EPFL

Thèse n° 9814

Quantization for Distributed Processing and Learning of Structured Data

Présentée le 19 décembre 2022

Faculté des sciences et techniques de l'ingénieur
Laboratoire de traitement des signaux 4
Programme doctoral en génie électrique

pour l'obtention du grade de Docteur ès Sciences

par

Isabela CUNHA MAIA NOBRE

Acceptée sur proposition du jury

Prof. J.-Ph. Thiran, président du jury
Prof. P. Frossard, directeur de thèse
Prof. R. Nassif, rapporteuse
Prof. P. S. R. Diniz, rapporteur
Dr M. Mattavelli, rapporteur

École polytechnique fédérale de Lausanne

2022

It's the possibility of having a dream
come true that makes life interesting.
— Paulo Coelho, The Alchemist

With love to my family…

Acknowledgements

My PhD studies[1] have taken me on a very rich and experience-filled journey that has helped me grow both academically and personally. Thus, I want to take a moment to express my gratitude to everyone who helped make these experiences possible.

I would like to first thank my PhD advisor, Prof. Pascal Frossard, for giving me this great opportunity. Thank you Pascal for trusting me in choosing my research topic. I appreciate the discussions, suggestions, and in-depth feedback throughout my PhD journey. Thank you also for checking in on me when I was sick.

I would also like to thank the jury members of my thesis committee. Thank you Prof. Jean-Philippe Thiran, Prof. Roula Nassif, Prof. Paulo Diniz and Dr. Marco Mattavelli for taking the time to evaluate my thesis work to give me your much appreciated feedback.

I found extremely enjoyable the supervision of student projects during my PhD. Special thanks to my master student Daniel, for all his great work. Daniel, you are brilliant and it was such a joy working with you. I am excited to see you grow into a bright future.

Further, I would like to sincerely thank all the current and past LTS4 labmates Abdellah, Ádám, Ahmet, Alessandro, Apostolos, Arun, Bastien, Beril, Carol, Clément, Clémentine, Cristina, Dorina, Eda, Ersi, Guillermo, Harshitha, Hermina, Hossein, Javier, Jelena, Marcele, Mattia, Nikos, Ortal, Renata, Roberto, Sahar, Seyed, Stefano, Vaishnavi, William, Yamin for the great time we spent together and for our discussions. Special thanks go out to: Eda, you were the very first person I met in LTS4, thank you for helping me to set up in the lab in a time when everyone was on Christmas break. Also thank you for the delicious hand-made coffees and Netflix sessions. Ahmet, thank you for your kindness and availability in the many times I asked for help. Our discussions were appreciated, and specially thank you for the very useful feedback on my private defence dry-run. Same for Carol, thank you for the discussions on my dry-run. Ersi, thank you for the many lovely lunches we had together. Thank you for our light and positive conversations. Marcele, thank you so much for being someone I could count on difficult times. Cristina, you were here for such a short time, but you made my internship experience at LTS4 a very pleasant one. You are very cool and fun, I am glad I kept your friendship after you left the lab, it was wonderful meeting you later again in Padua. Thank you Hermina for the many useful tips and advices. Thank you Clément for your help translating the abstract of this thesis into French, for the ski tips and the volleyball matches we played together. Thank you Harshita and Javier for the lively exchanges. I would also like to give my

[1] This PhD thesis was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001 (grant number 88881.174577/2018-01). I am grateful for their support.

special thanks to Anne for her kindness, assistance and efficiency.
Then I would like to express my gratitude to all my friends in Switzerland. Destiny wanted me to meet Mariana, who is both a Brazilian (from my home town Maceió!) and a Swiss, allowing me to keep my home connections and at the same time to create new connections with the country I live. Mariana, I always have a great time seeing you, thank you for the many times you welcomed me to your home in Basel. To my dear Crazy Pilgrims, life during my PhD would not have been the same without you. What a fun, adventurous, exciting group! The UN of friendships, representing every single continent on this planet. Thanks to you I could explore this beautiful country and keep so much energy and enthusiasm in my daily life. Jacob, you were the first pilgrim I met, thank you for reaching out that first time, when I was lost. Thank you for exchanging messages with me while I was still in Brazil and thank you for the many tips before I moved into Switzerland. Thank you for introducing me to this group and specially thank you for (unintentionally) introducing me (twice!) to Chris. Ludo and Mira, you are such a power couple! I had the absolute joy seeing you getting married this year. Thank you for your great vibes and enthusiasm. Nikki, your History knowledge is amazing, you would make a great tour guide. Thanks for always taking the best shots and registering our moments together. It will be very interesting following your next adventures in Thailand. Stephane, Damn! The greatest pilgrim of all (sorry to the others), you are always there in every single journey. Your love for nature reminds us why we love Switzerland, and I thank you for that. Beatriz, you were here for a very short time, but it definitely felt longer. Thank you for being such a good neighbour and friend, and for always offering a place to stay in Lisbon. I will never forget you flew all over to Switzerland just for a pool party. Tom, please never lose your child-like temperament, it is endearing to all of us. No worries, the baby plant you gave me is in good hands. Oussama and Evan, it is always a delight hanging out with you.
I love EPFL's (and Switzerland's) incredibly international environment, it was very enriching to learn from people from all corners of the world. However, I believe that when living abroad, it is important to try our best to connect with the locals. And I managed to break my international bubble thanks to volleyball. Through the sport I came across many awesome people, Swiss and international. I am thankful for all of you for the great matches and the healthy life balance they provided me. The most immersive experience I had in this country was being part of a club in a Swiss amateur league. Thank you Natacha for being an inspiring coach and for the girls in the team for making us reach the final in the *Championnat Vaudois*. Outside of the club, I thank you Daniel, the most brilliant volleyball player I know, for the great time we had together and for being such a good friend. It was fun discovering snow volleyball together. Ehsan, your Sunday's volleyball at EPFL was part of my routine during the winter seasons preceding Covid, and I thank you for all the effort on weekly organizing that. Flaviano, such a delight (and coincidence) meeting another person from Maceió! Thank you for all the invitations for dinner at your home, along with Estefani. Our matches were really fun.
A big thank you also to all of my friends from Brazil who, despite of the distance, give me a warm welcome every time I go back. Thank you all for the support and encouragement.
I am also extremely thankful for the loving family I have. Special thanks to my grandparents Acilea, Aldo, Vinicius and Zelia, for inspiring me with good values and for being positive

examples in life. Thank you for your support and for always believing in me. Gabi, thank you for visiting me here in Switzerland, what a marvellous time we had together and I will never forget it. Thank you for being such an affectionate, humane and adventurous sister. Guiga, you are a brave, peaceful and kind man, and I admire you deeply. I wish you could have come here as well my brother, but unfortunately Covid spoiled our plans. I guess we will have to make a trip together here, I am sure you are going to love it.
A huge thank you Chris, for being with me during all the years of my PhD. You watched me practice the private defence many times and I am very grateful for that. You were my strength. You suffered with me when I suffered, but much more importantly (and frequently), you rejoiced with me when I rejoiced. I am so lucky to be contaminated with your positivity. You are very funny, and can lighten up any mood I am in. You make me fly!
Finally I thank you, my parents, for your unconditional love and unfailing support. Thank you for always believing in me. I don't find the words to express how grateful I am to have you in my life. I was born during a PhD abroad and that might mean something. Mom, thanks for coming with me to Lausanne at the very beginning, your help setting me up was greatly appreciated. Dad, thanks for encouraging me to visit this country before my PhD started, so that I would be sure of my choice. Thank you for praying everyday for me during these years. I love you both so much.
This accomplishment would not have been possible without all of you. Thank you.

Lausanne, December 6, 2022 Isabela Nobre

Abstract

In the domains of machine learning, data science and signal processing, graph or network data, is becoming increasingly popular. It represents a large portion of the data in computer, transportation systems, energy networks, social, biological, and other scientific applications. Often, such data is physically distributed over different network nodes, and there is a communication cost involved with bringing it to a central unit for processing and analysis. Decentralized algorithms offer solutions to deal with network data and relax communication costs, with nodes sharing messages over communication channels in order to jointly implement data processing or learning tasks. However, messages are typically quantized in practice and represented by a finite number of bits in digital communication channels. As a result, imperfections in received signals may accumulate and eventually degrade the algorithm's overall performance. This thesis focuses on designing new methods to efficiently allocate bits in the different steps of messages exchanges between network nodes when implementing distributed graph signal tasks. First, we consider graph filters that can decompose and shape graph signal frequency components, in order to realize a desired response. Distributed graph filters can be used in applications such as smoothing, denoising and semi-supervised learning. We propose an optimal bit allocation technique that adapts to the network topology and the message importance, such that it minimizes the quantization error. Second, we consider distributed Graph Neural Networks that can be used in applications such as anomaly detection, decentralized control and traffic prediction. We study the effect of quantization in the GNN inference stage and we propose an analytical solution to an optimized bit allocation problem, by solving the corresponding Karush–Kuhn–Tucker (KKT) system of equations. Our method is shown to be beneficial in reducing the error due to quantization, compared to other baselines, on the tasks of distributed denoising and distributed source localization. The optimized bit allocation gives a higher relevance to messages in the middle layers of the neural network model. Finally, we consider the distributed graph learning problem whose objective is to infer an unknown data graph from network observations, in order to enable further processing tasks or interpretability. We propose a novel distributed graph learning algorithm under the assumption that the data is smooth on the data graph. With the use of local projection constraints, we solve the distributed optimization problem and infer a valid graph. For the same accuracy, our distributed algorithm has a lower communication cost compared to a centralized version, especially for sparse networks. Additionally, we propose a bit allocation scheme for the distributed graph learning algorithm. We show that the scheme presents a better accuracy and bit cost trade-off than a baseline uniform bit allocation scheme. Overall, this thesis proposes

novel bit allocation techniques for signal quantization in distributed implementations of signal processing and machine learning tasks. We believe that our research efforts will hasten the development of intelligent distributed processing algorithms for network data that balance performance, communication bandwidth, and computational complexity, in a wide range of potential applications in social, sensor, energy, transportation, and other fields.
Keywords: graph signal processing, wireless sensor networks, distributed algorithms, quantization, graph filtering, graph neural networks, graph learning.

Résumé

Dans les domaines de l'apprentissage automatique, de la science des données et du traitement du signal, les données organisées sous forme de graphes sont de plus en plus populaires. Les graphes sont une façon naturelle de représenter les données issues de l'informatique, des systèmes de transport, des réseaux énergétiques, des applications sociales, biologiques et d'autres applications scientifiques. Souvent, ces données sont physiquement distribuées sur différents nœuds d'un réseau, et leur acheminement vers un processeur central entraîne un coût de communication. Les algorithmes décentralisés offrent des solutions pour traiter les données distribuées des graphes, les nœuds partageant des messages sur des canaux de communication afin d'exécuter conjointement des tâches de traitement des données ou d'apprentissage. En pratique, les messages sont quantifiés et représentés par un nombre fini de bits. Par conséquent, les imperfections dans les signaux reçus s'accumulent et finissent par dégrader la performance globale de l'algorithme. Cette thèse se concentre sur la conception de nouvelles méthodes pour allouer efficacement les bits dans les différentes étapes des échanges de messages entre les nœuds, lors de l'implémentation de tâches distribuées de signaux de graphes. Tout d'abord, nous considérons les filtres de graphe qui peuvent décomposer et façonner les composantes de fréquence d'un signal sur un graphe, afin de réaliser une réponse spectrale désirée. Les filtres de graphes distribués peuvent être utilisés dans des applications telles que le lissage, le débruitage et l'apprentissage semi-supervisé. Nous proposons une technique d'allocation optimale des bits qui s'adapte à la topologie du réseau et à l'importance du message et qui s'avère efficace pour minimiser l'erreur de quantification, surpassant les algorithmes de base. Deuxièmement, nous considérons les réseaux neuronaux graphiques distribués qui peuvent être utilisés dans des applications telles que la détection d'anomalies, le contrôle décentralisé et la prédiction du trafic. Nous étudions l'effet de la quantification dans l'étape d'inférence du GNN et nous proposons une solution analytique à un problème d'allocation de bits optimisée, en résolvant le système d'équations de Karush-Kuhn-Tucker (KKT) correspondant. Nous montrons que notre méthode permet de réduire l'erreur due à la quantification pour les tâches de débruitage distribué et de localisation de sources distribuées. L'allocation optimisée des bits donne une plus grande pertinence aux messages dans les couches intermédiaires du modèle de réseau neuronal. Enfin, nous considérons le problème de l'apprentissage de graphes dont l'objectif est de déduire un graphe de données inconnu à partir d'observations, afin de permettre d'autres tâches de traitement ou d'interprétation. Nous proposons un nouvel algorithme d'apprentissage de graphe distribué en supposant que les données sont lisses sur le graphe de données. Grâce à l'utilisation de contraintes de projection

locales, nous résolvons le problème d'optimisation distribuée et déduisons un graphe valide. Pour la même précision, notre algorithme distribué a un coût de communication inférieur à celui d'un algorithme centralisé, en particulier pour les réseaux avec peu d'arêtes. De plus, nous proposons un schéma d'allocation de bits pour l'algorithme d'apprentissage de graphe distribué. Nous montrons que ce schéma présente un meilleur compromis entre la précision et le coût des bits que le schéma d'allocation de bits uniforme. Dans l'ensemble, cette thèse propose de nouvelles techniques de quantification dans les implémentations distribuées de tâches de traitement de signaux de graphes. Nous pensons que nos efforts de recherche accéléreront le développement d'algorithmes distribués intelligents pour les données de réseau qui équilibrent la performance, la bande passante de communication et la complexité de calcul, dans les vastes applications potentielles dans les domaines sociaux, des capteurs, de l'énergie, du transport et autres.

Mots clés : traitement du signal sur les graphes, réseaux de capteurs sans fil, algorithmes distribués, quantification, filtrage sur les graphes, réseaux neuronaux sur les graphes, apprentissage sur les graphes.

Contents

1 Introduction

1.1 Motivation

Structured data, which is defined over the nodes of a graph, is getting very popular nowadays in machine learning, data science and signal processing fields. Much of the data in computer, transportation systems, energy networks, social, biological and other scientific applications can be represented as graph data. This abundant and diverse resource needs to be processed and analyzed before becoming useful. The field of graph signal processing (GSP) provides the necessary tools to manipulate this type of data, also called graph signals. These tools are used in diverse applications, such as compression, denoising or reconstruction of sensor data [1, 2]. They also permit to design filters that process graph signals by attenuating their components at specific graph frequencies, in an analogous fashion to classical filters.

Centralised settings tend to assume that the data is available at no cost in a central processing unit. Often however, the observation data is physically separated and there is a communication cost associated with its collection, specially if there is not a direct communication path between sensor nodes and the central unit. Centralized solutions are not even sometimes viable, since they cause a bottleneck on the central processor, or are too costly in communication costs. Decentralized methods of graph signal processing emerged as solutions to handle data in large networks but also to deal with privacy concerns and also bandwidth/energy constraints [3–5]. Another advantage of distributed algorithms compared to centralized ones is that they can add robustness to the network in case of node failures.

There are many studies on distributed processing for graph signals or networked data [6–8]. However, only few deal with the fact that, in real case scenarios, the network is subject to communication constraints, which limit the precision of the messages exchanged by the different distributed algorithms. In order to collectively implement these algorithms, nodes should exchange information through messages that are quantized and represented by bits, with a finite precision. This leads to quantization errors in the transmitted data, which accumulate and eventually deteriorate the overall algorithm performance. Higher number of bits leads to less quantization error, but also to communication costs. Hence, a trade-off has

to be defined in practice, between distributed application performance and communication costs.

The overall objectives of this thesis is to find the optimal way to allocate bits in the different steps of messages exchanges between nodes, when implementing distributed graph signal processing tasks. We find that the ideal solution presents the overall best trade-off between minimum quantization error and minimum total bit costs, thus maintaining not only high accuracy but also respecting a bit budget constraint in the network. In particular, we study three important distributed tasks for structured data: graph filtering, graph neural networks and graph learning and devise optimized bit allocation algorithms for each of them.

Graph Filtering A signal can generally be decomposed in its frequency components and a filter can shape these components in order to realize a desired response, by isolating or enhancing some of them. Similarly, graph filters can process graph signals along their spectral components with the help of the graph Fourier transform. Graph convolutions can then be modeled by finite impulse response (FIR) graph filters. The FIR filter, which is the focus of our work, is defined as a linear shift-and-sum operation. In order to enable their distributed computation, linear graph filters can be approximated by shifted Chebyshev polynomials [3], becoming more amenable to distributive computing for applications such as smoothing, denoising, inverse filtering and semi-supervised learning.

Graph Neural Networks - GNNs GNNs are generalizations of neural networks for irregular data structures and are widely used in different fields. Their inference stage can be implemented distributively by allowing communication among neighbouring nodes. GNNs have been used for inference in distributed settings in many different applications, such as anomaly detection [9], consensus [10, 11], decentralized control [12–18], decentralized resource allocation [19], distributed regression [10], distributed scheduling [20, 21], source localization [10, 22] and traffic prediction [23]. In this work we focus on the Graph Convolutional Neural Networks (GCNN), whose fundamental building block is the FIR graph filter (or graph convolution). It consists of a concatenation of layers, where each layer is composed of a bank of graph FIR filters and a point-wise nonlinearity. Given that FIR filters can be computed in a distributed fashion, GCNN inherit this property as well.

Graph Learning In most graph signal processing algorithms, the network topology underlying the data is assumed to be known. However, in some settings, this is not the case. This gives rise to the problem of graph learning (GL)[24–31]. In the graph learning literature, the graph needs to be learned from the signals, so that processing tasks can be implemented by taking the data structure into account [32, 33]. A common assumption is that the signal values change smoothly across adjacent nodes of the unknown graph topology [34–38]. This assumption allows the inference of the data structure, which means that the graph learning task amounts to finding the graph structure on which signal values differences on nodes associated with the

same edge (and large weights) are minimized.

1.2 Thesis outline

The goal of this thesis is to optimize the bit allocation for distributed graph signal processing and learning tasks. The thesis is organized as follows:

In Chapter 2 we provide mathematical preliminaries that will be used throughout the thesis.

In Chapter 3 we propose a novel bit allocation method for distributed graph filtering that minimizes the error due to message quantization without compromising the communication costs. It first bounds the exchanged messages and then allocates a limited bit budget in an optimized way to the different messages and network nodes. In particular, our novel quantization algorithm adapts to both the network topology and the message importance in a distributed processing task. Our results show that the proposed method is effective in minimizing the error due to quantization and that it permits to outperform baseline distributed algorithms when the bit budget is limited. They further show that errors produced in nodes with high eccentricity or in the first steps of the distributed algorithm contribute more to the global error. Also, sparse and irregular graphs require more irregular bit distribution. Our method provides one of the first quantization solutions for distributed graph processing, which is able to adapt to the target task, the graph properties and the communication constraints.

In Chapter 4 we extend the study to Graph Neural Networks (GNN), which are highly used tools for processing and learning on graph-structured data. In the distributed implementation of GNNs, messages are exchanged between neighboring nodes in order to achieve a common objective. These messages usually have a finite precision, which may require quantization before transmission. Thus, errors arise in the received signals, which accumulate and eventually deteriorate the GNN model accuracy. We study here the effect of quantization in the GNN inference step and propose an analytical solution, with the use of the Karush–Kuhn–Tucker (KKT) conditions, to the optimized bit allocation problem, maintaining not only high accuracy but also obeying a bit budget constraint. Our experiments on distributed denoising and distributed source localization tasks show that our method is effective in minimizing the error due to quantization under a limited bit budget, compared to a uniform allocation and other baselines. It also shows that the optimal bit allocation tends to give a higher relevance to the messages in the middle layers of the model.

