbf{x}^\top \mathbf{B}\mathbf{A}\mathbf{B}\mathbf{x} = \lambda \mathbf{x}^\top \mathbf{B}\mathbf{x}. \tag{B.2}$$

The eigenvalue of $\mathbf{A}\mathbf{B}$ can be represented as

$$\lambda = \frac{\mathbf{x}^\top \mathbf{B}\mathbf{A}\mathbf{B}\mathbf{x}}{\mathbf{x}^\top \mathbf{B}\mathbf{x}}. \tag{B.3}$$

[1] Due to the cyclic property of trace and $\mathrm{tr}(\mathbf{A}) = \mathrm{tr}(\mathbf{A}^\top)$

From [1, Theorem 7.2.7], matrix **BAB** is positive semi-definite independent of matrix **B**. Thus, the numerator $\mathbf{x}^\top\mathbf{BABx}$ is nonnegative. Since matrix **B** is negative semi-definite, so we have the denominator $\mathbf{x}^\top\mathbf{Bx}$ nonpositive. This results in a general nonpositive eigenvalue $\lambda$. Thus, the trace of matrix **AB** is nonpositive. □

**Lemma 9** ([1] )**.** *Let $n\times n$ matrices $\mathbf{A},\mathbf{B}$ be positive semi-definite, then $\mathbf{A}\odot\mathbf{B}$ is positive semi-definite and $\lambda_{max}(\mathbf{A}\odot\mathbf{B})\le\lambda_{max}(\mathbf{A})max\{b_{ii}\}$ where $b_{ii}$ is the $i$-th diagonal element of $\mathbf{B}$. Since the inequality is given in the exercise part of [1], so we give the proof below.*

*Proof.* Let $\lambda_n$ be the maximal eigenvalue of **A**, then $\mathbf{A}-\lambda_n\mathbf{I}\preceq\mathbf{0}$. From the preceding lemma, we have $(\mathbf{A}-\lambda_n\mathbf{I})\odot\mathbf{B}\preceq\mathbf{0}$. Let $\mathbf{x}\in\mathbb{C}^n$ be a nonzero vector, then

$$\mathbf{x}^\top((\mathbf{A}-\lambda_n\mathbf{I})\odot\mathbf{B})\mathbf{x}=\mathbf{x}^\top(\mathbf{A}\odot\mathbf{B})\mathbf{x}-\lambda_n\mathbf{x}^\top(\mathbf{I}\odot\mathbf{B})\mathbf{x}\le 0,$$

which leads to

$$\mathbf{x}^\top(\mathbf{A}\odot\mathbf{B})\mathbf{x}\le\lambda_n\mathbf{x}^\top(\mathbf{I}\odot\mathbf{B})\mathbf{x}=\lambda_n\sum_{i=1}^{n}b_{ii}|x_i|^2\le\lambda_n\max\{b_{ii}\}\|\mathbf{x}\|^2.$$

If **x** is the eigenvector, it completes the proof. □

**Theorem 3** (Weyl's Inequality[1])**.** *Let $\mathbf{A},\mathbf{B}$ be $n\times n$ Hermitian matrices and let the respective eigenvalues of $\mathbf{A},\mathbf{B}$ and $\mathbf{A}+\mathbf{B}$ be $\{\lambda_i(\mathbf{A})\}_{i=1}^n$, $\{\lambda_i(\mathbf{B})\}_{i=1}^n$ and $\{\lambda_i(\mathbf{A}+\mathbf{B})\}_{i=1}^n$, each of which is algebraically ordered as $\lambda_{min}=\lambda_1\le\lambda_2\le\cdots\le\lambda_{n-1}\le\lambda_n=\lambda_{max}$. Then*

$$\lambda_i(\mathbf{A}+\mathbf{B})\le\lambda_{i+j}(\mathbf{A})+\lambda_{n-j}(\mathbf{B}),\quad j=0,1,\ldots,n-i$$

*for each $i=1,\ldots,n$, with equality for some pair $i,j$ if and only if there is a nonzero vecto $\mathbf{x}$ such that $\mathbf{Ax}=\lambda_{i+j}(\mathbf{A})\mathbf{x}$, $\mathbf{Bx}=\lambda_{n-j}(\mathbf{B})\mathbf{x}$ and $(\mathbf{A}+\mathbf{B})\mathbf{x}=\lambda_i(\mathbf{A}+\mathbf{B})\mathbf{x}$. Also,*

$$\lambda_{i-j+1}(\mathbf{A})+\lambda_j(\mathbf{B})\le\lambda_i(\mathbf{A}+\mathbf{B}),\quad j=1,\ldots,i$$

*for each $i=1,\ldots,n$, with equality for some pair $i,j$ if and only if there is a nonzero vector $\mathbf{x}$ such that $\mathbf{Ax}=\lambda_{i-j+1}(\mathbf{A})\mathbf{x}$, $\mathbf{Bx}=\lambda_j(\mathbf{B})\mathbf{x}$ and $(\mathbf{A}+\mathbf{B})\mathbf{x}=\lambda_i(\mathbf{A}+\mathbf{B})\mathbf{x}$. If $\mathbf{A}$ and $\mathbf{B}$ have no common eigenvector, then every inequality above is a strict one.*

## B.2. PROOF OF LEMMA 5

To show $\mathrm{var}(\boldsymbol{\omega})\le\mathrm{var}(\omega_0)$, given $\omega_i^2\ge\omega_0$, for $i=1,2,\ldots,n$, it suffices to show

$$\mathrm{var}(\boldsymbol{\omega})-\mathrm{var}(\omega_0)=\mathrm{tr}(\mathbf{H}^2(\boldsymbol{\omega})\boldsymbol{\Sigma})-\mathrm{tr}(\mathbf{H}^2(\omega_0)\boldsymbol{\Sigma})=\mathrm{tr}\{[(\mathbf{H}(\boldsymbol{\omega})-\mathbf{H}(\omega_0))(\mathbf{H}(\boldsymbol{\omega})+\mathbf{H}(\omega_0))]\cdot\boldsymbol{\Sigma}\}\le 0$$

where the second equality comes from Lemma 7. Since matrices $\boldsymbol{\Sigma}$, $\mathbf{H}(\boldsymbol{\omega})$ and $\mathbf{H}(\omega_0)$ are positive semi-definite by definition, from Lemma 8 it suffices to show $\mathbf{H}(\boldsymbol{\omega})-\mathbf{H}(\omega_0)\preceq\mathbf{0}$ with the given condition $\omega_i^2>\omega_0$ for $i=1,\ldots,N$.