In Chapter 5 we introduce a distributed algorithm for the graph learning problem. Graph learning is generally performed centrally with a full knowledge of the graph signals, namely the data that lives on the graph nodes. We propose here a novel distributed graph learning algorithm, which permits to infer a graph from signal observations on the nodes under the assumption that the data is smooth on the target graph. We solve a distributed optimization problem with local projection constraints to infer a valid graph while limiting the number of messages to be transmitted. Our results show that the distributed approach involves less

messages than a centralised algorithm without compromising the accuracy in the inferred graph. It also scales better in number of messages with the increase of the network size, especially for sparse networks. Then, we extend the quantization analysis into the distributed graph learning algorithm. We propose an incremental bit allocation scheme between the three main message sets of the distributed algorithm, with a marginal analysis optimization in order to identify the best trade-off between bit costs and accuracy in the inferred graph. Experiments show that for both the training and the testing datasets, the proposed solution clearly presents a better trade-off than a baseline uniform bit allocation scheme.

Finally, in Chapter 6 we conclude the thesis and offer possible directions for future developments in this field.

1.3 Summary of Contributions

The main contributions of this thesis are summarized below. In particular, we propose:

- A modification of the classical graph filtering algorithm that permits to bound the range of the exchanged messages, by shifting the normalized Laplacian when implementing the filter. We modify the distributed filtering algorithm in order to recover the results that would be generated with the un-shifted Laplacian. The bounded range improves the quantization scheme.
- A novel method for distributed graph filtering that minimizes the error due to message quantization without compromising the communication costs. It adapts to the target task, the graph properties and the communication constraints.
- A study of the effects of quantization error in the accuracy of distributed inference with a graph convolution neural network and a solution for the resulting bit allocation problem with the Karush–Kuhn–Tucker (KKT) conditions.
- The first distributed graph learning framework that minimizes the number of exchanged messages. We solve a distributed optimization problem with local projection constraints to infer a valid graph while limiting the number of messages.
- An incremental bit allocation scheme for the distributed graph learning algorithm, with a marginal analysis technique. The proposed solution presents a better trade-off between accuracy and bit costs than a baseline uniform bit allocation scheme.

2 Preliminaries

In this chapter, we introduce some fundamentals in graph signal processing and the quantization rate-distortion model that we will be using in the following chapters of this thesis.

2.1 Graph Signal Processing

We consider a weighted undirected graph, which can be written as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$[1], representing our network. In this notation, $\mathcal{V}$ represents a set of $N = |\mathcal{V}|$ vertices, $\mathcal{E}$ represents a set of edges, and $\mathbf{W} \in \mathbb{R}^{N \times N}$ is the weight matrix whose entry W_{ij} represents the weight of the edge between nodes i and j. The value of W_{ij} is equal to 0 if these nodes are not connected. The degree matrix $\mathbf{D}$ contains diagonal entries equal to the sum of weights of all edges incident to the nodes on the graph, specifically

$$D_{ii} = d_i = \sum_j W_{ij}, \tag{2.1}$$

and has only zeros outside the diagonal.

We can also define a matrix whose sparsity pattern follows the structure of the network, namely the graph shift operator $\boldsymbol{S}$. It can represent the adjacency matrix, the Laplacian or their normalized and translated forms for example. In particular, the combinatorial Laplacian matrix is defined as

$$\mathcal{L}^c = \mathbf{D} - \mathbf{W}. \tag{2.2}$$

Additionally, the normalized graph Laplacian operator is frequently used in graph signal processing and is given by

$$\mathcal{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}. \tag{2.3}$$

Both Laplacians are real symmetric positive semi-definite matrices, which means they have a complete set of orthonormal eigenvectors with a set of non-negative eigenvalues. We then

[1] In this work, we represent scalars with non-bolded symbols, whereas we use bolded symbols for non-scalar arrays (such as vectors and matrices).

denote as $\{\lambda_n\}_{n=0..N-1}$ the set of eigenvalues of $\mathscr{L}$, which can be ordered as

$$0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq .. \leq \lambda_{N-1} \leq 2, \tag{2.4}$$

and the corresponding eigenvectors as $\{\mathscr{X}_0, \mathscr{X}_1, ..., \mathscr{X}_{N-1}\}$. Finally, we denote by $\mathbf{\Lambda}$ the diagonal matrix with the eigenvalues $\{\lambda_n\}_{n=0..N-1}$ on its diagonal and $\mathscr{X}$ the matrix whose columns are the eigenvectors of $\mathscr{L}$.

A graph signal is a function $\boldsymbol{f} : \mathcal{V} \longrightarrow \mathbb{R}$ defined on the vertices of the graph, which is represented by a vector $\mathbf{f} \in \mathbb{R}^N$. The graph signal $\boldsymbol{f}$ is defined on the vertex domain. The Laplacian eigenvalues carry a notion of frequency, so we can also represent this signal in the graph spectral domain. To that end, we make use of the graph Fourier transform [2], which can be defined as an expansion of the function in terms of the eigenvectors of $\mathscr{L}$. That is, the graph Fourier transform $\hat{\boldsymbol{f}}$ of a signal $\boldsymbol{f}$ at frequency λ_n can be defined as the sum

$$\hat{\boldsymbol{f}}(\lambda_n) = \sum_{i=1}^{N} \boldsymbol{f}(i)\mathscr{X}_n(i). \tag{2.5}$$

A very common assumption to be made about the graph signals is that they change smoothly among strongly connected vertices, that is, they take similar values at those vertices. In the graph spectral domain, this means that most of the signal's energy lie in the first eigenvalues of the underlying graph. We can measure the degree of smoothness with the Laplacian quadratic form

$$\mathscr{Q}(\mathscr{L}^c) = \text{tr}(\boldsymbol{X}^T \mathscr{L}^c \boldsymbol{X}) = \frac{1}{2}\sum_{i,j} \boldsymbol{W}_{ij} ||\boldsymbol{x}_i - \boldsymbol{x}_j||^2, \tag{2.6}$$

where $\boldsymbol{X} \in \mathbb{R}^{N \times M}$ is a matrix containing M graph signals as their columns. We call the N rows of $\boldsymbol{X}$ the feature vectors $\boldsymbol{x}_i \in \mathbb{R}^M$. We see that, when $\boldsymbol{W}_{ij}$ has a high value, nodes i and j are strongly connected. When a graph signal is smooth, by definition, the signal difference should be small for these nodes. Thus, we can see that we can minimize Eq. (2.6) with respect to $\boldsymbol{X}$ to obtain smooth signals, or alternatively, we can minimize it with respect to $\boldsymbol{W}$ to obtain graphs whose pre-existing signals can be smooth on.

2.2 Distributed Graph Signal Filtering

We further define filtering in graph signal processing. Given $\boldsymbol{H}(\lambda)$ the (traditional) Fourier transform of a graph filter, we can process the signal $\boldsymbol{f}_{in}$ by computing

$$\boldsymbol{f}_{out} = \boldsymbol{H}(\mathscr{L})\boldsymbol{f}_{in}, \tag{2.7}$$

with

$$\boldsymbol{H}(\mathscr{L}) := \mathscr{X} \begin{bmatrix} \boldsymbol{H}(\lambda_0) & & 0 \\ & & \\ 0 & & \boldsymbol{H}(\lambda_{N-1}) \end{bmatrix} \mathscr{X}^T. \tag{2.8}$$

We remind that $\mathscr{X}$ and λ are respectively, the eigenvectors and eigenvalues of the graph Laplacian. If the graph filter $\boldsymbol{H}(\mathscr{L})$ can be represented in polynomial format i.e.,

$$\boldsymbol{H}(\mathscr{L}) = \sum_{k=0}^{K} h_k \mathscr{L}^k, \tag{2.9}$$

with the filter coefficients $\boldsymbol{h} = [h_0, ..., h_K]^T$, it is denominated as a finite impulse response - FIR graph filter. It is also amenable to distributed implementation [3]. The corresponding graph convolution

$$\boldsymbol{f}_{out} = \sum_{k=0}^{K} h_k \mathscr{L}^k \boldsymbol{f}_{in} \tag{2.10}$$

can be implemented iteratively by local processing at each node, and message exchanges between nodes. That is, each node computes the local value of the function $\boldsymbol{f}_{out}$ by exchanging messages with neighbors in K interactions. Along this process, all nodes together participate in computing the full function $\boldsymbol{f}_{out}$.

The distributed implementation requires the computation of the different powers of the Laplacian matrix that appear in (2.10). We firstly define

$$\boldsymbol{z}_k = \mathscr{L}^k \boldsymbol{f}_{in} \tag{2.11}$$

and begin with

$$\boldsymbol{z}_0 = \boldsymbol{f}_{in}, \tag{2.12}$$

as illustrated in Fig. (2.1).

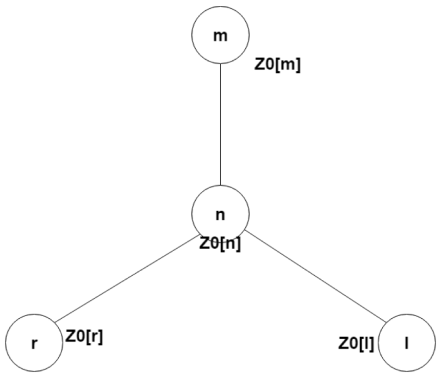


Figure 2.1 – Toy network with initial $\boldsymbol{z}_0$ local values.

The node n sends its value $z_0[n]$ to its one-hop neighbors in the graph, and all the other nodes do the same, as illustrated in Fig. (2.2).

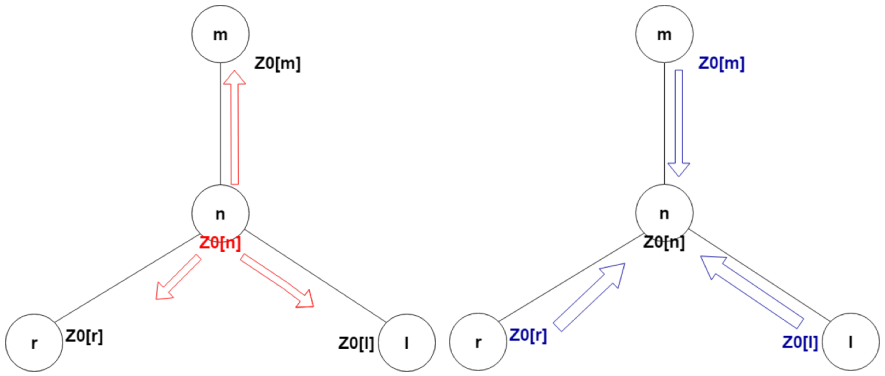


Figure 2.2 – Nodes share their z_0 local values with immediate neighbours.

After all values of $\boldsymbol{z}_0$ are exchanged in the network, all nodes update their local status (Fig. (2.3)) with the relation

$$\boldsymbol{z}_1 = \mathscr{L}\boldsymbol{z}_0. \tag{2.13}$$

To that end, each node n only calculates its value $z_1[n]$ by doing $\mathscr{L}_n^T \boldsymbol{z}_0$, where $\mathscr{L}_n$ is the row n of $\mathscr{L}$; $\boldsymbol{z}_0$ is filled with the values of the messages exchanged from the neighbors of node n. Since $\mathscr{L}_n$ is zero for the nodes that are not neighbors of n, the node n does not need the values of $\boldsymbol{z}_0$ at these nodes to calculate $z_1[n]$.

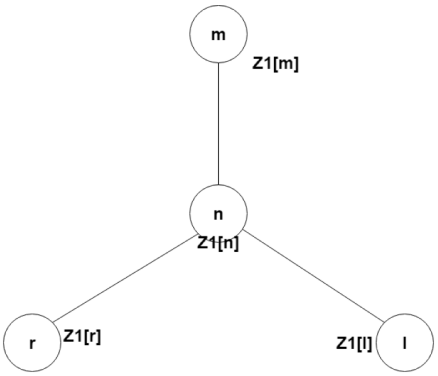


Figure 2.3 – All nodes update their local status, producing their local $\boldsymbol{z}_1$ values.

The messages with the values of $\boldsymbol{z}_1$ are then exchanged between neighbor nodes in the same way as for the values of z_0. Then $\boldsymbol{z}_2 = \mathscr{L}\boldsymbol{z}_1$ is obtained in a similar fashion as $\boldsymbol{z}_1$. This procedure repeats until K iterations, which permits to compute the full function $\boldsymbol{f}_{out}$ in Eq. (2.10). Specifically, after knowing $\{z_0[n], z_1[n], ..z_K[n]\}$, the node n computes the value of the filtered signal in its own node using the relation [39]

$$\boldsymbol{f}_{out}[n] = \left(\boldsymbol{H}(\mathscr{L})\boldsymbol{f}_{in}\right)[n] = \sum_{k=0}^{K} h_k z_k[n]. \tag{2.14}$$

More details are given in [3]. Distributed graph filtering is possible with nodes computing only local values, while exchanging messages with immediate neighbours. At every instance of the algorithm, the knowledge of the entire input or output signals is required nowhere in the network.

2.3 Rate-distortion model

We now introduce the quantization rate-distortion model used in this thesis. In distributed settings, messages are transmitted in order to attain the nodes' joint objective. In realistic cases, these messages exchanged by the network nodes have a limited precision, being typically quantized before transmission. We consider the high rate quantization regime and use uniform quantizers for each information message, as in high-rate regime uniform quantization is optimal for independent variables. In such setting, the error is directly related to the number of bits b used for each message which can be chosen to optimize performance. In uniform quantization, the step size Δ is determined by the ratio of the total quantization range Γ over the number of quantization intervals, that is

$$\Delta = \frac{\Gamma}{2^b}. \tag{2.15}$$

The quantization range is the value range that the message is being quantized on. Its maximum value is typically higher than the maximum possible value of the message, and similarly, the minimum value is typically lower than its minimum value. Sometimes it can be determined theoretically, from the input signal properties, or experimentally, where typical values are used.

It had been shown [40] that even though the quantization noise and input signal are deterministically related, when the quantization step size is sufficiently small (that is, in high rate conditions), the quantization noise is uncorrelated with the quantized signal. Furthermore, the probability density function of the quantization noise will be uniform with zero mean and variance $\frac{\Delta}{12}$. This is called the White-Noise Quantization Error Model [41].

It follows that the expected value of the square of the quantization error ϵ (MSE) on the message transmitted is equal to its variance (since it has zero mean). If we apply (2.15) into the variance expression of the White-Noise Quantization Error Model, we obtain for the MSE

$$E[\epsilon^2] = \frac{\Gamma^2}{12} \cdot 2^{-2b}. \tag{2.16}$$

With the White-Noise Quantization Error Model, we can additionally assume that the different messages are statistically independent, more details in [41]. This assumption will be used in the following chapters of this thesis.

3 Optimized Quantization in Distributed Graph Signal Filtering

3.1 Introduction

Many different networks, for instance wireless sensor networks, transportation networks, neural networks and social networks can be modeled as graphs where nodes support signal values or features. The field of signal processing on graphs has been providing many powerful tools to process such signals in diverse applications, such as compression, denoising or reconstruction of sensor data [1, 2].[1]

Numerous applications require that the signal defined on the network is however processed distributively. Decentralized methods of graph signal processing recently emerged in order to scale to large networks as a way to deal with big data applications, privacy concerns and also bandwidth/energy constraints [3–5]. They can also be necessary when centralized topologies are not viable or suffer from a bottleneck on the central processor, with nodes that are too far to reach the central processor. Another advantage of distributed algorithms compared to centralized ones is that they can add robustness to the network in case of node failures.

In practice, most nodes in distributed systems have limited computing power and are constrained in the amount of information they can communicate; this makes the design of efficient distributed algorithms essential. In order to enable distributed graph signal processing, linear graph signals operators can for example be approximated by shifted Chebyshev polynomials [3], becoming more amenable to distributive computing for applications such as smoothing, denoising, inverse filtering and semi-supervised learning. There have been many other studies on distributed processing for graph signals or networked data [4–8], but only few deal with the fact that, in real case scenarios, the network is subject to communication constraints that limit the precision of the messages exchanged by distributed algorithms.

In this chapter, we propose an adaptive quantization scheme for distributed graph signal

[1] This chapter contains work which has been published in:
Nobre, I.C.M. and Frossard, P., 2019, May. Optimized quantization in distributed graph signal processing. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 5376-5380).

processing tasks. Quantization approximates a continuous range of values by a set of discrete values and helps decreasing the communication costs. At the same time, it introduces errors that degrade the performance of distributed processing methods. We focus on the design of a quantization scheme that minimizes the quantization error in graph signal processing tasks, by bounding the transmitted messages and by optimizing the bit allocation. We first propose a distributed processing algorithm where the messages exchanged by network nodes are bounded. We then model the error due to quantization, as a function that depends on the network topology and on the characteristics of the distributed processing task. We further cast an optimization problem to efficiently distribute bit to messages and nodes, so that the total error is minimized under communication cost constraints. We finally solve the resulting bit allocation problem with the Karush-Kuhn-Tucker (KKT) conditions. The performance of our quantizer is evaluated and compared to the performance of an uniform quantizer. The results show that the bit allocation optimization improves the performance in terms of final error compared to a uniform distribution of the bit budget across messages. They also show that a more regular graph leads towards more uniform bit distribution in the optimal allocation, which confirms the proper adaptation of our solution to the network properties. Also, since the errors propagate through the successive steps of the distributed signal processing operations, we also confirm that it is efficient to allocate a large share of the bit budget in the first steps of the iterative distributed processing algorithm.

Some works have considered different aspects of quantization in graph signal processing. The works in [42] and [43] briefly studied the effect of quantization in a linear prediction filter applied to graph signals. However the main objective of those works was to design graph filters, without considering specifically the quantization effects in the design itself. On the other hand, the work in [44] studied the effect of quantization in the representation of graph signals, mitigating the numerical effects caused by the finite-precision machines that centrally realize the filtering process. The authors specifically designed graph filters that are robust to finite precision effects. That is, [44] focused on mitigating quantization effects by designing filter coefficients, not by optimising the bit allocation on different nodes, like we do. The above works had a centralized graph filtering approach, which means no quantization in communication between nodes. In our case, the whole processing algorithm is done in the network itself. There are also some works that focused on distributed optimization with quantization constraints [45–49], but these algorithms have been developed mainly using consensus protocols, and we are interested in solving more general processing tasks in this chapter, not merely average computation. Despite some similarities between graph filtering and consensus, there are some differences. Consensus can be considered a special case of low pass filtering, and many consensus works are not focused in building optimal quantizers, but in the convergence to consensus between nodes in the presence of quantized messages. While in graph filters the design of an optimal quantizer does not require that nodes reach consensus. More recently, [50] provided a broad analysis of the quantization effects of graph filters over time-invariant and time-varying graphs. Also, [51] designed graph filters that are jointly robust to quantized data and time-varying graph topologies. Closer to our framework, the work [39]

derived the quantization error in distributed computations of graph signal operators and then proposed an algorithm that learns graph dictionaries to sparsely approximate graph signals while staying robust to the quantization noise. The work however does not focus on improving the quantization but rather on the design of robust graph filters. To the best of our knowledge, our work the first one that optimizes the quantization scheme for general distributed filtering tasks.

The rest of this chapter is organized as follows. In Section 3.2, we model the graph filter implementation in a distributed way and derive the quantization error. In Section 3.3, we present our new bounding scheme for processing graph signals distributively, and in Section 3.4 we describe our optimal bit allocation algorithm. Finally, the results and analysis of the performance of our new framework are shown in Section 3.5.

3.2 Quantization Error for Distributed Graph Filtering

As shown in Section 2.2, one can perfectly compute the response of graph filters in distributed settings. However, in realistic cases, the messages exchanged by the network nodes have a limited precision. They are typically quantized before transmission and this modifies the outcome of the distributed filtering process.

The quantized message at node n at step k can be written as $\tilde{z}_k[n] = \boldsymbol{z}_k[n] + \boldsymbol{\epsilon}_k[n]$, where $\boldsymbol{\epsilon}_k[n]$ is the quantization error for a message at iteration step k at node n. After its transmission, the distributed update equation at node n becomes $\boldsymbol{z}_{k+1} = \mathscr{L}\tilde{z}_k[n] = \mathscr{L}(\boldsymbol{z}_k[n] + \boldsymbol{\epsilon}_k[n])$ and integrates a quantization error term.

We define by

$$\boldsymbol{\epsilon}_k = \begin{bmatrix} \boldsymbol{\epsilon}_k[0] \\ \vdots \\ \boldsymbol{\epsilon}_k[N] \end{bmatrix} \tag{3.1}$$

the vector containing error values for all nodes at step k, and $\boldsymbol{\epsilon} = [\boldsymbol{\epsilon}_0^T, ..., \boldsymbol{\epsilon}_{K-1}^T]^T$ the vector containing all errors at all iteration steps and nodes.

By taking into account the quantization errors that accumulate through all iterations of the distributed processing task, the filtered signal can finally be written as

$$\boldsymbol{H}(\mathscr{L})\boldsymbol{f}_{in} = \sum_{k=0}^{K} h_k \mathscr{L}^k \boldsymbol{z}_0 + \sum_{k=0}^{K-1} \left[\sum_{j=1}^{K-k} h_{k+j} \mathscr{L}^j \right] \boldsymbol{\epsilon}_k, \tag{3.2}$$

as opposed to $\boldsymbol{f}_{out}$ in Eq. (2.10) for the perfect settings. More details are available in [39].

3.3 Quantized Distributed Filtering with bounded messages

3.3.1 Quantization error analysis

We first analyze more deeply the impact of quantization. Since the first term of the above expression (3.2), $\sum_{k=0}^{K} h_k \mathscr{L}^k \boldsymbol{z}_0$, is the filtered signal in a setting without quantization (as in Eq (2.14)), we can define the second term,

$$\boldsymbol{Q} = \sum_{k=0}^{K-1} \left[\sum_{j=1}^{K-k} h_{k+j} \mathscr{L}^j \right] \boldsymbol{\epsilon}_k, \tag{3.3}$$

as the total error caused by the accumulated effects of the quantization errors.

We can further make the following observation on the evolution of the quantization error with the iterations of the distributed processing algorithm. At step k, the maximum value (in an absolute sense) of the messages to be transmitted is $||\mathscr{L}^k f_{in}||_\infty$. Considering that, for the norm indexes $p > r > 0$, we have $\| \boldsymbol{v} \|_r \geq \| \boldsymbol{v} \|_p$, for any $\boldsymbol{v}$ pertaining to the vector space where these norms are defined, we can write

$$\| \mathscr{L}^k \boldsymbol{f}_{in} \|_\infty \leq \| \mathscr{L}^k \boldsymbol{f}_{in} \|_2 \leq \| \mathscr{L}^k \|_2 \cdot \| \boldsymbol{f}_{in} \|_2, \tag{3.4}$$

where $\| \mathscr{L}^k \|_2$ is a matrix norm induced from the 2-norm for vectors, which can be computed by $\| \mathscr{L}^k \|_2 = \sigma_{max}(\mathscr{L}^k)$, where $\sigma_{max}(\mathscr{L}^k)$ represents the largest singular value of matrix $\mathscr{L}^k$ [52]. It corresponds to the square root of the largest eigenvalue of the positive-semidefinite matrix $(\mathscr{L}^k)^T(\mathscr{L}^k)$. Since the Laplacian matrix is diagonalizable and symmetric, we can write $(\mathscr{L}^k)^T = (\mathscr{L}^k)$, and since

$$\mathscr{L} = \mathscr{X} \Lambda \mathscr{X}^T, \tag{3.5}$$

we have

$$\mathscr{L}^{2k} = \mathscr{X} \Lambda^{2k} \mathscr{X}^T, \tag{3.6}$$

which means that the eigenvalues of $\mathscr{L}^{2k}$ are the same as the eigenvalues of $\mathscr{L}$ to a power of $2k$. Hence, since

$$\| \mathscr{L}^k \|_2 = \sigma_{max}(\mathscr{L}^k) = \sqrt{\lambda_{max}(\mathscr{L}^{2k})} = \sqrt{\lambda_{N-1}^{2k}}, \tag{3.7}$$

and considering that all eigenvalues of the Laplacian are real and positive values, we finally have

$$\| \mathscr{L}^k \boldsymbol{f}_{in} \|_\infty \leq \lambda_{N-1}^k \cdot \| \boldsymbol{f}_{in} \|_2 . \tag{3.8}$$

This means that, as k increases, the transmitted messages can increase their ranges proportionally to the eigenvalues of $\mathscr{L}$, as shown in Eq. (3.8). This also means that, at a high value of k, the value of the respective error $\boldsymbol{\epsilon}_k$ will be very high, hence increasing the value of the total error. We show below how to bound these transmitted messages such that the quantization errors will remain bounded as well.

3.3.2 Distributed processing with bounded messages

Based on the previous observations, we propose a modification of the classical distributed processing algorithm that permits to bound the range of the exchanged messages. Instead of using the operator given by the normalized Laplacian $\mathscr{L}$ at every step of the distributed algorithm[2], we use

$$\dot{\mathscr{L}} = \mathscr{L} - \boldsymbol{I}. \tag{3.9}$$

Hence, the eigenvalues of $\dot{\mathscr{L}}$ will be bounded in $[-1,1]$, instead of $[0,2]$, with $\mathscr{L}$. Therefore the values of the messages being transmitted at step k will surely not surpass the range of the original signal $\boldsymbol{z}_0 = \boldsymbol{f}_{in}$, as shown in Eq. (3.8).

In order to integrate the modified Laplacian operator $\dot{\mathscr{L}}$, we modify the distributed filtering algorithm of the previous section as follows. We start with a scenario without quantization. Firstly we set $\boldsymbol{z}_0 = \boldsymbol{f}_{in}$, which is then exchanged with the neighbor nodes, as before. Now, instead of multiplying the received values by $\mathscr{L}$, the nodes rather compute $\dot{\boldsymbol{z}}_1 = \dot{\mathscr{L}}\boldsymbol{z}_0$. The value $\dot{\boldsymbol{z}}_1$ is then exchanged with neighbor nodes, in a similar way as the algorithm described in the previous section. The algorithm then proceeds iteratively in the same way.

In an ideal scenario (without quantization), we have $\dot{\boldsymbol{z}}_k = \dot{\mathscr{L}}^k\boldsymbol{z}_0 = (\mathscr{L} - \boldsymbol{I})^k\boldsymbol{z}_0$. We observe that, at each step, we can perfectly recover $\boldsymbol{z}_k = \mathscr{L}^k\boldsymbol{z}_0$ from $\dot{\boldsymbol{z}}_k$. Since the identity matrix commutes with all matrices, $\mathscr{L}$ and $\boldsymbol{I}$ also commute. Hence, we can use the Binomial formula and derive

$$\dot{\boldsymbol{z}}_k = \left(\sum_{i=0}^{k}\binom{k}{i}(-1)^{k-i}\mathscr{L}^i\right)\boldsymbol{z}_0 = \sum_{i=0}^{k}\binom{k}{i}(-1)^{k-i}\boldsymbol{z}_i, \tag{3.10}$$

or equivalently,

$$\boldsymbol{z}_k = \dot{\boldsymbol{z}}_k - \sum_{i=0}^{k-1}\binom{k}{i}(-1)^{k-i}\boldsymbol{z}_i = \sum_{i=0}^{k}\binom{k}{i}\dot{\boldsymbol{z}}_i. \tag{3.11}$$

Therefore, with Eq. (3.11), we can build a distributed algorithm where the values of $\dot{\boldsymbol{z}}_k$ are exchanged between neighbor nodes, but only the values $\boldsymbol{z}_k$ corresponding to the iterative solutions of the original distributed filtering algorithm need to be stored. This is useful if the sensors have memory constraints. Note that the term $\sum_{i=0}^{k}\binom{k}{i}\dot{\boldsymbol{z}}_i$ is purely combinatorial, that is, it does not depend on any specific component such as the network, data, iteration or task, etc.

[2] We note that the signal is still being processed with $\mathscr{L}$, and the operator $\dot{\mathscr{L}}$ is only used to, at transmission stage, temporarily transform transmitted messages in order to bound the quantization range and consequently reduce the quantization error. Once transmitted messages are received, the equivalent values that would have been generated by processing the signal with $\mathscr{L}$ are recovered (at every step). Thus, this process does not change graph filtering properties intrinsic from processing with $\mathscr{L}$.

3.3.3 Quantization error with bounded messages

In a scenario with quantization, the distributed filtering algorithm with the modified Laplacian operator $\dot{\mathscr{L}}$ is modified as follows. After we set $\boldsymbol{z}_0 = \boldsymbol{f}_{in}$, we quantize it as $\tilde{\boldsymbol{z}}_0 = \boldsymbol{z}_0 + \boldsymbol{\epsilon}_0$. The values $\tilde{\boldsymbol{z}}_0$ are then exchanged with the neighbor nodes, as before. Now, instead of multiplying the received values by $\mathscr{L}$ as in the original algorithm, the nodes rather compute $\dot{\boldsymbol{z}}_1 = \dot{\mathscr{L}}\tilde{\boldsymbol{z}}_0$ in the bounded scheme, the resulting value is then quantized as $\tilde{\dot{\boldsymbol{z}}}_1 = \dot{\boldsymbol{z}}_1 + \boldsymbol{\epsilon}_1$. The quantized value $\tilde{\dot{\boldsymbol{z}}}_1$ is eventually exchanged with neighbor nodes, in a similar way as the algorithm described above.

To recover $\boldsymbol{z}_k$ from $\dot{\boldsymbol{z}}_k$ we use the same process as in Eq. (3.11). However, the recovery is not perfect anymore due to quantization. The quantization error now accumulates through iterations and the value of $\dot{\boldsymbol{z}}_k$ becomes

$$\dot{\boldsymbol{z}}_k = (\mathscr{L} - \boldsymbol{I})^k \boldsymbol{z}_0 + \sum_{l=0}^{k-1} (\mathscr{L} - \boldsymbol{I})^{k-l} \boldsymbol{\epsilon}_l, \qquad \text{for } k > 0, \tag{3.12}$$

while it is given by Eq. (3.10) in the ideal settings. More specifically, we compute below the quantization error in receiving the output of the filtering process. First, Eq. (3.11) is equivalent to

$$\boldsymbol{z}_k = \boldsymbol{z}_0 + \sum_{i=1}^{k} \binom{k}{i} \dot{\boldsymbol{z}}_i. \tag{3.13}$$

If we replace $\dot{z}_i$ in (3.13) with the expression in (3.12), we obtain

$$\boldsymbol{z}_k = \boldsymbol{z}_0 + \sum_{i=1}^{k} \binom{k}{i} \left[(\mathscr{L} - \boldsymbol{I})^i \boldsymbol{z}_0 + \sum_{l=0}^{i-1} (\mathscr{L} - \boldsymbol{I})^{i-l} \boldsymbol{\epsilon}_l \right]. \tag{3.14}$$

Notice that, if we write $\mathscr{L}^k$ as $[(\mathscr{L} - \boldsymbol{I}) + \boldsymbol{I}]^k$ and use the Binomial formula, we obtain

$$\mathscr{L}^k = \sum_{i=0}^{k} \binom{k}{i} (\mathscr{L} - \boldsymbol{I})^i. \tag{3.15}$$

Thus, (3.14) can be written as

$$\boldsymbol{z}_k = \mathscr{L}^k \boldsymbol{z}_0 + \sum_{i=1}^{k} \binom{k}{i} \sum_{l=0}^{i-1} (\mathscr{L} - \boldsymbol{I})^{i-l} \boldsymbol{\epsilon}_l, \qquad \text{for } k > 0. \tag{3.16}$$

The distributed filtering of the graph signal f_{in} in the quantization regime can then be written as

$$\dot{f}_{out} = \boldsymbol{H}(\mathscr{L}) \boldsymbol{f}_{in} = \sum_{k=0}^{K} h_k \mathscr{L}^k \boldsymbol{z}_0 + \sum_{k=1}^{K} h_k \sum_{i=1}^{k} \binom{k}{i} \sum_{l=0}^{i-1} (\mathscr{L} - \boldsymbol{I})^{i-l} \boldsymbol{\epsilon}_l, \tag{3.17}$$

from which we derive the total error, which corresponds to the second term in Eq. (3.17),

$$\dot{\boldsymbol{Q}} = \sum_{k=1}^{K} h_k \sum_{i=1}^{k} \binom{k}{i} \sum_{l=0}^{i-1} (\mathscr{L} - \boldsymbol{I})^{i-l} \boldsymbol{\epsilon}_l. \tag{3.18}$$

It can be rewritten as

$$\dot{\boldsymbol{Q}} = \sum_{k=0}^{K-1} \left[\sum_{i=1}^{K-k} h_{k+1} \sum_{j=1}^{i} \binom{k+i}{k+j} (\mathscr{L} - \boldsymbol{I})^j \right] \boldsymbol{\epsilon}_k. \tag{3.19}$$

For the sake of clarity, we now write

$$F_k[n] = \left(\boldsymbol{H}_k^T \boldsymbol{H}_k\right)[n,n], \tag{3.20}$$

with

$$\boldsymbol{H}_k = \sum_{i=1}^{K-k} h_{k+1} \sum_{j=1}^{i} \binom{k+i}{k+j} (\mathscr{L} - \boldsymbol{I})^j. \tag{3.21}$$

If we combine (3.21) and (3.20), we get

$$F_k[n] = \sum_{i=1}^{K-k} h_{k+i} \sum_{j=1}^{i} \binom{k+i}{k+j} \sum_{p=1}^{K-k} h_{k+p} \sum_{q=1}^{p} \binom{k+p}{k+q} \cdot (\mathscr{L} - \boldsymbol{I})^{j+q}[n,n]. \tag{3.22}$$

Hence,

$$\dot{\boldsymbol{Q}} = \sum_{k=0}^{K-1} H_k \boldsymbol{\epsilon}_k, \tag{3.23}$$

and we can finally calculate the mean squared error

$$\| \dot{\boldsymbol{Q}} \|^2 = \sum_{k=0}^{K-1} \sum_{l=0}^{K-1} \boldsymbol{\epsilon}_k^T \boldsymbol{H}_k^T \boldsymbol{H}_l \boldsymbol{\epsilon}_l, \tag{3.24}$$

and its expected value

$$E_{\boldsymbol{\epsilon}} \left[\| \dot{\boldsymbol{Q}} \|^2 \right] = \sum_{k=0}^{K-1} \sum_{l=0}^{K-1} E_{\boldsymbol{\epsilon}} [\boldsymbol{\epsilon}_k^T \boldsymbol{H}_k^T \boldsymbol{H}_l \boldsymbol{\epsilon}_l]. \tag{3.25}$$

3.4 Optimized bit allocation

3.4.1 Rate-distortion model

In this section, we now seek to minimize the expected value of the square of the total error in Eq. (3.24) so that the impact of communication constraints on the distributed computations is minimum.

We define $b[n,k]$ as the number of bits used to represent the message sent from node n at step k. We consider the high rate regime and use uniform quantizers for each message. In this case, the error is directly related to the number of bits used for each message. The quantization step size is then determined by the ratio of the total quantization range over the number of

quantization intervals, that is

$$\Delta[n,k] = \frac{2 \cdot \| \boldsymbol{f}_{in} \|_\infty}{2^{b[n,k]}}, \tag{3.26}$$

the quantization range is $2 \cdot \| \boldsymbol{f}_{in} \|_\infty$ for all k since we filter the signal with the modified Laplacian $\dot{\mathscr{L}}$ to bound the messages. The 2-factor is used because we opt for a symmetric quantization around 0, that is, the quantization range lowest value is $- \| \boldsymbol{f}_{in} \|_\infty$ and maximum value is $+ \| \boldsymbol{f}_{in} \|_\infty$.

As seen in Chapter 2, if we apply (3.26) into the variance expression of the White-Noise Quantization Error Model, we obtain

$$E_{\boldsymbol{\epsilon}}[\boldsymbol{\epsilon}_k[n]^2] = \frac{\| \boldsymbol{f}_{in} \|_\infty^2}{3} \cdot 2^{-2b[n,k]}. \tag{3.27}$$

With the White-Noise Quantization Error Model, we can assume that $\boldsymbol{\epsilon}_k[n]$ and $\boldsymbol{\epsilon}_p[m]$ are statistically independent for $k \neq p$ or $n \neq m$. Hence the expected value of the crossed terms in Eq. (3.25) is zero and we finally obtain the expected value of the total mean squared error as

$$E_{\boldsymbol{\epsilon}}[\| \dot{\boldsymbol{Q}} \|^2] = \sum_{k=0}^{K-1} \sum_{n=0}^{N-1} \left(\boldsymbol{H}_k^T \boldsymbol{H}_k \right)[n,n] E_{\boldsymbol{\epsilon}}[\boldsymbol{\epsilon}_k[n]^2], \tag{3.28}$$

which is equivalent to

$$E_{\boldsymbol{\epsilon}}[\| \dot{\boldsymbol{Q}} \|^2] = \sum_{k=0}^{K-1} \sum_{n=0}^{N-1} F_k[n] E_{\boldsymbol{\epsilon}}[\boldsymbol{\epsilon}_k[n]^2], \tag{3.29}$$

with (3.27), the MSE finally reads

$$E_{\boldsymbol{\epsilon}}[\| \dot{\boldsymbol{Q}} \|^2] = \frac{\| \boldsymbol{f}_{in} \|_\infty^2}{3} \sum_{k=0}^{K-1} \sum_{n=0}^{N-1} F_k[n] 2^{-2b[n,k]}. \tag{3.30}$$

We can see from (3.29) that the quantization error is higher when $F_k[n]$ increases. An interesting problem would be to minimize $F_k[n]$ to lower this error. This is outside the scope of this work, but we can indicate ways this could be achieved. First of all, from (3.22) we see that the number of steps K needed to distributively process the signal directly impacts the amount of terms in the sum to compute $F_k[n]$. Not only that, it directly impacts the amount of error terms that compose the global error in (3.29). Thus, if the filter can be designed flexibly, it would beneficial to perform lower-order polynomial approximations for it. Alternatively, a low-order polynomial filter could be directly designed. Secondly, the filter coefficients h_k values could also be designed in such a way that (3.22) is minimized. A third way of optimizing $F_k[n]$ is to choose denser topologies, since the redundant information shared among nodes would mitigate the global quantization error. This last strategy however is not as good as the first two, since it would also directly impact the communication costs.

3.4.2 Optimal Allocation

Our objective is now to minimize the total quantization error given the constraint bit budget in the network. To that end, it is necessary to find the values of $b[n,k]$ for every combination of k and n that obey the budget constraint and minimize $E_{\boldsymbol{\epsilon}}[\| \dot{\boldsymbol{Q}} \|^2]$ of Eq. (3.30). This optimization can be described by the following problem:

$$\boxed{\begin{aligned} &\underset{x}{\text{minimize}} && E_{\boldsymbol{\epsilon}}[\| \dot{\boldsymbol{Q}} \|^2] \\ &\text{subject to} && \sum_{k=0}^{K-1} \sum_{n=0}^{N-1} b[n,k]\, d[n] \leq B \end{aligned}} \tag{3.31}$$

Here, $d[n]$ is the degree of node n, which drives the transmission costs, and B is the total bit budget constraint over the whole network. The term $b[n,k]d[n]$ represents the total number of bits used by node n at step k to send the respective message for all $d[n]$ neighbors. When we sum the $b[n,k]d[n]$ term for every k and n, we obtain the total number of bits used to process the signal, which is our constraint in the optimization problem. Since the objective and the constraint functions are both continuously differentiable and convex on the values of $b[n,k]$, we can use the Karush-Kuhn-Tucker (KKT) conditions and they will be sufficient for finding the optimal solution. Also, we observe that the problem satisfies the KKT regularity conditions since the constraint function is affine [53].