With this condition, we have $\boldsymbol{\omega}\boldsymbol{\omega}^\top\succeq\omega_0\mathbf{1}\mathbf{1}^\top$. Since $\mathrm{diag}(\boldsymbol{\omega})\mathbf{L}\mathrm{diag}(\boldsymbol{\omega})=\boldsymbol{\omega}\boldsymbol{\omega}^\top\odot\mathbf{L}$ and $\omega_0\mathbf{L}=\omega_0\mathbf{1}\mathbf{1}^\top\odot\mathbf{L}$, we further have

$$(\mathbf{I}+\omega_0\mathbf{1}\mathbf{1}^\top\odot\mathbf{L})-(\mathbf{I}+\boldsymbol{\omega}\boldsymbol{\omega}^\top\odot\mathbf{L})=(\omega_0\mathbf{1}\mathbf{1}^\top-\boldsymbol{\omega}\boldsymbol{\omega}^\top)\odot\mathbf{L}\preceq\mathbf{0}$$

where the equality is because Hadamard product is distributive over addition. Then, we left multiply both sides by $(\mathbf{I}+\omega_0\mathbf{1}\mathbf{1}^\top \odot \mathbf{L})^{-1}$ and right multiply both sides by $(\mathbf{I}+\boldsymbol{\omega}\boldsymbol{\omega}^\top \odot \mathbf{L})^{-1}$. This does not change the sign because they are both PSD. Hence, we have

$$(\mathbf{I}+\boldsymbol{\omega}\boldsymbol{\omega}^\top \odot \mathbf{L})^{-1} - (\mathbf{I}+\omega_0\mathbf{1}\mathbf{1}^\top \odot \mathbf{L})^{-1} \preceq \mathbf{0}$$
$$\mathbf{H}(\boldsymbol{\omega}) - \mathbf{H}(\omega_0) \preceq \mathbf{0}$$

which completes the proof.

## B.3. PROOF OF THEOREM 2

Proving $\text{mse}(\boldsymbol{\omega}) \leq \text{mse}(\omega_0)$ with the given conditions, is equivalent to showing $\Delta = \text{mse}(\boldsymbol{\omega}) - \text{mse}(\omega_0) \leq 0$. We expand the latter as

$$\Delta = \text{tr}\left\{(\mathbf{I}-\mathbf{H}(\boldsymbol{\omega}))^2 \mathbf{x}^* \mathbf{x}^{*T}\right\} + \text{tr}(\mathbf{H}^2(\boldsymbol{\omega})\boldsymbol{\Sigma}) - \text{tr}\left\{(\mathbf{I}-\mathbf{H}(\omega_0))^2 \mathbf{x}^* \mathbf{x}^{*T}\right\} + \text{tr}(\mathbf{H}^2(\omega_0)\boldsymbol{\Sigma}).$$

Since $\mathbf{x}^*\mathbf{x}^{*T} = \mathbf{P}\boldsymbol{\Sigma}$ holds by definition, by working out the above equation, we have

$$\Delta = \text{tr}\{\boldsymbol{\Sigma}\,(\mathbf{H}(\omega_0) - \mathbf{H}(\boldsymbol{\omega})) \cdot [2\mathbf{P} - (\mathbf{H}(\boldsymbol{\omega}) + \mathbf{H}(\omega_0))\,(\mathbf{I}+\mathbf{P})]\}.$$

Due to the covariance matrix $\boldsymbol{\Sigma} \succeq \mathbf{0}$ and from the first condition (6.9a) we have also that the filter difference is positive semi-definite $\mathbf{H}(\omega_0) - \mathbf{H}(\boldsymbol{\omega}) \succeq \mathbf{0}$. Thus, it suffices to show that

$$2\mathbf{P} - (\mathbf{H}(\boldsymbol{\omega}) + \mathbf{H}(\omega_0))\,(\mathbf{I}+\mathbf{P}) \preceq \mathbf{0}.$$

Further, since $\mathbf{I}+\mathbf{P} \succ \mathbf{0}$ and invertible, we can focus on proving the smallest eigenvalue is greater or equal to zero, i.e.,

$$\lambda_{\min}\left\{\mathbf{H}(\boldsymbol{\omega}) + \mathbf{H}(\omega_0) - 2\mathbf{P}\,(\mathbf{I}+\mathbf{P})^{-1}\right\} \geq 0.$$

Let us then define the matrix $\boldsymbol{\Gamma} \triangleq \mathbf{P}\,(\mathbf{I}+\mathbf{P})^{-1}$ and scalar $\gamma \in (0,1)$ as its only nonzero eigenvalue for simplifying the notations. From Theorem 3, we have

$$\lambda_{\min}\{\mathbf{H}(\boldsymbol{\omega}) + \mathbf{H}(\omega_0) - 2\boldsymbol{\Gamma}\} \geq \lambda_{\min}\{\mathbf{H}(\boldsymbol{\omega})\} + \lambda_{\min}\{\mathbf{H}(\omega_0)\} + \lambda_{\min}\{-2\boldsymbol{\Gamma}\}.$$

So, a sufficient condition for $\text{mse}(\boldsymbol{\omega}) \leq \text{mse}(\omega_0)$ is

$$\lambda_{\min}\{\mathbf{H}(\boldsymbol{\omega})\} + \lambda_{\min}\{\mathbf{H}(\omega_0)\} - \lambda_{\max}\{2\boldsymbol{\Gamma}\} \geq 0$$

where $\lambda_{\max}\{2\boldsymbol{\Gamma}\} = 2\gamma \in (0,2)$, and $\lambda_{\min}\{\mathbf{H}(\boldsymbol{\omega})\}$, $\lambda_{\min}\{\mathbf{H}(\omega_0)\}$ can be found as follows, respectively. From the following eigen-decomposition

$$\mathbf{H}(\omega_0) \triangleq (\mathbf{I}+\omega_0\mathbf{L})^{-1} = \sum_{i=1}^{N} \frac{1}{1+\omega_0\lambda_i}\mathbf{u}_i\mathbf{u}_i^\top$$

where $\lambda_i$ is the eigenvalue of Laplacian matrix $\mathbf{L}$, and $\mathbf{u}_i$ is the eigenvector of $\mathbf{L}$, we have

$$\lambda_{\min}\{\mathbf{H}(\omega_0)\} = \lambda_{\min}\left\{(\mathbf{I}+\omega_0\mathbf{L})^{-1}\right\} = \frac{1}{1+\omega_0\lambda_{\max}(\mathbf{L})}.$$

From Lemma 9, we have

$$\lambda_{\max}\left\{\boldsymbol{\omega}\boldsymbol{\omega}^{\top}\odot\mathbf{L}\right\}\leq\lambda_{\max}(\mathbf{L})\max\left\{\omega_i^2\right\},$$

which results in

$$\lambda_{\max}\left\{\mathbf{I}+\boldsymbol{\omega}\boldsymbol{\omega}^{\top}\odot\mathbf{L}\right\}\leq 1+\lambda_{\max}(\mathbf{L})\max\left\{\omega_i^2\right\},$$

which is further followed by the lower bound of $\lambda_{\min}\{\mathbf{H}(\boldsymbol{\omega})\}$

$$\lambda_{\min}\left\{(\mathbf{I}+\boldsymbol{\omega}\boldsymbol{\omega}^{\top}\odot\mathbf{L})^{-1}\right\}\geq\frac{1}{1+\lambda_{\max}(\mathbf{L})\max\left\{\omega_i^2\right\}}.$$

Finally, sufficient conditions for $\mathrm{mse}(\boldsymbol{\omega})\leq\mathrm{mse}(\omega_0)$ are

$$\begin{aligned}&\omega_i\geq\sqrt{\omega_0}>0,\text{ for } i=1,2,\ldots,n\\&\frac{1}{1+\lambda_{\max}(\mathbf{L})\max\left\{\omega_i^2\right\}}+\frac{1}{1+\omega_0\lambda_{\max}(\mathbf{L})}\geq 2\gamma,\end{aligned}$$

which complete the proof.

## B.4. PROOF OF COROLLARY 1

If we simply let the following two hold

$$\begin{aligned}&\frac{1}{1+\lambda_{\max}(\mathbf{L})\max\left\{\omega_i^2\right\}}\geq\gamma,\\&\frac{1}{1+\omega_0\lambda_{\max}(\mathbf{L})}\geq\gamma,\end{aligned}$$

then we have the condition (6.9a) sufficiently satisfied. Since $\gamma=\rho/(1+\rho)=1/(1+\frac{1}{\rho})$, we substitute this relation into above two conditions, then we have

$$\begin{aligned}&\max\left\{\omega_i^2\right\}\lambda_{\max}(\mathbf{L})\leq\frac{1}{\rho}\\&\omega_0\lambda_{\max}(\mathbf{L})\leq\frac{1}{\rho}\\&\omega_i\geq\sqrt{\omega_0}>0,\text{ for } i=1,2,\ldots,n\end{aligned}$$

The second one can be omitted since $\max\left\{\omega_i^2\right\}\geq\omega_0$, which completes the proof.

## B.5. SIMPLIFYING THE COST FUNCTION FOR PROBLEM (6.18)

First we expand the inner term as follows

$$\begin{aligned}&\mathbb{E}\left\{\|\mathbf{y}-(\mathbf{I}+\boldsymbol{\Omega}\odot\mathbf{L})\mathbf{x}^*\|_2^2\right\}\\&=\mathbb{E}\left\{\mathrm{tr}\{[\mathbf{y}-(\mathbf{I}+\Omega\odot\mathbf{L})\mathbf{x}^*]\cdot[\mathbf{y}-(\mathbf{I}+\Omega\odot\mathbf{L})\mathbf{x}^*]^{\top}\}\right\}\\&=\mathbb{E}\left\{\mathrm{tr}\{\mathbf{y}\mathbf{y}^T-2\mathbf{y}\mathbf{x}^{*T}(\mathbf{I}+\Omega\odot\mathbf{L})+(\mathbf{I}+\Omega\odot\mathbf{L})^2\mathbf{x}^*\mathbf{x}^{*T}\}\right\}.\end{aligned}$$

Since the trace operation can be switched with expectation operation, the cost function is equal to

$$\mathrm{tr}\{\mathbb{E}(\mathbf{y}\mathbf{y}^\top) - 2\mathbb{E}(\mathbf{y}\mathbf{x}^{*T})(\mathbf{I}+\Omega\odot\mathbf{L}) + (\mathbf{I}+\Omega\odot\mathbf{L})^2\mathbf{x}^*\mathbf{x}^{*T}\}.$$