The KKT stationarity condition states that the solution of the optimization problem (3.31) will be the values of $b[n,k]$ that satisfy the equation

$$\frac{\partial}{\partial b[n,k]} \left[(E_{\boldsymbol{\epsilon}}[\| \dot{\boldsymbol{Q}} \|^2]) + \mu \Big(\sum_{k=0}^{K-1} \sum_{n=0}^{N-1} b[n,k]\, d[n] - B \Big) \right] = 0, \tag{3.32}$$

where $\frac{\partial}{\partial b[n,k]}$ means the partial derivative with respect to $b[n,k]$, and μ is the KKT multiplier. The solution to Eq. (3.32) is given by

$$b[n,k] = -\frac{1}{2\ln(2)} \ln\left(\frac{3\,\mu\, d[n]}{2 \,\| \boldsymbol{f}_{in} \|_\infty^2 \ln(2) F_k[n]} \right). \tag{3.33}$$

The complementary slackness condition states that

$$\mu \Big(\sum_{k=0}^{K-1} \sum_{n=0}^{N-1} b[n,k]\, d[n] - B \Big) = 0, \tag{3.34}$$

but if $\mu = 0$, the expression in (3.33) would present a $\ln(0)$ term, which is undefined. Then, (3.34) and (3.32) would no longer be optimality conditions for (3.31). This leads to the second term of Eq. (3.34) being equal to zero instead, and applying (3.33) into (3.34) and solving for μ,

it results into

$$\mu = \exp\left(- \frac{B\ln(4) + \sum_{n=0}^{N-1} d[n] \sum_{k=0}^{K-1} \ln\left(\frac{3d[n]}{2\|\boldsymbol{f}_{in}\|_{\infty}^{2} \ln 2 \cdot F_k[n]} \right)}{K \sum_{n=0}^{N-1} d[n]} \right), \tag{3.35}$$

which is always non-negative for being an exponential function, thus guaranteeing that the KKT dual feasibility condition is satisfied. And we can finally replace in (3.33) to obtain the optimized number of bits for each message.

In practice, a conventional approach [54] is to further round the non-integer values in Eq. (3.33) to become integers. Then, if some of the values of $b[n,k]$ obtained in (3.33) are negative or zero, they are replaced by 1, to guarantee that there is a minimal communication between neighboring nodes in every step to keep the whole system synchronized.

We can analyze the optimal bit allocation solution as follows. We observe that the number of bits $b[n,k]$ in Eq. (3.33) depends on $d[n]$ and $F_k[n]$. As the degree $d[n]$ grows, the communication cost in node n grows as well, since it shares its messages with more neighbors. In order to satisfy the budget constraint, $b[n,k]$ has thus to be smaller in a node where $d[n]$ is larger. On the other hand, the factor $F_k[n]$ is related to the topology and to the filter coefficients. From Eq. (3.29), we can see that it weights the contribution of each individual error $\boldsymbol{\epsilon}_k[n]$ in the global error. If we have a big $F_k[n]$, it means that the relative contribution of $\boldsymbol{\epsilon}_k[n]$ becomes big, so that its contribution needs to be reduced by increasing $b[n,k]$. Also, for the same node n, there are more error terms in the global error computation (in Eq. (3.22)) when k is low, which means that $F_k[n]$ becomes higher in this case. It further means that the relative contribution of the error terms $\boldsymbol{\epsilon}_k$ in the first iterations (small k) of the distributed processing algorithm is higher compared to the error in the later iterations. This is in agreement with the fact that the first error terms lead to higher propagation effects. These observations will be further confirmed in the upcoming result section.

3.5 Results and Discussion

3.5.1 Performance of the proposed scheme

a) Performance of the optimal quantizer

The performance of our quantizer is now evaluated in details. First, we create a network with $N = 50$ nodes that are uniformly placed at random in a unit square. The weight matrix for the network edges is generated based on a thresholded Gaussian kernel function that takes into account the physical distance between nodes. The edges weights are given by $\boldsymbol{W}_{ij} = e^{-l_{ij}^2/\theta}$ if the distance l_{ij} between vertices i and j is less or equal to κ, and zero otherwise. We fix $\kappa = 0.2$.

A graph signal is defined as $\boldsymbol{f}_{in}[n] = a[n]^2 + b[n]^2 - 1$, where $a[n]$ and $b[n]$ are the coordinates

of node n, and a random noise with zero-mean normal distribution is added to it. We consider denoising as the distributed graph signal processing task, via a low-pass filter

$$H(\lambda) = \frac{\tau}{\tau + 5\lambda} \tag{3.36}$$

with $\tau = 3$. In order to implement it distributively, a Chebyshev polynomial approximation of order K is performed, and its polynomial coefficients $\{h_k\}_{k=0..K}$ are determined as in [3].

The distributed graph signal processing task is first performed without quantization. Then, the processing is performed with uniform quantization that is used as baseline solution. In this case, the number of bits used to represent the transmitted messages is the same for every node n and iteration step k. Finally, another processing is performed, using the optimization scheme described in this chapter. In all three cases, the bounding scheme from Subsection 3.3.2 is used. We calculate the MSE between the output of the quantized and the unquantized schemes for both uniform and optimized quantization. We repeat the entire experiment 1000 times for different networks and compute average performance shown in Figure 3.1.

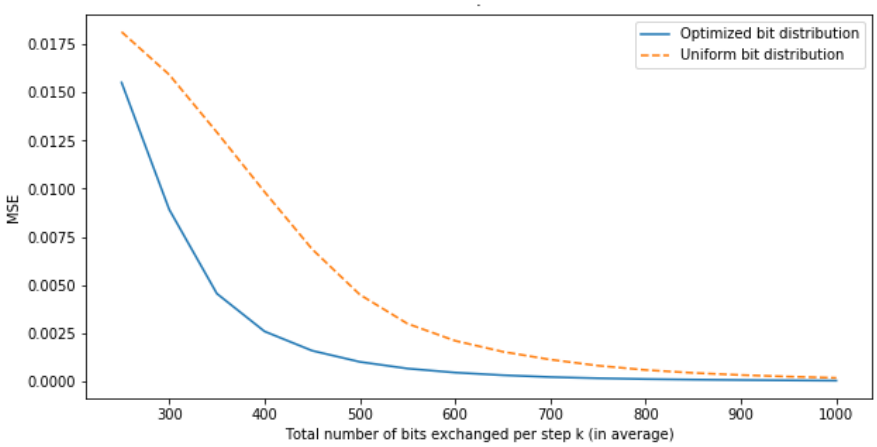


Figure 3.1 – Average MSE vs average number of bits for uniform, and optimized bit allocation for $K = 9$ and $\theta = 2$ ($\kappa = 0.2$, $\boldsymbol{f}_{in}[n] = a[n]^2 + b[n]^2 - 1$, $H(\lambda) = \frac{3}{3+5\lambda}$).

It can be seen from Fig. 3.1 that the optimization of the bit allocation proposed in this chapter clearly improves the performance in terms of MSE if compared to a quantizer with the uniform bit distribution. A bigger gain appears at low bit rate where effective allocation is more important as resource are scarcer.

To evaluate the efficiency of the bounding scheme, we process the signal using the distributed algorithm proposed in [39], where the messages are not bounded. The settings are the same as the previous experiment. We obtain a MSE of 0.77 for 300 exchanged bits, a MSE of 0.55 for 400 exchanged bits and a MSE of 0.26 for 600 exchanged bits. These results show that, regardless of optimizing the bit distribution or not, our bounded range algorithm yields much lower MSE values compared to the baseline algorithm for the same bit rates. This is due to the fact that,

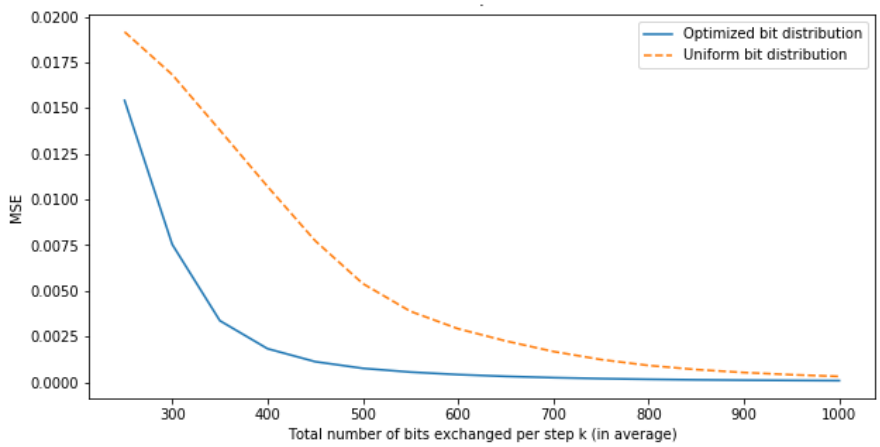


Figure 3.2 – Average MSE vs average number of bits for uniform and optimized bit allocation for $K = 9$ and $\theta = 0.001$ ($\kappa = 0.2$, $\boldsymbol{f}_{in}[n] = a[n]^2 + b[n]^2 - 1$, $H(\lambda) = \frac{3}{3+5\lambda}$).

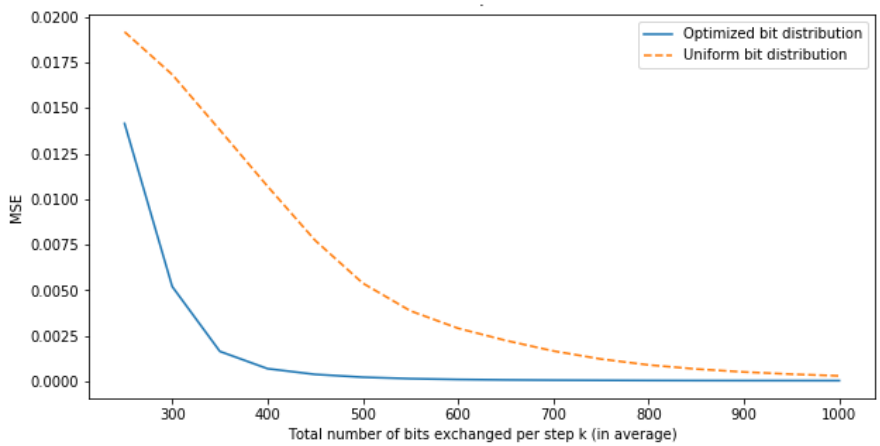


Figure 3.3 – Average MSE vs average number of bits for uniform and optimized bit allocation for $K = 17$ and $\theta = 0.001$ ($\kappa = 0.2$, $\boldsymbol{f}_{in}[n] = a[n]^2 + b[n]^2 - 1$, $H(\lambda) = \frac{3}{3+5\lambda}$).

without bounding the transmitted messages, they grow substantially and in consequence, the quantization errors grow as well.

We can analyze the effects of the graph structure and the number of iterations on the performance of the optimization scheme. First, the same processing (with bounding scheme) is applied into a graph with a smaller θ, which results in a bigger discrepancy between edges weights, that is, a less regular graph. The communication constraints are however unchanged, since the number of edges only depends on κ, which remains constant in our experiments. The results can be seen in Fig. 3.2, where the difference between the uniform and the optimized bit distribution is slightly bigger than in the original experiments with a more regular graph (Fig. 3.1). It means that a more regular graph tends towards a more uniform bit distribution,

which seems reasonable. Finally we look at the impact of the polynomial order K on the performance. We run experiments similar to the previous ones, but with a bigger value of K. The difference between the uniform and the optimized bit allocations is shown in Fig. 3.3. We see that the gain due to optimal allocation is bigger than in the previous experiments with the same network (Fig. 3.2). When k grows, the errors propagate more across iterations and the optimized bit allocation tries to compensate it by allocating more bits in the first iteration steps. This allocation improves the performance of the optimized scheme with respect to the uniform scheme, and this improvement effect is more visible for greater K, since the errors propagate for more iterations.

b) Performance with different filters

We now analyze how the performance of our algorithm changes with different filters implemented in the distributed processing task. Graphs are generated in the same way as in Subsection 3.5.1 with $\theta = 0.001$. The same input graph signal is now filtered by different operators with the polynomial approximation order fixed at $K = 9$. Two different tasks are performed with varying filter parameters. First, we perform denoising with distributed Tikhonov regularization. Such a task can be implemented by applying the graph filter

$$\boldsymbol{H}(\lambda) = \frac{\tau}{\tau + 2\lambda^r} \tag{3.37}$$

to the input signal. We fix the value of τ at 10, and we choose for r the values of 1 or 3. The second task consists in distributed smoothing, which can be performed by applying the heat kernel

$$\boldsymbol{H}(\lambda) = e^{-\tau \frac{\lambda}{\lambda_{\max}}} \tag{3.38}$$

to the input signal. We consider τ taking the values 1 or 10. The results of the filter processing using different quantization methods are plotted in Figures 3.4 and 3.5.

Notice that, for the Tikhonov regularization, the higher r, the less smooth the filter. As for the heat kernel, the higher τ, the less smooth the filter. As we can see in Figures 3.4 and 3.5, when the filter becomes smoother, the values of the MSE become smaller (both for optimized and uniform bit allocation). This happens because smooth filters tend to have smaller values of h_k, the filter coefficients derived from the Chebyshev polynomial approximation. Thus, the quantization errors are multiplied by smaller factors, resulting in smaller quantization error.

c) Performance with different input signals

We now analyze the performance for different input graph signals. A graph is generated in the same way as in Subsection 3.5.1 with $\theta = 0.001$, but now the input signal is a uniform random vector, whose values vary from 0 to 1. A Gaussian noise with zero-mean is added to the signal and the denoising is performed by distributively applying the same low-pass filter of Eq. (3.36) to the signal, with its Chebyshev polynomial approximation of order $K = 9$. The experiment is

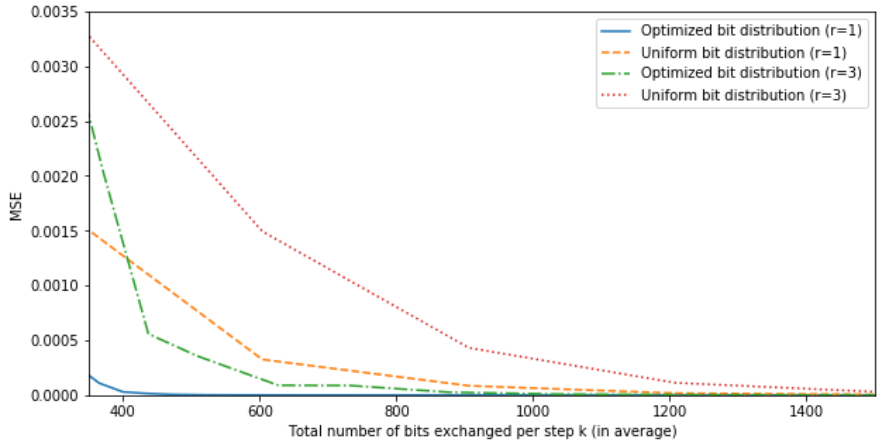


Figure 3.4 – Average MSE vs average number of bits for distributed Tikhonov regularization with optimal and uniform bit allocation for $r = 1$ and $r = 3$ ($\tau = 10$, $\theta = 0.001$, $\kappa = 0.2$, $K = 9$, $\boldsymbol{f}_{in}[n] = a[n]^2 + b[n]^2 - 1$).

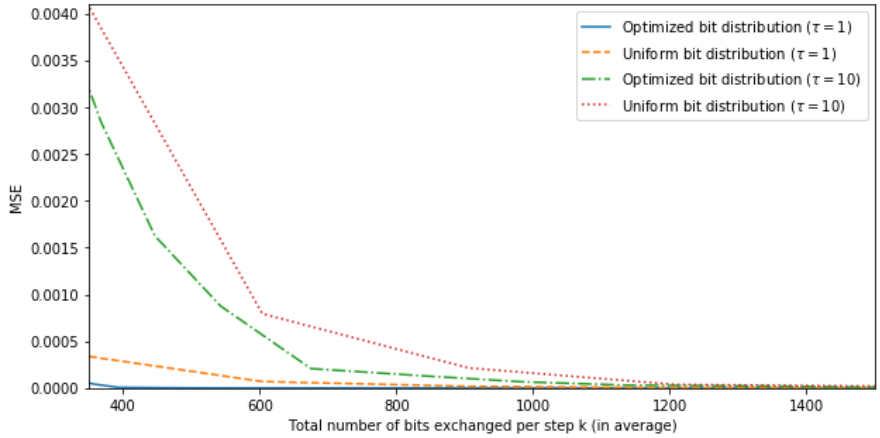


Figure 3.5 – Average MSE vs average number of bits for distributed smoothing with optimal and uniform bit allocation for $\tau = 1$ and $\tau = 10$ ($\theta = 0.001$, $\kappa = 0.2$, $K = 9$, $\boldsymbol{f}_{in}[n] = a[n]^2 + b[n]^2 - 1$).

repeated 1000 times, with a different input signal at each time (while the other parameters remain fixed) and the average performance is computed and shown in Figure 3.6. We can see that the optimized bit allocation improves the filtering performance compared to the uniform bit distribution.

d) Performance on a real dataset

We now consider a dataset containing the rain gauge time series components of a curated set of historical daily rainfall data for the Amazonian rainforest region in Brazil [55]. The data is provided by the Brazillian water management agency Agencia Nacional de Aguas (ANA).

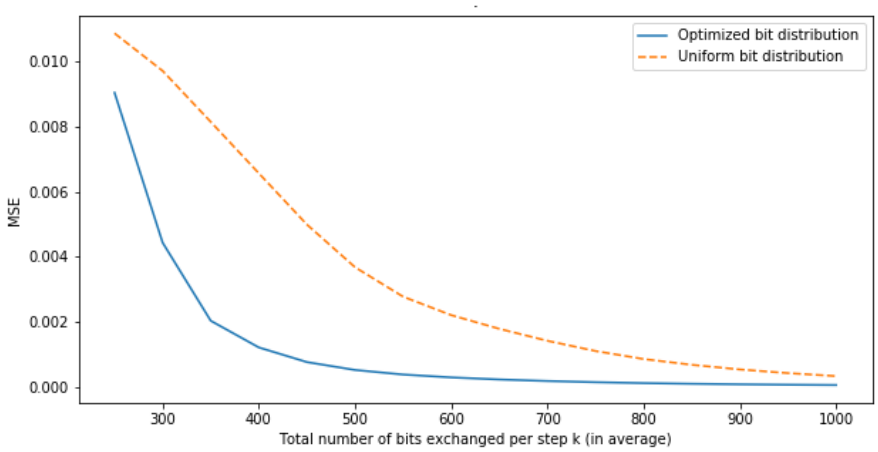


Figure 3.6 – Average MSE vs average number of bits for uniform and optimized bit allocation ($K = 9$, $\theta = 0.001$, $\kappa = 0.2$, $\boldsymbol{H}(\lambda) = \frac{3}{3+5\lambda}$), for random input signals.

Daily rainfall intensities in [mm/day] were recorded by 850 rain gauges spread in a large region spanning the southern Amazonian rainforest to the Cerrado biomes of Brazil from 1926 to 2013. We define a geographical graph, where the nodes of the graph consist of the rain gauge stations. The weight matrix is generated based on a thresholded Gaussian kernel function that takes into account the geographical distance between stations. The year of 2010 was chosen for containing the most complete data, and the average daily rain of that year was computed for each rain site to form the signal value at each node. This results in a graph signal of dimension 850. This graph can be visualized in Fig. 3.7.

A Gaussian noise with zero-mean is added to the rain signal and denoising is performed by distributively applying the same low-pass filter as in Eq. (3.36) (actually its Chebyshev polynomial approximation of degree K) to the signal and results are shown in Fig 3.8. When the signal is processed with the algorithm proposed in [39], where the messages are not bounded, we obtain a MSE of 93.7 for 60000 exchanged bits and a MSE of 87.5 for 80000 exchanged bits. Comparing these results with Fig. 3.8, we can see that our proposed modified algorithm yields much lower MSE values compared to the baseline algorithm. Also in Fig. 3.8 we see that the optimized bit allocation further improves the MSE in the bounded scheme, when compared to the uniform bit distribution.

The rain signal is smooth, and the filter is the same as the one used in Subsection 3.5.1, so the differences in amplitude observed between Fig. 3.8 and the results in Subsection 3.5.1 stem from the range of the input signal and the topology. The graph of the rain topology dataset has more nodes, which increases the number of computations necessary to filter the signal. With the increase in the number of computations, more errors are committed and accumulated, hence why the MSE values tend to be higher.

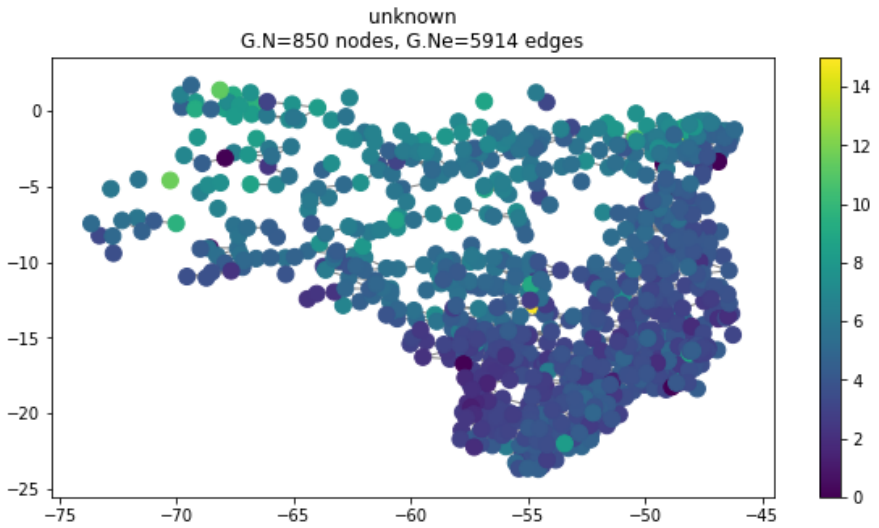


Figure 3.7 – Graph of the rain dataset. Each node represents the rain gauge stations. The colors represent the signal defined on the graph, that is, the daily average of rain in [mm/day] in the year 2010, at each station

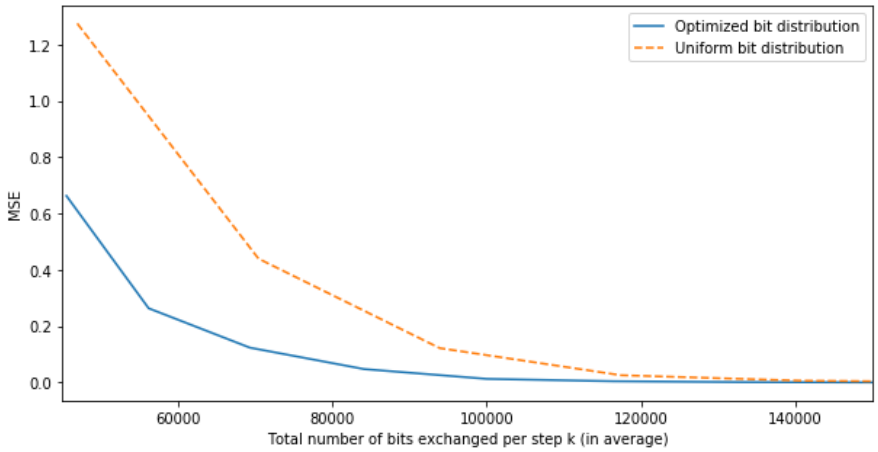


Figure 3.8 – MSE between filter outputs in perfect settings and in quantized settings vs average number of bits for the rain dataset ($K = 9$, $\kappa = 1.8$, $\theta = 0.05$ and $H(\lambda) = \frac{3}{3+5\lambda}$).

3.5.2 Analysis of the bit allocation

a) Evolution of $F_k[n]$

Since $F_k[n]$ influences the optimal bit allocation $b[n,k]$, it is important to understand which factors influence it. A new graph is generated in the same way as in Subsection 3.5.1, also the same graph signal is considered to be filtered by the same filter of Eq. (3.36) with an approximation of order $K = 9$ and edge weight scale factor $\theta = 2$. We compute $F_k[n]$ with Eq. (3.22). Now we observe the relationship of $F_k[n]$ with n and k. In order to analyze the relation

with n first, we plot $F_0[n]$ over the graph in Fig. 3.9, namely $F_k[n]$ at step $k = 0$. At this initial step we have the highest variance of $F_k[n]$, so we can visualize its relationship with n better. The colors represent the values of $F_0[n]$ at different nodes. It can be noted that groups of nodes that are more isolated tend to have higher values of $F_k[n]$. As a result, a higher number of bits needs to be allocated to those nodes. This means that the contribution to the global error of these nodes becomes higher. That happens because isolated nodes tend to amplify their errors, whereas more connected nodes tend to dissolve their errors into the network.

We now plot in Fig. 3.10 the values of $F_k[n]$ versus k. For a given k, we show different values of $F_k[n]$, one value for each n. We observe that the general trend is to $F_k[n]$ decrease with k, as expected since initial errors indeed propagate more than the later quantization errors as discusses earlier.

Some additional results on the relationship between $F_k[n]$ and $b[n,k]$ and an illustration of the error propagation are available in the appendix.

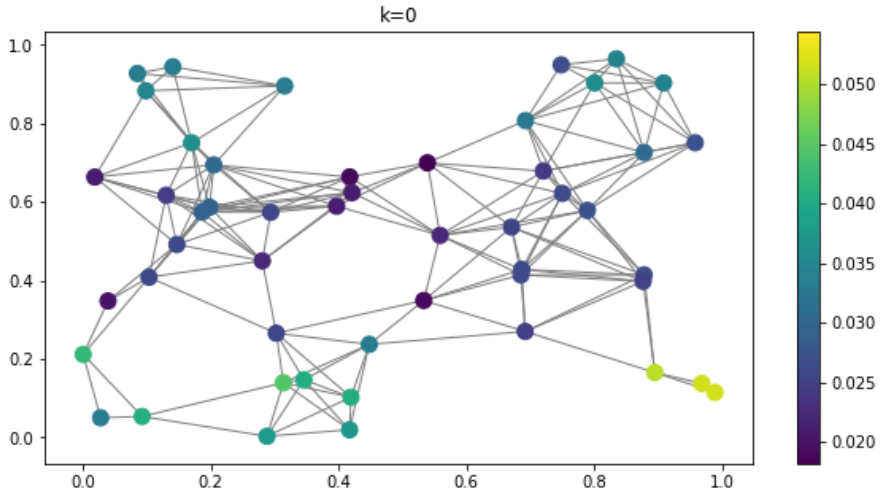


Figure 3.9 – $F_k[n]$ at step $k = 0$. The colors represent the values of $F_0[n]$ at different nodes ($K = 9$, $\theta = 2$, $\kappa = 0.2$, $\boldsymbol{H}(\lambda) = \frac{3}{3+5\lambda}$, $\boldsymbol{f}_{in}[n] = a[n]^2 + b[n]^2 - 1$).

b) Variance of $b[n,k]$

We now have a deeper look at the actual optimal bit allocation in order to understand for which topologies the variance of $b[n,k]$ is high or low. A low variance means that the optimal solution approaches the one of a uniform solution. In this case, it might be simpler to perform a uniform bit allocation even with slightly worse MSE results.

In another set of experiments, 300 graphs generated from a weight matrix based on a thresholded Gaussian kernel function were generated with the edge weight scale factor θ fixed at 2. The edge threshold value κ varies from 0.15 to 0.4, so that we can have graphs with different values of mean degree and variance. The unconnected graphs were removed from the experi-

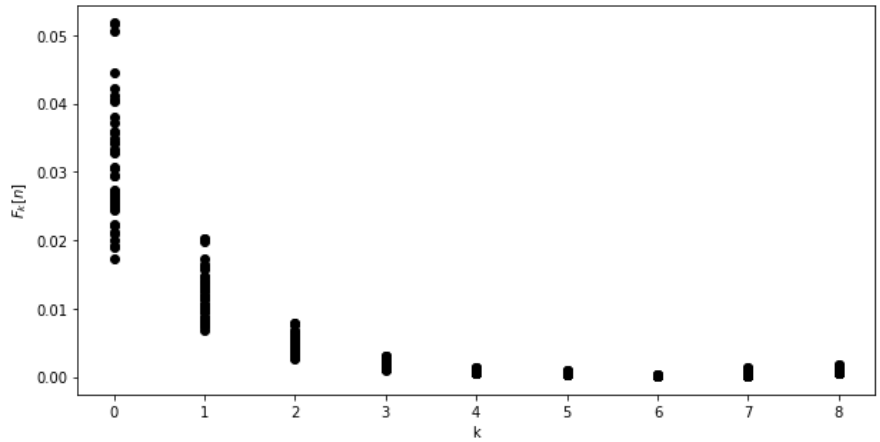


Figure 3.10 – $F_k[n]$ versus k for all nodes ($K = 9$, $\theta = 2$, $\kappa = 0.2$, $\boldsymbol{H}(\lambda) = \frac{3}{3+5\lambda}$, $\boldsymbol{f}_{in}[n] = a[n]^2 + b[n]^2 - 1$).

ment. The graphs were chosen with $N = 50$ nodes. The same graph signal as in Subsection 3.5.1 is generated and filtered with the same filter of Eq. (3.36) with polynomial approximation order $K = 9$. We use Eq. (3.33) for computing $b[n,k]$.

In Fig. 3.11, we can see the relationship between the degrees' variance and mean with the variance of $b[n,k]$ with respect to n ($b[n,k]$ is averaged with respect to k). Each dot corresponds to the experiment resulted with the use of a different graph. It can be noted from Figs. 3.11 that if the graph has low degree variance, that is, it is a more regular graph, it tends to have a more uniform bit distribution. At the same time, graphs with higher degree mean (that is, denser graphs) also tend to have more uniform bit distribution, which is expected since more connected graphs tend to have the quantization errors smoothed out more uniformly in the network. These facts mean that when the graph is both sparser and irregular it will require a more irregular bit distribution hence optimal bit allocation is important.

In Fig. 3.12, we can further see the relationship with the variance of the nodes' eccentricities. The eccentricity of a graph node n is the maximum distance between n and any other node of the graph. In Fig. 3.12 we can also notice the tendency that graphs with more irregular eccentricities require more irregular bit distribution, because more eccentric nodes require more bits than the less eccentric ones, as discussed previously. Which means that, for graphs with more irregular eccentricities, the benefit of solving the optimal bit allocation problem is higher.

A similar analysis with graphs generated from node degrees that follow a binomial distribution is also available in the appendix.

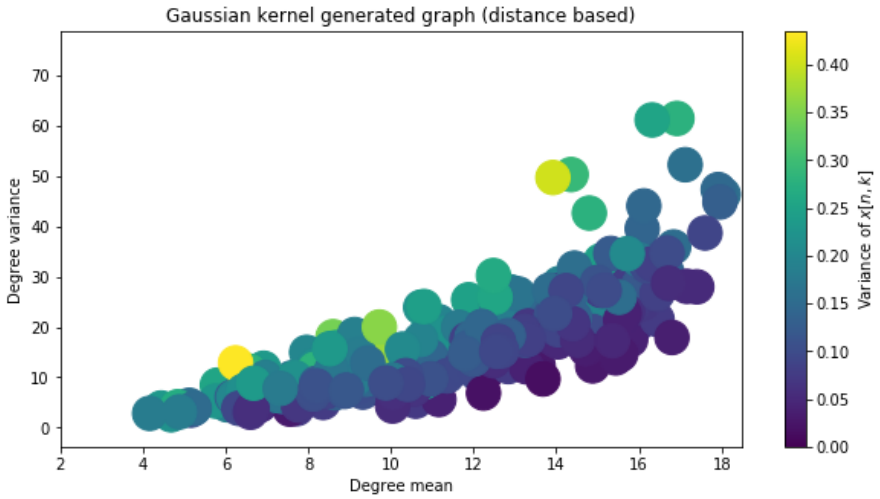


Figure 3.11 – Relationship between the degrees' variance and mean with the variance of $b[n,k]$ for graphs generated from a weight matrix based on a thresholded gaussian kernel function. The colors represent the variance of $b[n,k]$. Each dot is a different graph.

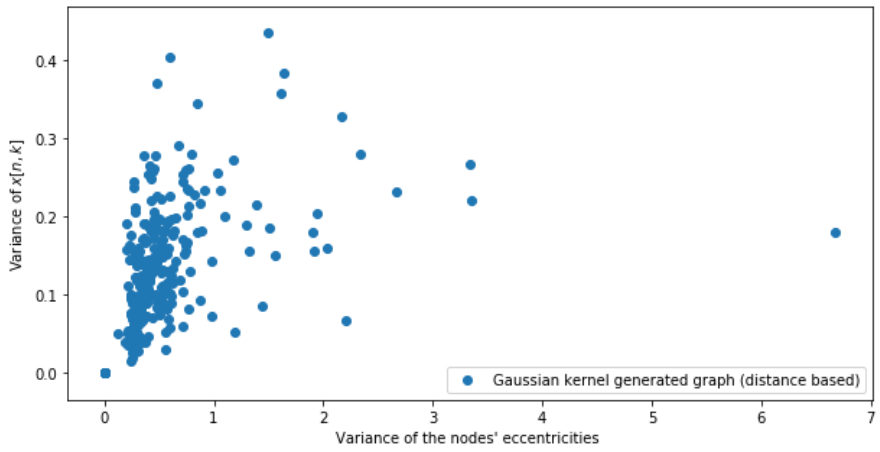


Figure 3.12 – Variance of $b[n,k]$ versus variance of the nodes' eccentricities for Gaussian kernel generated graphs.

3.6 Conclusion

In this chapter, we have shown how the quantization error in distributed graph signal processing tasks can be minimized by bounding the transmitted messages and by optimizing the bit allocation in the network. Our method is important in cases where we have a bit budget constraint. The optimal bit allocation varies with the network nodes and the steps of the iterative filtering process; more specifically, nodes at the border of the network and the initial steps of the iterative algorithm tend to require more bits. Its variance varies with the topology, where irregular and sparse topologies lead to high variance of the optimal bit allocation. Ex-

perimental results show that our distributed processing algorithm substantially decreases the quantization error with regard to previous solutions and to a uniform bit allocation scheme. In the next chapter, we will show how we extend these results to propose an optimal bit allocation for distributed Graph Convolutional Neural Networks - GCNNs.

4 Optimized Bit Allocation for Distributed Processing with Graph Convolutional Neural Networks

4.1 Introduction

Network data require the design of learning and processing methods, that are adapted to the underlying structure. These methods can be implemented either centrally or distributively, depending on the application requirements. Successful learning (for classification, regression or clustering tasks) from network data is challenging due to the complexity of such data. A suitable machine learning model should exploit the underlying graph structure in order to be efficient. Graph Neural Networks (GNN) are excellent tools for learning meaningful representations for data defined over graph. These are generalizations of neural networks for irregular data structures and are widely used in different fields [56].

Centralized processing assumes that the data is available at a central processor. However, there are situations where centralized processing is not possible. For instance, the data could be physically separated (e.g. in the case of wireless sensor networks) and its gathering is either unfeasible or would incur high communication costs or bandwidth, energy and bottleneck constraints. In some scenarios, there might be further requirements on data privacy protection, which prevent data collection in a central entity. In the above settings, the signal is typically processed distributively. Interestingly, GNNs can be implemented distributively by allowing communication among neighboring nodes. This property is typically inherited from graph filters, which are building blocks for GNNs and can be implemented distributively [3]. As a matter of fact, GNNs have been successfully used for inference in distributed settings in many different applications, such as anomaly detection [9], consensus [10, 11], decentralized control [12–18], decentralized resource allocation [19], distributed regression [10], distributed scheduling [20, 21], source localization [10, 22] and traffic prediction [23].

The many works in this field all assume infinite precision messages, or ignore the communication costs altogether in the decentralized inference stage. However, in order to collectively implement the GNN, nodes have to exchange information messages through communication channels. In practice, such messages are quantized and represented by a finite number of bits. This leads to quantization errors in the received signals, which accumulate and eventually

deteriorate the GNN model accuracy, if quantization is not optimized when bit budgets are constrained.

Here, we propose an analytical solution to the optimal bit allocation problem for all message-passing steps of the distributed GNN architecture, which permits to both preserve accuracy of the target task but also to respect network bit budget constraints. We study the effects of quantization error in the accuracy of distributed inference of a graph convolution neural network - GCNN [57–59], that is a GNN architecture with built-in local and distributed nature (i.e., fundamental operations are implemented locally). We assume that the architecture has been trained a priori, possibly under infinite-precision assumption. This scenario can happen, for instance, if the inference quantization conditions (such as the bit budget) are not previously known, and one might opt for a general training to be used for different settings. Alternatively, we might need to use models trained by a third party that were not quantization aware. We formulate an optimization problem where we want to minimize the total quantization quadratic error at the output of the GCNN, subject to communication costs constrained by a given budget. We analytically solve the system of equations and inequalities corresponding to the Karush–Kuhn–Tucker (KKT) conditions, which are necessary conditions for a solution to be optimal in such a problem. Finally, we verify experimentally our optimized solution. We performed experiments for the tasks of distributed denoising and distributed source localization, for synthetic and real datasets. The results confirm that the optimized bit allocation significantly reduces the quantization error compared to a uniform bit allocation strategy and other baseline quantization strategy. Our experiments also show that the optimal bit allocation tends to give a higher relevance to the messages in the middle layers of the GCNN models. Thus, GCNNs can be implemented distributively in a setting with constrained communication, without significant accuracy losses, as long as proper quantization is deployed.

To the best of our knowledge, no prior work has considered quantization in distributed GNNs, but quantization has been considered in other settings. Recently, in the context of neural networks [60–63], and specifically graph neural networks [64, 65], quantization has been considered for the centralized task of model compression. The goal is to reduce the learned parameters to smaller precision, without impacting the accuracy of the model, in order to improve memory usage and inference speedup. In the graph signal processing field, quantization has already been considered in the tasks of distributed graph filtering [44, 50, 51, 66] and distributed graph signal compression [67]. Closer to our work, the authors of [68] analyzed the performance of decentralized inference with GNNs over noisy wireless channels, in uncoded and coded communication systems, and developed retransmission mechanisms to improve robustness for both cases. However, they did not study quantization specifically, nor the design of an optimal quantizer, and are limited to single-layer GNN classifiers.

The remaining of this chapter is organized as follows. In Section 4.2, we present the background and setup for the distributed GCNN inference. In Section 4.3, we introduce our bit allocation

optimization problem. In Section 4.4, we present our analytical solution to this problem. In Section 4.5, we present our experimental results and analysis. Finally, in Section 4.6 we conclude our work.