Further, since $\mathbb{E}(\mathbf{y}\mathbf{y}^\top) = \mathbf{\Sigma} + \mathbf{x}^*\mathbf{x}^{*T}$ and $\mathbb{E}(\mathbf{y}\mathbf{x}^{*T}) = \mathbf{x}^*\mathbf{x}^{*T}$, the above is equivalent to

$$\begin{aligned}
&\mathrm{tr}\{\mathbf{\Sigma} + \mathbf{x}^*\mathbf{x}^{*T} - 2\mathbf{x}^*\mathbf{x}^{*T}(\mathbf{I}+\Omega\odot\mathbf{L}) + (\mathbf{I}+\Omega\odot\mathbf{L})^2\mathbf{x}^*\mathbf{x}^{*T}\} \\
&= \mathrm{tr}\{\mathbf{\Sigma} + [\mathbf{I} - (\mathbf{I}+\Omega\odot\mathbf{L})]^2\mathbf{x}^*\mathbf{x}^{*T}\} \\
&= \mathrm{tr}\{\mathbf{\Sigma} + (\Omega\odot\mathbf{L})^2\mathbf{x}^*\mathbf{x}^{*T}\}.
\end{aligned}$$

As our optimization variable is $\Omega$, we can drop the unrelated covariance matrix $\mathbf{\Sigma}$, which gives the cost function in problem (6.19).

## B.6. A DETAILED EXPRESSION OF THE COST FUNCTION IN (6.20)

The cost function in (6.20) is

$$\mathbb{E}\left\{\|\mathbf{H}(\Omega)\mathbf{y} - \mathbf{x}^*\|_2^2\right\},$$

which can be expressed as follows

$$\mathbb{E}\{\mathrm{tr}(\mathbf{H}^2(\Omega)\mathbf{y}\mathbf{y}^\top - 2\mathbf{H}(\Omega)\mathbf{y}\mathbf{x}^{*\top} + \mathbf{x}^*\mathbf{x}^{*\top})\}.$$

Then we exchange the expectation and trace operator and we have the following detailed expression of the cost function

$$\mathrm{tr}\{(\mathbf{H}^2(\Omega)) - 2\mathbf{H}(\Omega) + \mathbf{I})\mathbf{x}^*\mathbf{x}^{*\top} + \mathbf{H}^2(\Omega)\mathbf{\Sigma}\}.$$

From this expression, we can see that this cost function indeed includes the consideration of both the signal and noise.

## REFERENCES

[1] R. Horn and C. Johnson, *Matrix Analysis*, Matrix Analysis (Cambridge University Press, 2013).

# C

# A SIMPLE NODE-ADAPTIVE GRAPH REGULARIZATION WEIGHTS DESIGN

In Part II, we proposed the node-adaptive graph signal regularization to promote a more general smoothness assumption on the graph signals. The parameters allow for flexibility is the NA regularization weight vector $\boldsymbol{\omega}$. The design of such vector is critical for its performance.

As we have seen in section 6.4, the weight design methods minimizes the MSE to design the parameters, but they are dependent on the ground truth signals. Though we proposed the *min-max* formulation to adapt the methods, it has limited performance improvement as in section 6.5. The weight design methods also heavily rely on solving optimization problems. This is impractical in real world problems, and hurdles the NA regularizer applicability. To make the NA regularizer more practical, we need a simple and intuitive parameter design scheme. For example, in Tikhonov regularization, as reviewed in subsection 5.2.6 and [1] , the increase of $\omega_0$ will lead to a monotonic change in bias and variance. The optimal design of $\omega_0$ does not rely on solving an optimization problem [1].

Thus, in this chapter, we revisit the NA graph signal regularization from another topological perspective. Then, we propose a simple NA weight design methods based on a so-called *constant transform* equation. We follow it by extensive descriptive experiments on synthetic and real world data. Afterwards, along with the idea of *constant transform*, we give an insight of a *smooth transform* based graph signal processing framework.

## C.1. REVISIT NODE-ADAPTIVE GRAPH SIGNAL REGULARIZATION

If a graph signal is smooth over the underlying graph, then we can reconstruct it using the graph topology knowledge. When we look at the Tikhonov regularization from an-

other perspective, we can see (5.15) as

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{x}^\top \mathbf{L}\mathbf{x} \\ \text{s.t.} \quad & \|\mathbf{y}-\mathbf{x}\|_2 \leq \delta, \end{aligned} \tag{C.1}$$

where $\delta$ is a predefined fitting error tolerance. Basically, the goal of Tikhonov regularization is to minimize the signal smoothness subject to an acceptable tolerance on the fitting error. We can see that that *if a signal is constant over the graph, then we can reconstruct this signal with great confidence by Tikhonov approach.*

However, can we not rely on the smoothness assumption to reconstruct the graph signal? Can we reconstruct a graph signal that has more high frequency components? Or can we reconstruct a graph signal that has very random low and high frequency components? With *node-adaptive regularization*, we can achieve a more general signal prior and explore a simple way of the regularization parameters design.

### C.1.1. NA GRAPH SIGNAL REGULARIZATION FROM CONSTANT TRANSFORM

The core idea to answer the previous questions is: if we transform an arbitrary signal into a smoother version, we can use the fact that this "new" signal is smooth enough to be reconstructed. Then, we can recover the original signal if the previous smooth transform is known and invertible.