4.2 Distributed Graph Neural Network Implementation

4.2.1 Network Information Processing

In Chapter 2, we saw how we can filter an input graph signal $\boldsymbol{f}_{in} = \boldsymbol{x}$ to obtain an output graph signal $\boldsymbol{f}_{out} = \boldsymbol{y}$ using the normalized Laplacian as the graph shift operator (GSO) for processing. Using a generic GSO instead, Eq. (2.10) can be rewritten as

$$\boldsymbol{y} = \boldsymbol{H}(\boldsymbol{S})\boldsymbol{x} = \sum_{k=0}^{K} h_k \boldsymbol{S}^k \boldsymbol{x}. \tag{4.1}$$

In order to learn meaningful representations for graph signals, to be used in tasks such as classification, regression or clustering, we need suitable machine learning tools that work for graphs. We call neural networks that operate on graph data as Graph Neural Networks (GNNs), which come in many diverse architectures. We consider specifically the Graph Convolutional Neural Networks (GCNNs) architectures [57–59], due to their structure amenable to distributed implementation. In its most general form, one layer of the GCNN is composed of an FIR graph filter bank, a feature aggregation function, and a point-wise non-linearity. After the last GCNN convolutional layer, the resulting features are combined in a fully connected layer in order to calculate the final GCNN output. We consider a GCNN with L layers and each layer with respectively F_l output features per node. The feature f of layer l is obtained by

$$\boldsymbol{x}_l^f = \sigma\left(\sum_{g=1}^{F_{l-1}} \boldsymbol{H}_l^{fg}(\boldsymbol{S})\boldsymbol{x}_{l-1}^g\right), \tag{4.2}$$

where $\sigma()$ is an activation function, and $\boldsymbol{H}_l^{fg}(\boldsymbol{S})$ is the graph filter of layer l that processes the feature g of layer $l-1$ and outputs part of feature f of layer l.

After layer L, each node aggregates the F_L features with the use of a perceptron (i.e., a single-layer fully connected neural network) and obtains the final output $\boldsymbol{y}$. Both the non-linearity (or activation function) and the fully connected layer are defined per-node (i.e., each node can implement it independently, without communicating with each other), which allows a distributed implementation as described in the next section. An illustration of the GCNN architecture, with distributed operations, is given in Fig. 4.1. Next, we define our quantization error and our minimization objective.

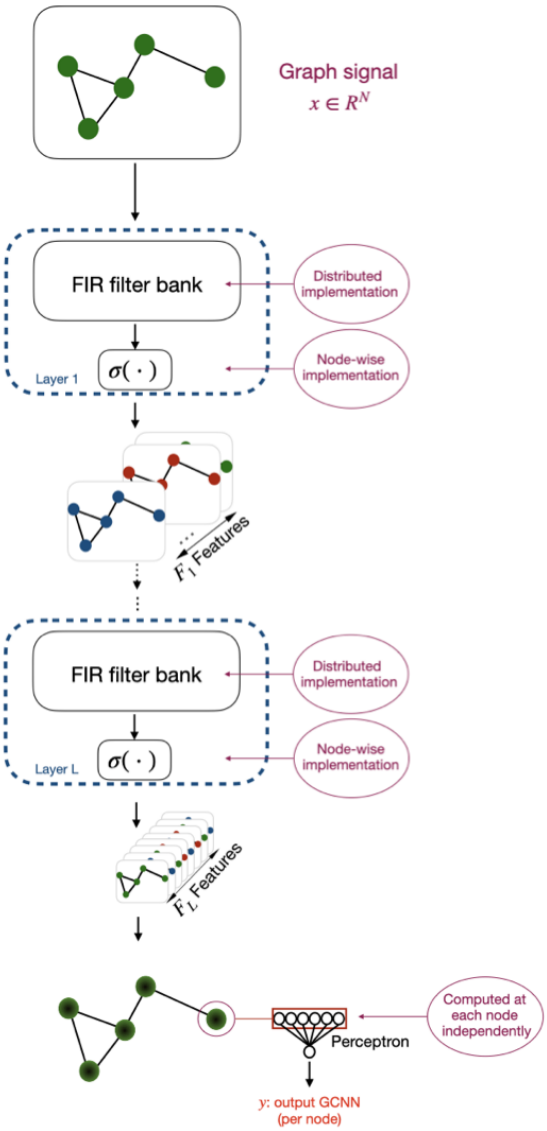


Figure 4.1 – GCNN architecture, with distributed operations.

4.2.2 Distributed GCNN Inference

We consider now the distributed implementation of the inference stage in a GCNN model[1]. We assume that the GCNN model has already been trained, possibly under infinite-precision assumptions. For distributed inference, the nodes share messages in order to jointly implement all the FIR graph filters in every layer of the GCNN model. Each FIR graph filter only requires communication with immediate neighbor. Besides the filters, every layer contains a feature

[1] We assume that the communication network and the data graph that describes the structure of the signal, are aligned.

aggregation function and a point-wise non -linearity operator. Both are defined per node, so no communication is needed with the neighbor nodes. Firstly, the fully connected layer, which combines the resulting features from the last GCNN convolutional layer, is also defined per-node, with shared parameters, so this last step can also be performed locally at each node independently from neighbors. We define as $\boldsymbol{y}$ the output of this GCNN if the inference is performed under infinite-precision assumptions. In practice however, the messages have finite precision, and must go through a quantization process before transmission. This adds error to the system, which accumulates after going through the many iterations and layers of the GCNN. We call the output of the quantized inference step as $\boldsymbol{y}^q$. We define the difference between the inference output with and without quantization as

$$\mathscr{E} = \boldsymbol{y}^q - \boldsymbol{y}. \tag{4.3}$$

The objective is to minimize this error when quantization has to respect a global bit budget for the sum of all exchanged messages.

The quantization error is a function of the number of bits, since it affects $\boldsymbol{y}^q$, which will in turn affect $\mathscr{E}$, so we can determine a relationship $\mathscr{E} = f(b_j^i)$. The total communication cost is also a function of the number of bits, defined as $C(b_j^i)$. To simplify the notation, we define the index i to represent the triple index l, f, g, for instance, the filter $\boldsymbol{H}_l^{fg}$ will be referred as $\boldsymbol{H}^i$ instead. Similarly, we will replace the double index n, k by the index j. We define then b_j^i the number of bits used to represent the messages sent from node n at step k to its neighbors when implementing the filter $\boldsymbol{H}_l^{fg}$. Finally, our objective is to find the bit allocation that minimizes the expected squared error, with a communication cost that obeys a given bit budget B, that is,

$$\begin{aligned} \underset{b_j^i}{\text{minimize}} \quad & E\left\{||\mathscr{E}||^2\right\} = f(b_j^i) \\ \text{subject to} \quad & C(b_j^i) \leq B. \end{aligned} \tag{4.4}$$

We will solve (4.4) in two levels: a filter level bit allocation, and a network-level bit allocation, detailed next.

4.3 Bit Allocation Algorithm

4.3.1 Filter Level Bit Allocation

We develop here the quantization analysis for graph filters, for a general graph shift operator, and additionally derive the equations for the quadratic error under optimal bit allocation. The details are in Appendix B.1 for a generic graph shift operator (GSO), and also in Chapter 3 for the translated normalized Laplacian. The mean squared error (MSE) of the quantization error

$\boldsymbol{Q}^i$ introduced at the output of filter $\boldsymbol{H}^i$ reads

$$E\left\{\| \boldsymbol{Q}^i \|^2\right\} = \frac{(\Gamma^i)^2}{12} \sum_{k=0}^{K-1} \sum_{n=0}^{N-1} F_j^i 2^{-2b_j^i}, \tag{4.5}$$

with

$$F_j^i = F_{n,k}^i = \sum_{j=1}^{K-k} \sum_{p=1}^{K-k} h_{k+j}^i h_{k+p}^i \boldsymbol{S}^{j+p}[n,n], \tag{4.6}$$

where $\boldsymbol{S}^{j+p}$ represents the GSO to the power $j+p$, and h_k^i is the coefficient of the graph filter $\boldsymbol{H}^i$ at step k. The term F_j^i is an auxiliary variable.

In order to compute the bit allocation that will minimize the error in the output of the filter $\boldsymbol{H}^i$, we minimize in Chapter 3 the MSE under the bit budget constraint B^i, by solving the optimization problem with the Karush-Kuhn-Tucker (KKT) conditions. The bit budget $B^i = \sum_j b_j^i$ is the number of bits that can be used in the distributed implementation of the graph filter $\boldsymbol{H}^i$. The sum of the budget of each filter should be equal or smaller than the total bit budget B available for the full GCNN implementation. With similar derivations of the ones in Chapter 3, we find that the resulting optimal bit allocation that minimizes the MSE for filter $\boldsymbol{H}^i$ is

$$b_j^i = -\frac{1}{2\ln(2)} \ln\left(\frac{6\,\mu^i\, d[n]}{(\Gamma^i)^2 \ln(2) F_j^i}\right), \tag{4.7}$$

with

$$\mu^i = \exp\left(-\frac{B^i \ln(4) + \sum_{n=0}^{N-1}\left(d[n] \sum_{k=0}^{K-1} \ln\left(\frac{6d[n]}{(\Gamma^i)^2 \ln 2 \cdot F_j^i}\right)\right)}{K \sum_{n=0}^{N-1} d[n]}\right), \tag{4.8}$$

where K is the FIR filter order (i.e., the degree of the polynomial), N is the graph size, and $d[n]$ is the degree of node n. As discussed in Chapter 3, these results show that it is more efficient to allocate a large share of the bit budget in the first steps of the iterative distributed graph filtering algorithm.

We now calculate the value of $E\left\{\| \boldsymbol{Q}^i \|^2\right\}$ when we use the above bit allocation b_j^i for a given filter bit budget B^i. By applying (4.8) to (4.7), subsequently applying the result into (4.5), and finally simplifying, we obtain

$$E\left\{\| \boldsymbol{Q}^i \|^2\right\} = \tfrac{K}{2\ln 2} \sum_{n=0}^{N-1} \zeta^i \exp\left(\frac{-B^i \ln 4}{K \sum_{n=0}^{N-1} d[n]}\right). \tag{4.9}$$

with

$$\zeta^i = d[n] \exp\left(\frac{-\sum_{n=0}^{N-1} \left(d[n] \sum_{k=0}^{K-1} \ln\left(\frac{6d[n]}{(\Gamma^i)^2 \ln 2 \cdot F_j^i} \right) \right)}{K \sum_{n=0}^{N-1} d[n]} \right). \tag{4.10}$$

As expected, when we have a large B^i budget, the expected error $E\{\| \boldsymbol{Q}^i \|^2\}$ decreases. We can see that with the negative exponential term that depends on B^i in (4.9).

4.3.2 Network-level Bit Allocation

a) Additivity Property

We have analysed thus far the quantization for filter level bit allocation. In this subsection we link those results to the global level GCNN bit allocation problem. The error term used in previous subsection, $\boldsymbol{Q}^i$, is the quantization error introduced at the output of filter $\boldsymbol{H}^i$. Let's define here another error term for filter $\boldsymbol{H}^i$, $\mathscr{E}^i$, as the contribution of that filter error to the global error $\mathscr{E}$. In other words, the error $\mathscr{E}^i$ would correspond to the error obtained by implementing the GCNN in a scenario where only the messages sent while implementing filter $\boldsymbol{H}^i$ are quantized, and all the other messages from the implementation of other filters are assumed to have infinite precision. The term $\mathscr{E}^i$ is usually different than $\boldsymbol{Q}^i$ since the latter goes through the remaining of the network, after the output of $\boldsymbol{H}^i$, undergoing alterations, to become the former.

We want to relate the quantization error at the output of the graph filters to the global quantization error at the output of the GCNN model. So, we first use an approximation called the additivity property

$$E\{||\mathscr{E}||^2\} = \sum_i E\left\{||\mathscr{E}^i||^2\right\}. \tag{4.11}$$

The additivity property has been validated both theoretically and empirically in [69, 70] for the quantization of neural networks. We also experimentally validate this property for our specific GCNN case, in Section 4.5. If we define the total number of bits used to implement filter $\boldsymbol{H}^i$ as B^i, and using the additivity property, we rewrite the problem (4.4) as

$$\begin{aligned} \underset{b_j^i}{\text{minimize}} \quad & E\{||\mathscr{E}||^2\} = \sum_i E\left\{||\mathscr{E}^i||^2\right\} \\ \text{subject to} \quad & \sum_i B^i \leq B. \end{aligned} \tag{4.12}$$

b) Linear Quantization Approximation

In order to relate the quantization errors at the output of the graph filters $\boldsymbol{Q}^i$, to the global quantization at the GCNN model output $\mathscr{E}$, we have previously obtained a relationship between $\mathscr{E}^i$ and $\mathscr{E}$ in (4.11). We still need to find a relationship between $\mathscr{E}^i$ and $\boldsymbol{Q}^i$. We know

that Equation (4.3) becomes $\mathscr{E}^i = \mathscr{E} = \boldsymbol{y}^q - \boldsymbol{y}$ when only the implementation of filter $\boldsymbol{H}^i$ uses quantization, by definition. So, we implement the GCNN with only the quantization of $\boldsymbol{H}^i$, in order to see how $\boldsymbol{Q}^i$ evolves in the network and relate it to the final error output $\mathscr{E}^i = \mathscr{E}$. Before the notation simplification, we were referring to $\boldsymbol{H}^i$ as $\boldsymbol{H}_l^{fg}$ instead, the graph filter of layer l that filters the feature g of layer $l-1$ and outputs part of feature f of layer l, as in Equation (4.2). After implementing $\boldsymbol{H}_l^{fg}$, and only this filter, with quantization, the feature f of layer l becomes

$$(\boldsymbol{x}_l^f)^q = \sigma\left(\boldsymbol{Q}^i + \sum_{g=1}^{F_{l-1}} \boldsymbol{H}_l^{fg}(\boldsymbol{S})\boldsymbol{x}_{l-1}^g\right), \tag{4.13}$$

as opposed to Eq. (4.2) for non-quantized settings.

It has been shown [69] that deep neural networks (DNNs) can be linear to the quantization error, under the assumption that the error is much smaller than the original value. The work focused on the validity of the linearization for quantization noise on the non-linear layers of ReLU, Max-pooling, Sigmoid and ReLU. Given that the differences between GNNs and DNNs lie in the convolutional layers, which are linear in both cases, the conclusions are also valid here. Based on the linearity assumption, we can simplify Eq. (4.13) as

$$(\boldsymbol{x}_l^f)^q = \sigma\left(\boldsymbol{Q}^i\right) + \sigma\left(\sum_{g=1}^{F_{l-1}} \boldsymbol{H}_l^{fg}(\boldsymbol{S})\boldsymbol{x}_{l-1}^g\right). \tag{4.14}$$

Applying Eq. (4.2) on the term on the right of Eq. (4.14), and using again the linearity assumption (which is only valid for the error term) on the term on the left, we obtain

$$(\boldsymbol{x}_l^f)^q = \boldsymbol{Q}^i + \boldsymbol{x}_l^f. \tag{4.15}$$

We now derive how this error term propagates to the features in the following layer $(l+1)$. The feature t of layer $l+1$, under quantization in the filter $\boldsymbol{H}_l^{fg}$, is given by

$$(\boldsymbol{x}_{l+1}^t)^q = \sigma\left(\sum_{g=1}^{F_l} \boldsymbol{H}_{l+1}^{tg}(\boldsymbol{S})(\boldsymbol{x}_l^g)^q\right), \tag{4.16}$$

where $(\boldsymbol{x}_l^g)^q$ with $g \in [1, F_l]$ represents all features from the previous layer l, under quantization in the filter $\boldsymbol{H}_l^{fg}$. Out of those, only when $g = f$ we have the error term (Eq. (4.15)). For every other value of g, $(\boldsymbol{x}_l^g)^q = \boldsymbol{x}_l^g$. If we apply (4.15) into (4.16), we obtain

$$(\boldsymbol{x}_{l+1}^t)^q = \sigma\left(\boldsymbol{H}_{l+1}^{tf}(\boldsymbol{S})\boldsymbol{Q}^i + \sum_{g=1}^{F_l} \boldsymbol{H}_{l+1}^{tg}(\boldsymbol{S})\boldsymbol{x}_l^g\right), \tag{4.17}$$

and using the linearity assumption once more, we obtain

$$(\boldsymbol{x}_{l+1}^t)^q = \boldsymbol{H}_{l+1}^{tf}(\boldsymbol{S})\boldsymbol{Q}^i + \boldsymbol{x}_{l+1}^t. \tag{4.18}$$

If we keep propagating the error of quantization in filter $\boldsymbol{H}_l^{fg}$ through the subsequent layers,

until the final output, we obtain the relationship

$$\boldsymbol{y}^q - \boldsymbol{y} = \mathscr{E}^i = \boldsymbol{M}^i \boldsymbol{Q}^i, \tag{4.19}$$

with

$$\boldsymbol{M}^i = \boldsymbol{M}^{fg}_{L-p} = \sum_{i=1}^{F_L} \sum_{j=1}^{F_{L-1}} \dots \sum_{t=1}^{F_{L-(p-1)}} \alpha_i \boldsymbol{H}^{ij}_L(\boldsymbol{S}) \boldsymbol{H}^{jl}_{L-1}(\boldsymbol{S}) \dots \boldsymbol{H}^{tf}_{L-(p-1)}(\boldsymbol{S}), \tag{4.20}$$

with $L-p=l$ and with α_i being the ith weight of the fully connected layer after the last graph convolutional layer.

c) Approximate Optimization Problem

We now use the above approximations of Eq. (4.19) and Eq. (4.11) to rewrite the bit allocation problem in a simpler form. First, we note that the Euclidean norm $||\ ||_2$ has the following property

$$||\mathscr{E}^i||_2 = ||\boldsymbol{M}^i \boldsymbol{Q}^i||_2 \leq ||\boldsymbol{M}^i||_2 ||\boldsymbol{Q}^i||_2 \leq ||\boldsymbol{M}^i||_F ||\boldsymbol{Q}^i||_2, \tag{4.21}$$

where $||\ ||_F$ denotes the Frobenius norm, as the Euclidian norm and the spectral norm (the matrix norm induced from the Euclidian norm for vectors) are sub-multiplicative (i.e., obey the property $||\boldsymbol{AB}|| \leq ||\boldsymbol{A}||||\boldsymbol{B}||$). Besides, we also use another inequality from linear algebra, $||\boldsymbol{A}||_2 \leq ||\boldsymbol{A}||_F$, in order to relate the error to the Frobenius norm of the operator $\boldsymbol{M}^i$, since it is easier to compute than the spectral norm. We can obtain (omitting the index in the Euclidian norm for simplicity of notation) the following inequality:

$$E\left\{||\mathscr{E}^i||^2\right\} \leq ||\boldsymbol{M}^i||^2_F E\left\{||\boldsymbol{Q}^i||^2\right\}. \tag{4.22}$$

Using the above inequality, we rewrite the optimization problem of Eq. (4.12) as

$$\begin{aligned} \underset{B^i}{\text{minimize}} \quad & \sum_i ||\boldsymbol{M}^i||^2_F E\left\{||\boldsymbol{Q}^i||^2\right\}. \\ \text{subject to} \quad & \sum_i B^i \leq B \\ & B^i \geq 0, \end{aligned} \tag{4.23}$$

assuming that within each filter, the bits are allocated according to Eq. (4.7) and Eq. (4.8). Thus, we effectively minimize an upper-bound on the error, as a proxy for the true optimal bit allocation.

4.4 Bit Allocation Problem Solution

We now propose an algorithm to solve the optimization problem of Eq. (4.23). In order to use our optimization tools, we rewrite the objective and constraint functions as

$$\text{minimize} \qquad f(B^i) = \sum_i ||\boldsymbol{M}^i||_F^2 E\left\{||\boldsymbol{Q}^i||^2\right\}, \tag{4.24}$$

$$\text{subject to} \qquad g_1(B^i) = \sum_i B^i - B \leq 0, \tag{4.25}$$

$$g^i(B^i) = -B^i \leq 0, \tag{4.26}$$

where Eq. (4.26) constraints the solution to positive budgets. The Karush–Kuhn–Tucker (KKT) conditions are necessary conditions for a solution to be the optimal solution of such a problem. If the problem is convex, the conditions become sufficient. As both objective and constraints in Eq. (4.23) are continuously differentiable and convex on B^i, we can actually use the KKT conditions to solve the optimization problem of Eq. (4.24). As the constraints are also affine, regularity conditions are also satisfied.

The solution of the problem should satisfy the stationary condition

$$\nabla f(B^i) + \sum_i \mu_i \nabla g_i(B^i) = 0. \tag{4.27}$$

Applying the equations (4.24), (4.25) and (4.26) into Eq. (4.27), we obtain the solution

$$B^i = \frac{-1}{I} \ln \frac{\mu - \mu^i}{IR^i}, \tag{4.28}$$

with

$$I = \frac{\ln 4}{K \sum_{n=0}^{N-1} d[n]}, \tag{4.29}$$

and

$$R^i = ||\boldsymbol{M}^i||_F^2 \frac{K}{\ln 4} \sum_{n=0}^{N-1} \zeta^i. \tag{4.30}$$

Above, the parameter μ is associated with g_1 and the parameters μ^i are associated with g^i.

The solution also has to obey the complementary slackness conditions

$$\mu_i g_i(B^i) = 0, \tag{4.31}$$

which translate into

$$\mu\left(\sum_i B^i - B\right) = 0, \tag{4.32}$$

$$\mu^i B^i = 0. \tag{4.33}$$

The trivial solution of the optimization problem (every $B^i = 0$) does not satisfy the KKT conditions, since that would mean $\mu = 0$ in Eq. (4.32), considering that we have a positive budget B. If we put Eq. (4.28) to 0, with $\mu = 0$, and solve for μ^i, we would obtain $\mu^i = -IR^i$, which is negative. This means that the trivial solution does not satisfy the dual feasibility condition ($\mu_i \geq 0$).

Therefore, we know that some (at least one) filters would have a non-zero budget. We call this set of filters B^{i+}. Similarly, the set of filters that have zero budget would be called B^{i0} (in most cases this set would be empty, especially with a bit budget that is large enough). We do not know a priori how many (and which) filters each set contains. If we consider only the set B^{i+} in the previous equations, with Eq. (4.33) we obtain $\mu^{i+} = 0$. Applying this to Eq. (4.28), we obtain

$$B^{i+} = \frac{-1}{I} \ln \frac{\mu}{IR^i}. \tag{4.34}$$

A solution with $\mu = 0$ would not satisfy the KKT conditions, since Eq. (4.34) would have a logarithmic function of zero, which is undefined. Thus, to obtain the correct value of μ, we apply Eq. (4.34) into Eq. (4.32), knowing that the term in the right hand side of Eq. (4.32) should be equal to zero. We finally obtain

$$\mu = \exp \frac{-BI + \sum_{i,+} \ln(IR^i)}{|B^{i+}|}, \tag{4.35}$$

where $|B^{i+}|$ is the cardinality of B^{i+}, that is, the number of filters with positive budget. Since Eq. (4.35) is an exponential function, it is always non-negative, which satisfies the KKT dual feasibility condition.

There is still an open question. We do not know from the start how many filters have positive budget, and which ones are those. Thus, we propose the following strategy: we initially consider that all filters have a positive budget. And then we calculate Eq. (4.34) and Eq. (4.35). If the final result is all positive, that means that we have found a solution that satisfies all KKT conditions, therefore is the optimal one. On the contrary, if one (or more) filters have a negative bit budget, the filter with the lowest B^i (the more negative one), is assigned $B^i = 0$ instead. The bit allocation is computed again for the remaining filters, and the same procedure is repeated until we have a bit allocation where everyone is positive. It is guaranteed that at least one filter will be positive ($B^i = B$ in this case). Finally, to be complete, we note that a zero number of bits means that the message being transmitted is assumed to be in the middle of the quantization range. Once Eq. (4.24) is solved, we allocate bit optimally for each filter with Eq. (4.7) and Eq. (4.8). In Algorithm 1 we present the overall solution and summarize the

above results.

Finally, the Algorithm 1 always converges to the optimal solution. Indeed it searches for B^i values that satisfy the KKT conditions, and as mentioned before, these conditions are necessary and sufficient to be the optimal solution, since it is a convex problem. As we start with the assumption that all filters have a positive budget, and decrease the value of $|B^{i+}|$, at any point that the algorithm finds only positive and zero values of B^i, we know that this set of values satisfies the KKT conditions. In case the overall bit budget is very small, and $|B^{i+}|$ eventually becomes 1 in the algorithm search, the KKT equations discussed before would yield the result $B^i = B$. This guarantees that the algorithm never outputs the trivial solution (every $B^i = 0$), which we already know does not satisfy the KKT conditions (for $B > 0$).

Input: Budget B
$|B^{i+}|$ = total number of filters
Calculate μ with Eq. (4.35)
for *every filter* **do**
    calculate B^i with Eq. (4.34)
while *there are negative* B^i **do**
    Assign the filter with lowest B^i to zero
    $|B^{i+}| = |B^{i+}| - 1$; recalculate μ with Eq. (4.35)
    for *all the other filters* **do**
        recalculate B^i with Eq. (4.34)
for *every filter with positive* B^i **do**
    calculate μ^i with Eq. (4.8)
    for *every individual message* **do**
        calculate b^i_j with Eq. (4.7)

Algorithm 1: Optimized bit allocation solution.

4.5 Experimental Results

In this section, we present experiments to validate the proposed allocation scheme on illustrative distributed processing tasks. In all cases, we pre-train a GCNN network architecture [57–59] that is eventually used for distributed inference. We study the performance of distributed inference under the optimized bit allocation and compare it to a uniform allocation and other heuristic baselines. The focus of the first set of experiments is on the regression task of distributed denoising, and later on, we perform experiments on the classification task of source localization.

4.5.1 Distributed Denoising

a) Experimental Setting

The distributed denoising of a graph signal is a node-level regression task. The goal is to process a noisy signal $\boldsymbol{x} = \boldsymbol{y}^{\text{original}} + \boldsymbol{w}$ such that the output $\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ resembles the original signal $\boldsymbol{y} \approx \boldsymbol{y}^{\text{original}}$, where $\boldsymbol{\theta}$ represents the learnable parameters for the GCNN model $f_{\boldsymbol{\theta}}$. The original signal $\boldsymbol{y}^{\text{original}}$ serves as the target, such that a model is trained to minimize the mean squared error (MSE) between the prediction and original signal. We perform experiments on two datasets, a synthetic dataset using graphs of $N = 30$ nodes and a realistic dataset with a graph of $N = 355$ nodes.

We build several synthetic datasets consisting of 1000 graph signals each, supported on the same graph $\mathcal{G}$. We generate 5 such datasets, each of them supported on a different graph $\mathcal{G}$. The 1000 signals, in every dataset, are split into sets of 800/100/100 signals for train/validation/test. We first generate the supporting graph $\mathcal{G}$ by placing $N = 30$ nodes uniformly at random in a unit square, and build the adjacency matrix using the distance between nodes. A connection is made if the distance is smaller than 0.3, and the weight of the edge is computed with a Gaussian kernel function as $W_{ij} = e^{d_{ij}^2/\tau}$ with $\tau = 0.25$, and d_{ij} is the distance between nodes i and j. Then, for each datapoint we sample a random smooth function for the 2D surface of the unit square, defined as $f(x,y) = A\cdot\sin(\alpha x)+B\cdot\cos(\beta y)$, where $A, B \sim \mathcal{U}(0.5, 1.5)$ and $\alpha, \beta \sim \mathcal{U}(3, 7)$ are random latent variables, $\mathcal{U}$ symbolizes the uniform distribution and (x, y) are the datapoint's coordinates. A graph signal $\boldsymbol{y}^{\text{original}}$ supported on $\mathcal{G}$ is generated by assigning to each node the value of the function $f(x, y)$ at that point in the plane. The signal $\boldsymbol{y}^{\text{original}}$ serves as the ground-truth objective, while the input noisy signal $\boldsymbol{x}$ is generated by adding Gaussian noise $\boldsymbol{w} \sim \mathcal{N}(0, \sigma^2)$ as $\boldsymbol{x} = \boldsymbol{y}^{\text{original}} + \boldsymbol{w}$, with $\sigma^2 = 0.1$.

We also validate the proposed method on a non-synthetic dataset supported on a larger graph. The Molène climatic dataset[2] contains recordings from 355 meteorologic stations in the area of Molène, France. By using their geographic position we form a supporting graph $\mathcal{G}$ of $N = 355$ nodes. The dataset has 930 samples corresponding to temperature recordings for 930 days. The original temperature recordings represents the ground-truth signal $\boldsymbol{y}^{\text{original}}$. The noisy input $\boldsymbol{x}$ is generated by adding noise $\boldsymbol{w}$ of variance $\sigma^2 = 0.1$. For all experiments, we always keep the same split proportion (80%/10%/10%).

The network architecture is fixed for the main experiments as a GCNN with $L = 3$ layers, $F = 2$ hidden features per layer and filters of order $K = 5$, as discussed in Section 4.2. With this architecture, the model has a total of 10 filters. As for the graph shift operator (GSO), we use the translated normalized Laplacian. In Chapter 3, it is shown that the signal stays bounded when it is processed with a graph filter with this GSO. The model is trained on the train set for 100 epochs, with a learning rate of 0.001, and we use the Adam optimizer with the hyperparameters arguments $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and batch size of 20. We train 20

[2] The dataset can be found at https://data.europa.eu/data/datasets/ 539f2f8ba3a729478718a7f3?locale=en.

different models with the same network architecture but different random initialization of the model parameters (i.e., the filters coefficients) and study the impact of using an optimized bit allocation on average. For the quantization range, we use the typical empirical values. In order to determine those, we compute the quantization range for each layer and trained model using training data. Specifically, for each trained model, we define the range as the interval $[0, x_l^{\max}]$, where $x_l^{\max}$ is the maximum absolute value observed on all features on layer l during one forward pass (or inference) on the 800 training samples. Precisely, this single forward pass all over the training data is done after the training phase, with the only end goal of determining $x_l^{\max}$, and should not be confused with the distributed testing phase. The minimum value of 0 is guaranteed as the input of each layer is the output of the `ReLU` non-linearity in the previous layer. For the input layer 1, a range of $[-x_1^{\max}, x_1^{\max}]$ is used. We assume that this interval encloses the majority of possible values that features will also take at test time. Thus, each model and layer use a custom working range.

To evaluate the proposed bit allocation strategy, we measure the MSE between the prediction of the model in an ideal communication scenario, and the one in a quantized communication setting. Considering $\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ the denoised signal under ideal communication (i.e., no quantization) and $\boldsymbol{y}^q = f_{\boldsymbol{\theta}}^{\text{quant}}(\boldsymbol{x})$ the denoised signal with quantization during inference, we compute the MSE between their difference and call it the quantization MSE (as opposed to the denoising MSE between $\boldsymbol{y}^{\text{original}}$ and $\boldsymbol{y}$).

As, to the best of our knowledge, there are currently no other bit allocation schemes for the distributed inference on graphs, we design several baselines to compare the proposed optimal allocation scheme to. The *uniform* bit allocation baseline allocates the same amount of bits to all messages. A *layer-wise* bit allocation follows the heuristic of allocating more bits on average to different layers of the network. Specifically, with $L = 3$ layers, we denote $(\Delta_1, \Delta_2, \Delta_3)$ the number of bits difference to the base rate. For example, with a base rate of 8 bits per message, the Layer 1 $(+1, 0, -1)$ scheme allocates 9, 8 and 7 bits, respectively, for messages in layers 1, 2 and 3, effectively allocating more bits to the first layer. In this *layer-wise* scheme, bits are allocated uniformly within each layer. The next baseline is a *range-based* allocation, which allocates bits to each layer depending on the quantization range of the layer. In this baseline, we aim at a uniform quantization step for all messages, by maintaining the ratio in Eq. (2.15) constant, with the constraint of meeting the overall communication budget. Lastly, we use a *graph-based* allocation scheme, which only uses the graph-level bit allocation from the optimal solution, but within the filters, it allocates bits uniformly (as opposed to the full optimal solution, which varies the allocation with the nodes n and filter tap k).

b) Bit Allocation Performance

We train 20 models with random initialization on each of the 5 graphs of the synthetic dataset. Under ideal communication conditions (i.e., no quantization), the GCNN models successfully denoise the input (test) signals, with the denoising MSE in the output signals around 0.05912 ± 0.005. Which is smaller than the additive noise variance of 0.1 added to the clean input signals.

The denoising MSE under optimal bit allocation quantization is of 0.05913 ± 0.05, for an average of 8 bits/message, which shows that, even with quantization in distributed denoising, the GCNN models are quite effective.

In Fig. 4.2, we plot the quantization MSE at different average bitrates for the optimal bit allocation and several baselines. Results are averaged over five synthetic graphs with error bars representing standard deviation. For each graph, we train 20 models with random initialization, use each of them for denoising, and compute the average quantization MSE. It is important to train several models because the filters learned in the GCNN play an important role in the impact of quantization, with some filters being more robust and others more vulnerable to small changes in the signal. We use the same 20 models for all baselines.

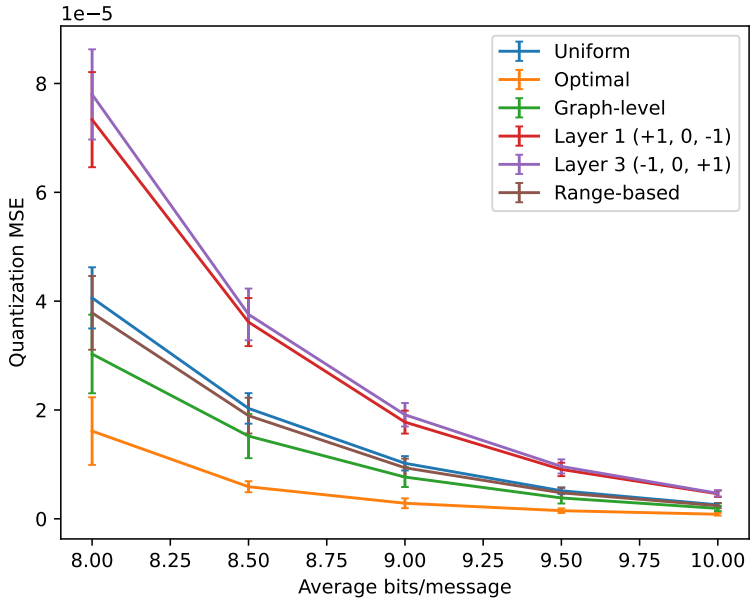


Figure 4.2 – Quantization MSE for different bit allocation schemes on a distributed denoising task. Average across 5 supporting graphs and 20 models for each graph of N=30 nodes, error bars represent standard deviation.

The proposed optimal bit allocation consistently beats every other baseline. It outperforms the uniform allocation by a factor greater than 2 in terms of quantization MSE. The second best scheme is the graph-level one, which only uses the graph-level bit allocation from the optimal solution. This means that a simpler and less complex optimisation can be obtained by only optimizing the bit allocation across filters, simplifying the in-filter allocation to a uniform allocation. This simplified bit allocation (green) still performs better than every other baseline, proving to be a good trade-off between complexity and performance. Yet, the better performance of the complete optimal allocation (orange) shows the incremental performance gain in also optimizing the allocation within the filters. The uniform and range-based allocation perform similarly, with the range-based strategy being slightly better, showing

that an equal quantization step size for all messages is a better strategy than an equal bit allocation for all messages. Finally, the two heuristic strategies of a decreasing bit allocation per layer (red) and an increasing bit allocation per layer (purple) are not good schemes.

We further test the proposed allocation scheme on a larger graph of N=355 nodes, by denoising a signal from the Molène climatic dataset. In this case, we use 5 different models (trained with different initialization of the learnable parameters) and average the results. We can see these results on Fig. 4.3. The standard deviations can be seen in the appendix. Again, here we see that the optimal bit allocation outperforms every other baseline, followed by the sub-optimal graph-level allocation. It shows the validity of our algorithm for non-synthetic and larger datasets.

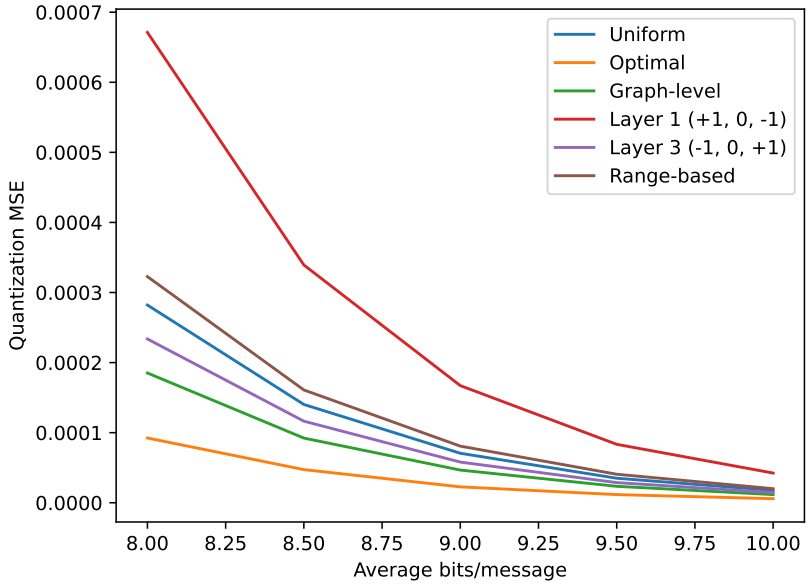
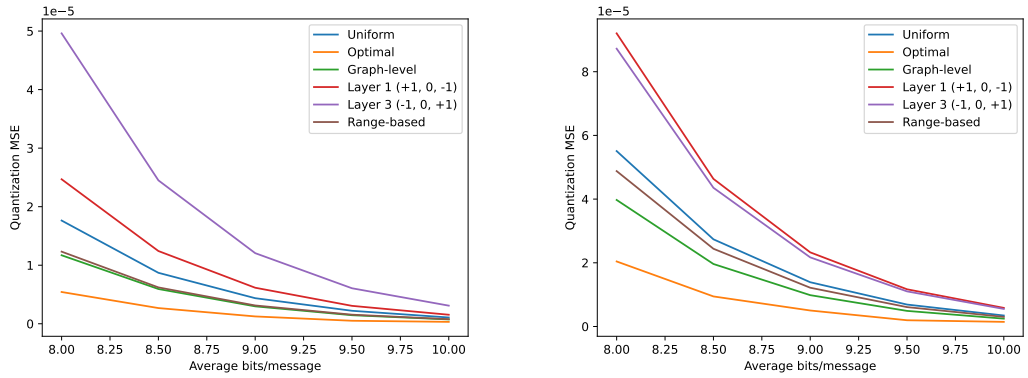


Figure 4.3 – Quantization MSE on the Molène dataset for distributed denoising, supporting graph of N=355 nodes. Average across 5 models. The standard deviations can be seen in the appendix.

Next, we study the impact of the model depth on the bit allocation results. We perform additional experiments on the same denoising dataset supported on graphs of $N = 30$ nodes, in similar settings, using models with $L = 2$ and $L = 4$ layers to validate our solution with different network depths. In Fig. 4.4 we observe that the optimal bit allocation (followed by the sub-optimal graph-level allocation) also outperforms every other baseline, for both $L = 2$ and $L = 4$. The standard deviations can be seen in the appendix.



(a) GCNN with L=2 layers.

(b) GCNN with L=4 layers.