This smooth transform can be achieved by a nodal loading vector $\boldsymbol{\omega}_{nl}$ on the arbitrary graph signal $\mathbf{x}$. This transform converts the signal into a constant signal, the smoothest signal that one can achieve. Mathematically, we essentially impose the following so-called *constant transform equation*

$$\text{diag}(\boldsymbol{\omega}_{nl})\mathbf{x} = c\mathbf{1}, \tag{C.2}$$

where $c$ is a predefined a constant. This transform is carried out by the NA regularization parameters $\boldsymbol{\omega}$ weighted over the graph signal $\mathbf{x}$. If we view the NA regularization (6.3) from the perspective of (C.1), then it is seen as follows

$$\begin{aligned} \min_{\mathbf{x}} \quad & (\text{diag}(\boldsymbol{\omega})\mathbf{x})^\top \mathbf{L}(\text{diag}(\boldsymbol{\omega})\mathbf{x}) \\ \text{s.t.} \quad & \|\mathbf{y}-\mathbf{x}\|_2 \leq \delta. \end{aligned} \tag{C.3}$$

Problem (C.3) recovers the original graph signal $\mathbf{x}$ by guaranteeing that its "transformed" version $\text{diag}(\boldsymbol{\omega})\mathbf{x}$ is globally smooth. That is, it minimizes the variation of the "transformed" signal, i.e., $(\text{diag}(\boldsymbol{\omega})\mathbf{x})^\top \mathbf{L}(\text{diag}(\boldsymbol{\omega})\mathbf{x})$. The latter is equivalent to our proposed node-adaptive regularization (6.3)

$$\min_{\mathbf{x}} \quad \|\mathbf{y}-\mathbf{x}\|_2^2 + (\text{diag}(\boldsymbol{\omega})\mathbf{x})^\top \mathbf{L}(\text{diag}(\boldsymbol{\omega})\mathbf{x}), \tag{C.4}$$

where the solution has the closed form (6.4) and the fitting term and regularization term can be traded by scaling the NA weights.

In short, our NA regularization recovers the signal without any smoothness assumptions by minimizing the signal variation of the loaded signal. Furthermore, if the NA weights follow the constant transform equation (C.2), then NA regularization looks for the signal estimate by minimizing the signal variation of a constant signal.

### C.1.2. CONSTANT TRANSFORM BASED NA WEIGHT DESIGN

We follow the constant transform equation (C.2) to design the NA weights. We do not differentiate the loading vector and NA weights anymore, since they act the same role. From (C.2), NA weights can follow

$$\text{diag}(\boldsymbol{\omega})\mathbf{x} = \mathbf{1}, \tag{C.5}$$

where the all-ones vector $\mathbf{1}$ can be scaled. We refer to (C.5) as the *constant transform design*. However, this is not practical as the ground truth signal $\mathbf{x}$ is required. To make it practical, we replace the true signal $\mathbf{x}$ by its estimate $\hat{\mathbf{x}}$, that is,

$$\text{diag}(\boldsymbol{\omega})\hat{\mathbf{x}} = \mathbf{1}. \tag{C.6}$$

What is important is to obtain a good estimate of the true signals which is inverse to the NA parameters. We consider the following situation.

**Practical consideration** Suppose we have a set of noisy realizations $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_M\}$ or we have real world data recordings like the Molene weather data or NOAA temperature data in section 6.5, with dimension of $N \times F$, where $N$ is the length of the data per observation, and $F$ is the number of observation at different time. We use a small (or large, even all, depending on the requirements) set of the noisy observations to train NA weights according to the following

$$\text{diag}(\boldsymbol{\omega}) = \mathbf{1} \odot^{-1} (\bar{\mathbf{y}}), \tag{C.7}$$

where $\odot^{-1}$ is the inverse Hadamard operator, i.e., the element-wise division and $\bar{\mathbf{y}}$ is the mean of the certain number of noisy signal realizations.

This is because under the given signal model $\mathbf{y} = \mathbf{x} + \mathbf{n}$, for deterministic case, the mean of all noisy measurements is a minimum variance unbiased (MVU) estimator, i.e., $\hat{\mathbf{x}}_{MVU} = \bar{\mathbf{y}}$, which can result in a good estimate of $\mathbf{x}$ when the number of measurements is properly large. As we will see later in the experiment part, with this consideration, we can basically obtain a performance close to the design (C.5) with true signals.

Note that the constant vector $\mathbf{1}$ needs to be scaled properly with a scalar $c$. In cases where the data value is large, for instance, the Molene and NOAA weather data in the real data experiment, if we do not scale the parameters, the regularization term is trivial and does not fit the data. In the following synthetic experiments, we set $c = 1$, but for the real data ones, we tune the value of $c$ to achieve a good performance.

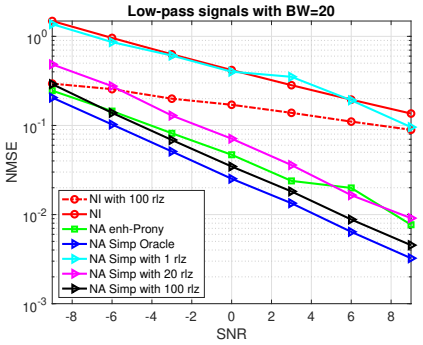
## C.2. NUMERICAL RESULTS

In this section, we analyze the behavior of the simple design of the NA regularization and compare it with NI regularization.

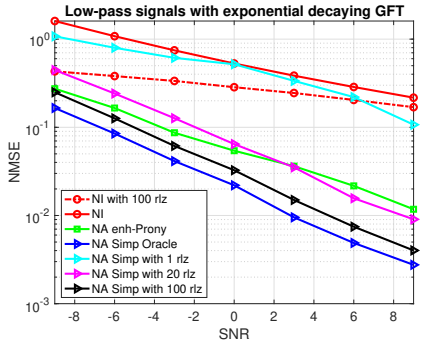
### C.2.1. SYNTHETIC DATA

First, we conduct experiments on synthetic data, where we consider multiple realizations of the noisy graph signals are available, and the mean of the realizations are used in NI and NA estimates. The experimental setting is same as before in section 6.5. But here we consider the following different graph signals

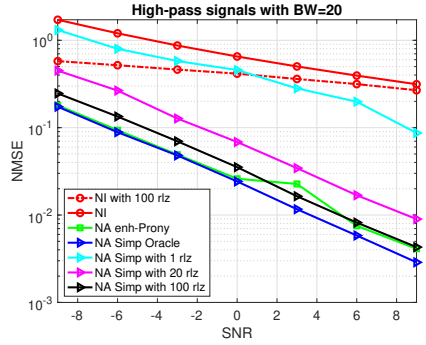
- Low-pass smooth signals with its GFT is constantly one in the first 20 graph frequencies, i.e., $\tilde{x}_i = 1$, for $i = 1, \ldots, 20$, and the rest are zeros.
- Low-pass smooth signals with decaying GFT in the first 20 graph frequencies, i.e., $\tilde{x}_i = \frac{1}{1+\lambda_i}$, for $i = 1, \ldots, 20$, and the rest are zeros.
- High-pass signals with constant GFT in the last 20 graph frequencies, i.e., $\tilde{x}_i = 1$, for $i = 31, \ldots, 50$, and the rest are zeros.
- Bandlimited graph signals with bandwidth of 20, and the GFT within the band follows a Gaussian distribution $\mathcal{N}(0, \mathbf{L}^{\dagger})$.



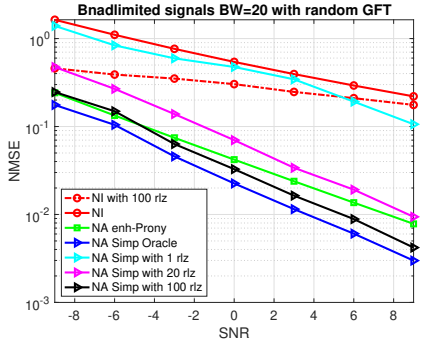

(a) Low-pass smooth signals with constant GFT in the bandwidth of 20.




(b) Low-pass smooth signals with exponential decaying GFT in the bandwidth of 20.




(c) High-pass signals with constant GFT in the bandwidth of 20.




(d) Bandlimited signals with random GFT in-band.

Figure C.1: Denoising performance of the simple NA weights design when many realizations are available.

For NI regularization method, we use the optimal parameter design from [1] and we also take the enhanced-Prony method as comparisons. For our simple design of NA regularization, we first consider the oracle design (C.5) by assuming that the true signal

is available as the benchmark. Then, we consider the practical situation (C.7) by making use of the mean of 1, 20 and 100 signals noisy realizations.

From the experiment results, shown in Figure C.1, we confirm our previous argument that NA regularization does not necessarily rely on the global smoothness assumption. When multiple noisy signal realizations are available, for signals with different characteristics, no matter low-pass being smooth, high-pass being non-smooth or even random, the NA regularization beats the NI regularization. Our simple design based on constant-transform has a comparable performance with the best what is achieved before based on enhanced-Prony method. When the number of noisy signal realizations improves, we can get the performance closer to the oracle design. For high-pass signals, we observe that even with one noisy realization, we can achieve a much better performance compared to the NI approach.

Note that all over the experiments, we also evaluated the performance of the NI approach on the mean of 100 noisy realizations for tighter comparison. We easily see that by increasing the number of realizations, one cannot improve much on NI approach, which is also not even comparable to our approach when the number of realizations is properly large. Moreover, the NA estimate has at least three times better NMSE performance the the simple mean of the multiple realizations, which attributes to the NA regularization scheme.

In general, if we have many noisy measurements or realizations of one graph signal, it is worth to have more discussions on the question of, should we consider the Tikhonov method working on the mean of these realizations or on each one of them, then measure the NMSE? It is reasonable to consider the Tikhonov method working on each single realization, then get the NMSE for each, and average them as a numerical NMSE measure. Regarding the results above, I reported the theoretical NMSE expression for each method and it is close to numerical NMSE when the number of realizations are large, so we believe that those results are valid and able to reflect the ability of our methods.

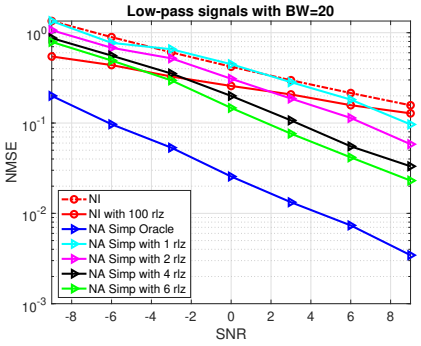
Actually we also tried the other case, applying on the Tikhonov method on the mean of the total, (e.g. 100 in this example) realizations, since we used at most total realizations to design the loading vector. Even in this case, our methods beat the NI regularization when the number of realization is larger than around 10.

### C.2.2. EFFECT OF THE NUMBER OF REALIZATIONS

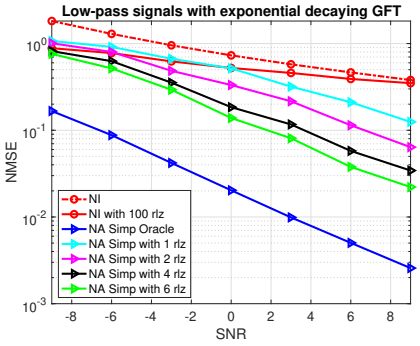
In this section, we discuss the effect of the number of noisy realizations on the simple NA regularization performance. We conduct several experiments and evaluate the performance by increasing the number of realization two by two. The oracle design and Tikhonov regularization on one realization and 100 realizations are used for comparisons.

In Figure C.2, we show the denoising performance when the number of realizations is $\{1, 2, 4, 6\}$. We observe that the simple design can perform well compared to the counterpart, Tikhonov regularization even with 100 realizations. These results indicate that the simple design method is robust in practice.

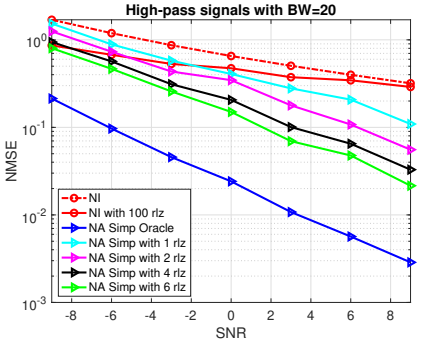
When the number of realizations increases gradually, the simple NA design performs better. This is benefiting from a better estimate of the graph signals by averaging more number of realizations. But without the simple NA design method, the mean estimate



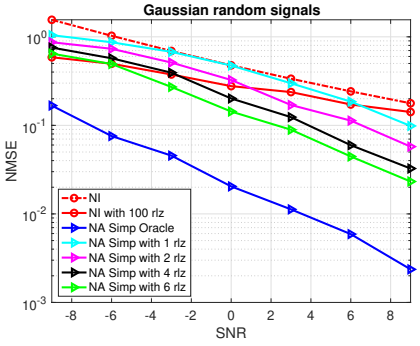

(a) Low-pass signals with constant GFT in the bandwidth of 20.




(b) Low-pass signal with exponential decaying GFT.




(c) Low-pass signals with constant GFT in the bandwidth of 20.




(d) Low-pass signal with exponential decaying GFT.

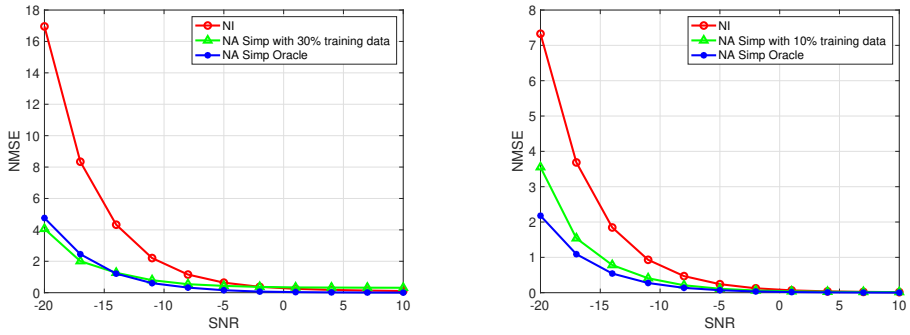
Figure C.2: Denoising performance when number of realizations is increasing.

still gives three times worse performance as observed in above synthetic results.

### C.2.3. REAL DATA

In this section, we investigate the behavior of the constant transform based NA weights design on real data. To keep the consistency, we use the Molene weather dataset and the NOAA dataset (the former consists of a dimension of $32 \times 744$ hourly temperature data and the latter data has a dimension of $109 \times 8759$, see in section 6.5). This is essentially different from the synthetic experiments in the last section where we have many realizations of one signal, while now multiple observations are obtained over time. Each observation is a separate signal, though sharing same pattern due to the physical fact.

To denoise with these data with the simple constant transform designed NA regularization, we first split the whole data into the training set and the testing set. Then we average the training data and with the mean which reveals the true signal pattern, we use the simple constant transform equation to obtain the NA parameters by point-wise




(a) Molene weather data.

(b) NOAA data.

Figure C.3: Simple constant transform based NA regularization behavior on real data.

division as in (C.7). Finally we use the trained NA parameters to denoise the remaining testing data. The reason that we are averaging the training data, is because these observations are obtained on a fixed physical setting. It means the pattern of the observations are similar and is able to reflect the true signal pattern. This is significant in the constant transform equation.

Figure C.3 shows the denoising performance with respect to the different noise level, which is added to generate SNRs of $\{-20\,\text{dB}, -17\,\text{dB}, \ldots, 10\,\text{dB}\}$. We compare with the Tikhonov regularization with the optimal parameter setting and use the oracle constant transform design as the benchmark. From the curves, the performance of NA regularization constantly outperforms the Tikhonov method and we also observe that even with small number of training data, the trained NA parameters can achieve performance close to the oracle design. Note that since the data values are large in these two dataset, we scale the trained and oracle NA parameters by hand-tuning them till a good performance is obtained.

This indicates the practicality of our method when dealing with data like Molene and NOAA temperature recordings. We can use certain small number of observations to train and get a good set of NA parameters based on the constant transform equations, then process the remaining data with NA regularization method, as we discussed in the practical consideration.

## C.3. DISCUSSION AND CONCLUSION

Inspired by the simple NA weights design, we discuss a more general GSP framework based on smooth transform. We consider a constant signal (all nodal values are one) over a ring graph with 16 nodes. We can use any low pass filter to denoise such signal, for instance, typical Tikhonov denoising, as shown in Figure C.4, because the constant signal is globally smooth over the graph.

However, if the first nodal signal value became 10 significantly without being known, it is not clear then how to handle this situation. Since the signal global smoothness is

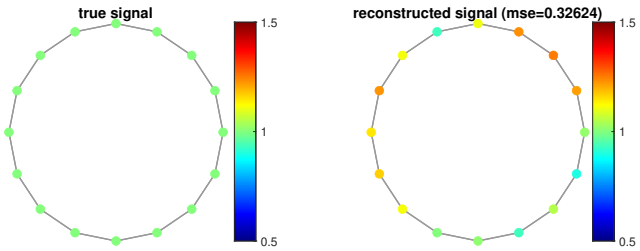



Figure C.4: Typical Tikhonov denoising of a constant signal over a ring graph with noise variance of 0.2 and regularization parameter 0.5.

broken, our idea is to design a loading vector on the signal such that the loaded true signal is globally smooth. As we know, a constant signal is a best choice for smoothness, since its signal variation is zero. Then, we load the noisy measurement and denoise this loaded (transformed) noisy signal by any smoothness-based denoising method. Finally we can retract the original signal by the loading vector we designed.

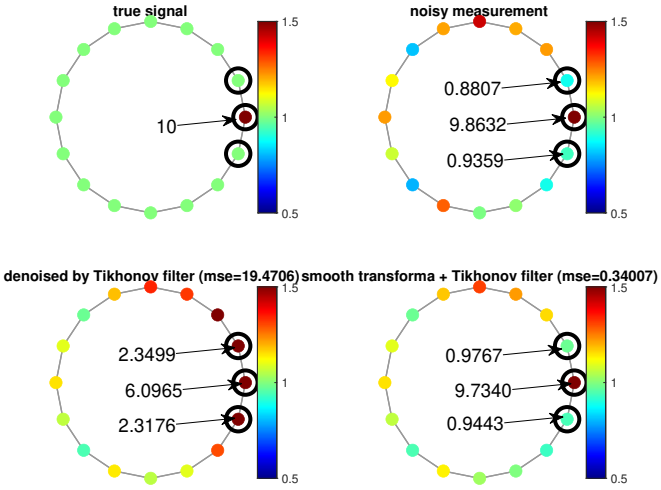



Figure C.5: Comparison of Tikhonov denoising and smooth transform based Tikhonov denoising of a non-smooth signal over a ring graph with noise variance of 0.2, Tikhonov regularization parameter 0.5 and loading vector $\boldsymbol{\omega}$ following $\mathrm{diag}(\boldsymbol{\omega})\mathbf{x} = \mathbf{1}$.

Such an example is shown in Figure C.5 and analyzed in graph spectral domain Figure C.6. In graph spectral domain, we observe that the original signal has many high frequency components, while using the pure Tikhonov filter denoising, the high frequency components will be suppressed, which degrades the performance. However, if we load the signal such that the noisy measurements have only low frequency components, then

we can achieve a good denoising performance with Tikhonov regularization such denoising methods relying on global smoothness. Finally, we can unload the nodal weights to reconstruct a good estimate of the signal.

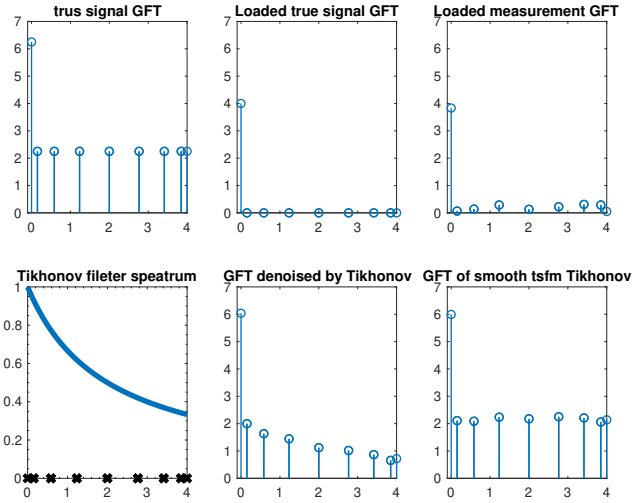



Figure C.6: Denoising of Tikhonov method and smooth transform based Tikhonov method in graph frequency domain.

**Observations.** We observe from Figure C.5 that Tikhonov regularization achieves a high MSE around 20, while the smooth transform plus Tikhonov denoising achievers an MSE of 0.34. More specifically, in Tikhonov denoising, we observe that the critical node (with value of 10) influences and gets influenced by the adjacent nodes, which results in the bad performance and is due to the global smooth assumption. Instead, a smooth transform makes the global smoothness reliable and true for Tikhonov regularization.

### C.3.1. FUTURE WORK

Finally, motivated by this, we propose the following general smooth transform based denoising scheme, which can escape the assumption on the global signal smoothness, shown in Figure C.7. This is left as future work. The goal of proposing it is to bridge us to a better way of understanding our node-adaptive regularization and a simple way of designing the NA parameters.

The smooth transform idea has some similarities with neural networks (NNs). The seminal work of explaining the power of convolutions NNs considers *scattering transforms* as the information processing architectures akin to CNNs [2]. For ones who are familiar with neural networks, they are nothing but convolutions with filter banks followed by nonlinear mapping. The reason that why NNs perform superior to the pure convolutions, is that, on the one hand, linear filter banks can only provide the stable information processing ability in the low frequencies; on the other hand, the nonlin-

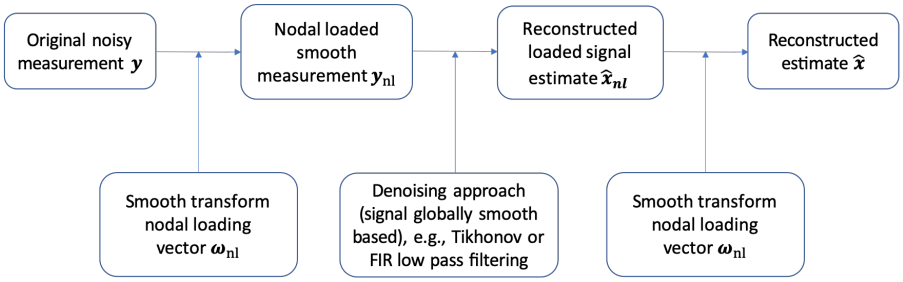



Figure C.7: Smooth transform based denoising framework

ear operation acts as a frequency mixer which brings part of the high frequency energy towards low frequencies where it can be discriminated with stable filters [2]. Our NA regularization and the smooth transform based denoising framework are along with the similar idea, to transform the signal as low-pass, then the smoothness based methods can be applied.

## C.4. CONCLUSION

In this chapter, we mainly propose a simple NA weight design method, which relies on a constant transformation equation. The idea of the method is, first see the NA weights as the transforming vector, then transform the arbitrary graph signals into the smoothest graph signal, constant signals. In this way, a simple elementwise division design method is proposed. When multiple noisy realizations are available, the method works well, as investigated in the synthetic and real data numerical results. Along with the similar idea, we propose a smooth transform based GSP framework, left as future work.

## REFERENCES

[1] P. Chen and S. Liu, *Bias-variance tradeoff of graph laplacian regularizer,* IEEE Signal Processing Letters **24**, 1118 (2017).

[2] F. Gama, J. Bruna, and A. Ribeiro, *Stability properties of graph neural networks,* (2019), arXiv:1905.04497 [cs.LG] .