Figure 4.4 – Quantization MSE for different bit allocation schemes on the distributed denoising task. Average across 5 supporting graphs of N=30 nodes. Results for each graph are averaged across 20 models of L=2 or L=4 layers. The standard deviations can be seen in the appendix.

c) Bit Allocation Analysis

We now study in more details the bit allocation results in different settings. We first analyse if any bit allocation pattern emerge, at a layer level, with the optimal bit allocation. Specifically, we are interested in understanding if messages exchanged in any particular layer are more important than in other layers. In Fig. 4.5 we plot the average quantization step size per filter in each of the three layers, for 100 models (20 models per graph, 5 graphs) on the case of synthetic graphs with N=30 nodes. Outliers have been removed for a better visualization. Interestingly, we see a tendency of a lower quantization step (i.e., a higher relative bit allocation) in the second layer, while the quantization step is higher in the first and third layers. This means the messages sent in the middle layers are more important than the ones sent in the first and last layers. A similar trend can be seen in the Molène dataset, in the appendix. A possible explanation for this is that the first layers contribute more to the global quantization error, as the errors accumulate through posterior layers. On the other hand, the last layers are closer to the model evaluation/prediction, so the accuracy impact might be stronger. The middle layer would then constitute a trade-off between these two opposing trends.

In Fig. 4.6 we plot the equivalent quantization step images. For $L = 4$, we have another confirmation of the trend that the middle layers are more important for the bit allocation. For $L = 2$, since there are no middle layers, we cannot observe the same trend, but we can observe that the first layer is more important than the last layer, probably due to the fact that quantization errors in the first layer accumulate more.

We finally perform experiments to empirically validate the additivity property used in Subsection 4.3.2. We use similar settings to those in previous subsections. Specifically, we use the same synthetic dataset and the same 100 trained models as in Section 4.5.1. We derive all the individual filters errors $\mathscr{E}^i$ by implementing the GCNN in a scenario where only the

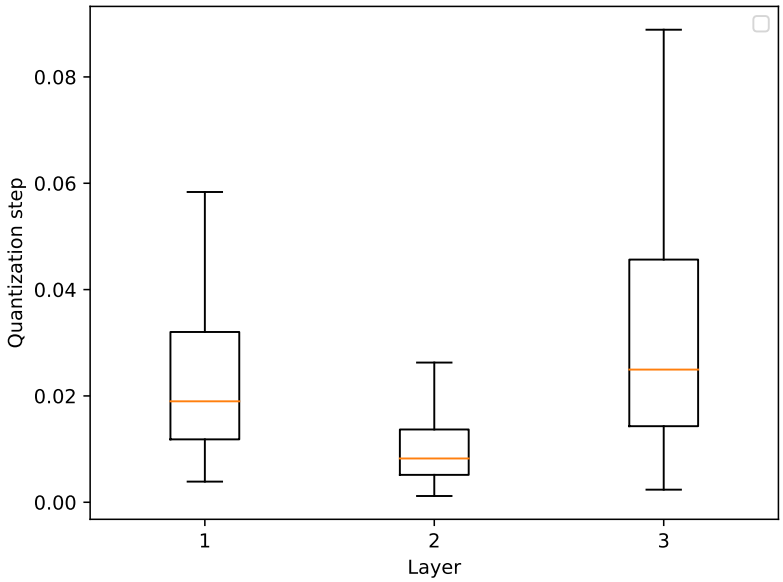
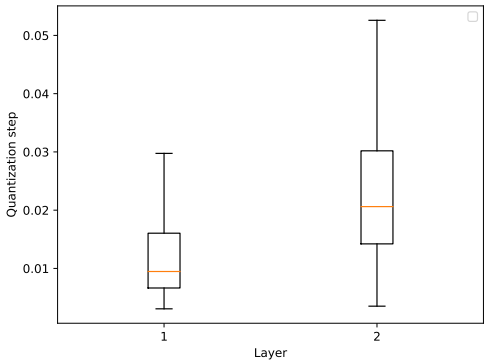
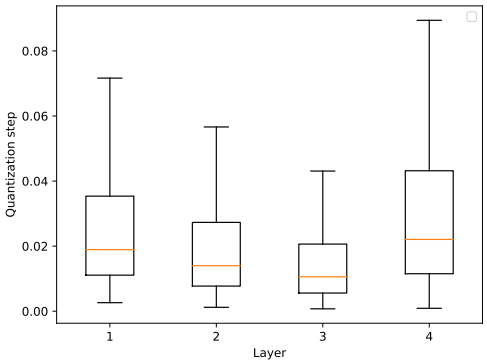


Figure 4.5 – Quantization step size per layer using the optimal allocation scheme. Lower values mean a higher relative bit allocation. Each datapoint represents the average quantization step of a filter in that layer of a model. A total of 100 models (20 models per graph, 5 graphs) are represented in the box plot, with 2, 4 and 4 filters in layers 1, 2 and 3 respectively. Bit budget of 8 bits/message on average.



(a) GCNN with L=2 layers.



(b) GCNN with L=4 layers.

Figure 4.6 – Quantization step size per layer using the optimal allocation scheme for models with $L = 2$ and $L = 4$ layers. Lower values mean a higher relative bit allocation. Each datapoint represents the average quantization step of a filter in that layer of a model. A total of 100 models (20 models per graph, 5 graphs) are represented in each box plot, with 2 and 4 filters for layers 1 and 2, respectively. The last two layers of $L = 4$ also have 4 filters each. Bit budget of 8 bits/message on average.

messages sent while implementing filter $\boldsymbol{H}^i$ are quantized, and all the other messages from the implementation of other filters are assumed to have infinite precision. We sum all values of $\mathscr{E}^i$ for all filters, and compare the sum to the global error value $\mathscr{E}$, obtained when all filters are quantized at the same time. We plot the results in Fig. 4.7, where the y-label represents the true error, i.e., $\mathscr{E}$, and the x-label represents the sum of individual errors. We also draw the equality line in dashed blue. The closer the values are to the equality line, the stronger the validity of the additivity property. Every dot represents an average of 5 runs for each one of the 100 models, to mitigate the randomness of the quantization noise introduced. The red dots represent quantization schemes with 8 bits per message in average, while the blue dots represent quantization schemes with 10 bits per message in average. In the image we can see that all dots are very close to the equality line, which builds a good evidence for the approximate additivity property for quantization in GCNNs. We also plot similar results for the uniform bit allocation and for a lower bit rate in Figs. 4.8 and 4.9, respectively. We can see that the additivity property holds for these cases as well.

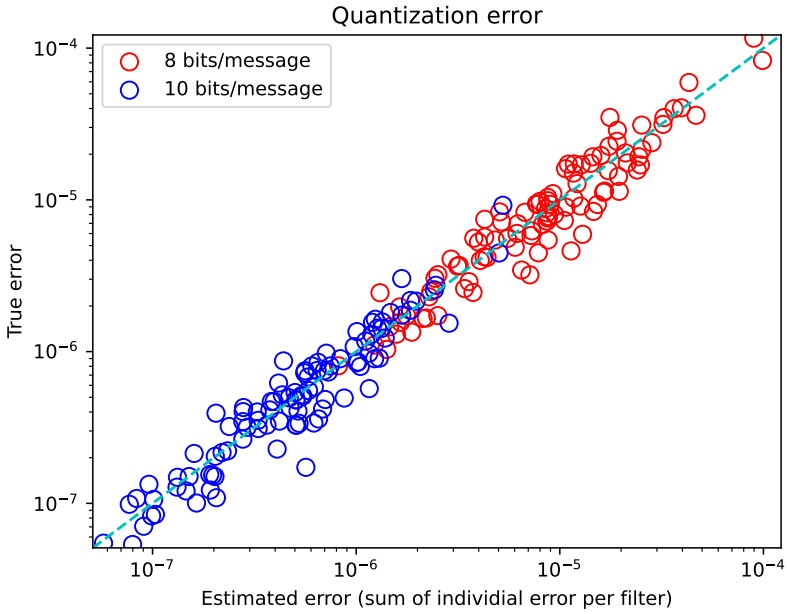


Figure 4.7 – Empirical evaluation of the additivity property for optimized bit allocation. Each point is from one model, representing the average across 5 runs on the synthetic dataset.

4.5.2 Source Localization

a) Experimental Setting

Source localization is a binary classification task at node-level. It refers to a broad class of applications in which the origins of a given diffused signal should be located. For example, finding the beginning of an epidemic, the source of heat, or the origin of an information

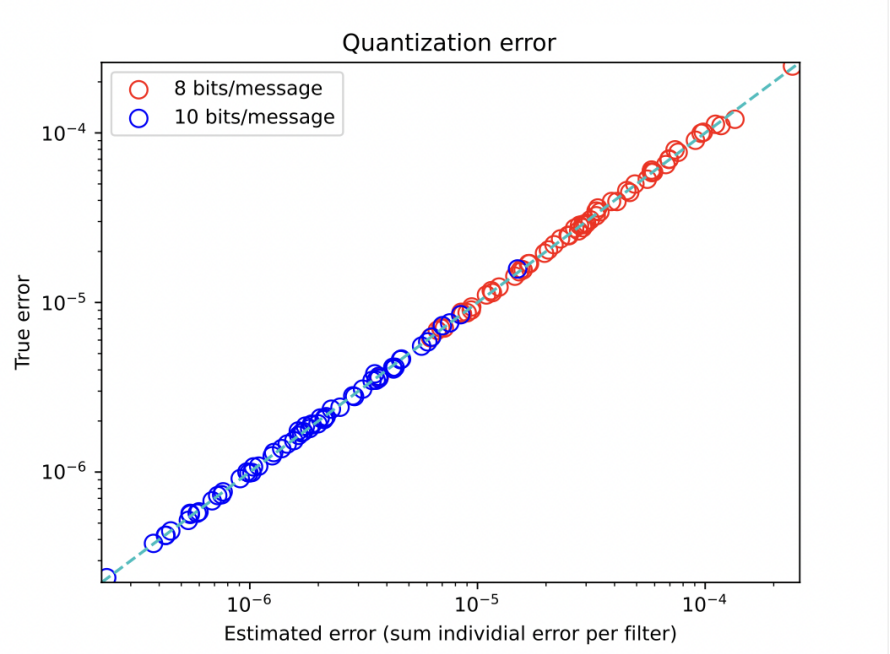


Figure 4.8 – Empirical evaluation of the additivity property for uniform bit allocation. Each point is from one model, representing the average across 5 runs on the synthetic dataset.

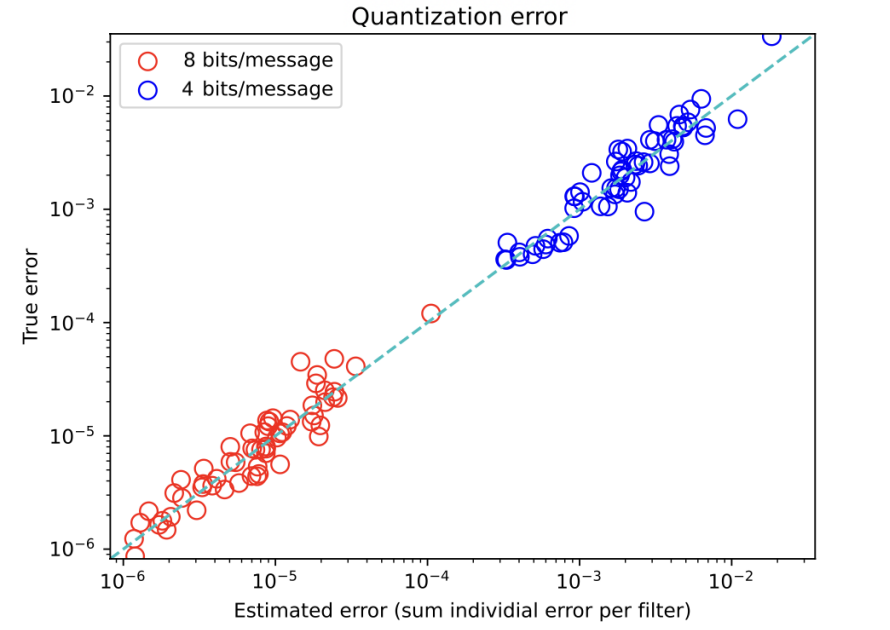


Figure 4.9 – Empirical evaluation of the additivity property for optimized bit allocation. Each point is from one model, representing the average across 5 runs on the synthetic dataset. We show here the lower bit rate of 4 and 8 bits/message in average.

spread in a social network. Given a diffused graph signal $\boldsymbol{x}$, the goal is to identify which nodes are the sources of the original signal $\boldsymbol{y}^{\text{original}}$, that is, which nodes were originally active in $\boldsymbol{y}^{\text{original}}$. The signal $\boldsymbol{y}^{\text{original}}$ can only have nodes with active (1) or inactive (0) values, in this type of task. We generate a synthetic dataset for this task on a supporting graph $\mathcal{G}$ of $N = 30$ nodes. We repeat the process for 5 different graphs $\mathcal{G}$, so we effectively generate 5 different datasets. The generation of a graph is random and analogous to the process described in Section 4.5.1. Then, we construct 1000 samples for each graph, which we split then into 800/100/100 sample sets for train/validation/test, respectively. For each datapoint we sample a source signal $\boldsymbol{y}^{\text{original}} \in \{0,1\}^N$ where nodes are either active, with a value of 1, or inactive, with a value of 0. This signal $\boldsymbol{y}^{\text{original}}$ is also used as the ground-truth. The probability of a node being active is set to $p = 0.1$ with a minimum of one node in $\mathcal{G}$ being active, for each sample. Then, we diffuse the signal as $\boldsymbol{x} = \boldsymbol{S}^k \boldsymbol{y}^{\text{original}}$ for a total of $k = 3$ steps.

Similarly to Section 4.5.1, we use a GCNN architecture with $L = 3$ layers, $K = 5$ filter order and $F = 2$ filters per layer. We train 20 models with random initialization for each of the 5 graphs, 100 models in total, and study the results on average. A model $f_{\boldsymbol{\theta}}$ receives the diffused signal $\boldsymbol{x}$ as input, and predicts, for every node, if it was originally active or not, as $\boldsymbol{y} = \sigma(f_{\boldsymbol{\theta}}(\boldsymbol{x}))$. This is a binary classification task for each node. The point-wise sigmoid function $\sigma(\cdot)$ transforms the output for each node into the range [0,1] such that it can be interpreted as a probability. A node n is predicted as an active source if $\boldsymbol{y}_n > 0.5$. The model is trained for 100 epochs on a weighted cross-entropy loss, with a weight $w = 5$ for the minority class "active" (to compensate for the imbalanced dataset), a learning rate of 0.001, and we use the Adam optimizer with the hyperparameters arguments $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and batch size of 20.

Similarly to the evaluation in Section 4.5.1, we measure the quantization MSE to evaluate a bit allocation scheme. Given the prediction under ideal communication scenario $\boldsymbol{y} = \sigma(f_{\boldsymbol{\theta}}^{\text{ideal}}(\boldsymbol{x}))$ and the prediction under quantization $\boldsymbol{y}^q = \sigma(f_{\boldsymbol{\theta}}^{\text{quant}}(\boldsymbol{x}))$, with $\sigma(\cdot)$ the Sigmoid function (transform a regression model into a classification one), we define quantization MSE as $\text{MSE}(\boldsymbol{y}, \boldsymbol{y}^q)$. This is a different evaluation from previous experiments, so we can observe the performance of our method in a classification task as well. We evaluate the optimal bit allocation scheme against a *uniform* allocation, a *layer-wise* allocation, a *range-based* allocation and a *graph-based* allocation, described in Section 4.5.1.

b) Optimal Bit Allocation Performance

The trained GCNN models were able to solve the source localization task with moderate success. The average performance across all 100 models (20 models per graph, 5 graphs) on an ideal communication scenario was of 26.8% ±2% precision [3] with a 56.4% ± 6.9% recall [4]. However, the performance when the messages are quantized under optimal bit allocation was of 26.5% ±1.8% precision with a 55.0% ± 6.1% recall, which shows that the GCNN can maintain

[3] Precision = true positives / (predicted true positives + predicted false positives).

[4] Recall = predicted true positives/ (true positives + false negatives).

its performance when quantization at inference time is done properly.

In Fig. 4.10, we plot the mean quantization MSE at different average bitrates using the optimal bit allocation scheme and several other baseline schemes. Results are averaged across 5 graphs. For each graph, we train 20 models with random initialization and take the average quantization MSE. The standard deviations can be seen in the appendix. The optimal bit allocation consistently outperforms every other baseline, achieving a lower quantization MSE for all bitrates. Once again, the second best allocation is the graph-level scheme, a simplified version of the optimal scheme, which proves to be a good trade-off between performance and complexity.

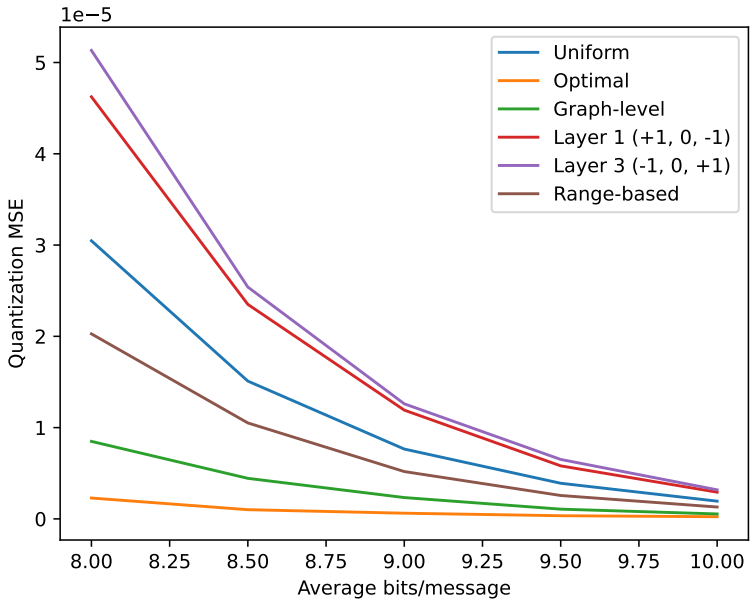


Figure 4.10 – Quantization MSE for different bit allocation schemes on a source localization task. Average across 5 supporting graphs and 20 GCNN models for each graph of N=30 nodes. The standard deviations can be seen in the appendix.

4.6 Conclusion

In this work, we studied the effect of quantization in the distributed implementation of GCNNs and we proposed an analytical solution to the bit allocation problem. We performed experiments for the tasks of distributed denoising (regression) and distributed source localization (classification), for synthetic and real datasets. The results validate the optimised bit allocation, which prove to reduce substantially the quantization error compared to a uniform allocation and other baselines. We also found that the optimised bit allocation tends to give a higher relevance to the messages in middle layers of the model.

So far, we have assumed that the communication network and the data graph that describes the structure of the signal, are the same. This is not always the case, and sometimes we do not even have the data graph a priori. We will explore these questions in the next chapter.

5 Distributed Graph Learning with Smooth Data Priors

5.1 Introduction

In the previous chapters, we have assumed that the communication network and the data graph that describes the structure of the signal, are aligned and known. However, there are settings where the underlying data graph is not given explicitly and has to be learned, which has led to the so-called problem of graph learning (GL)[24–31]. In the graph learning literature, the graph needs to be learned from the signals, so that processing tasks, besides interpretability, can be possible [32, 33]. A common assumption is that the signal values change smoothly across adjacent nodes of the unknown graph topology [34–38]. Under this assumption, the graph learning task amounts to finding the graph structure on which signal values differences on nodes associated with the same edge (and large weights) are minimized. [1]

Centralised settings tend to assume that the message is already gathered, with no cost, in the central processor. Many times the observation data is physically separated and in order to bring it to a central processor, there is a communication cost associated with it, specially if there is not a direct communication path between nodes and the central unit. The communication costs tends to be higher the bigger the network is, since it increases the number of hops to deliver the signals to the central node, and also when the number of graph signals is high. Distributed methods have recently emerged in order to scale to big networks, deal with privacy, bandwidth and energy constraints, or to avoid bottlenecks on the central processor (in the cases where centralized topologies are viable), besides the added robustness to the network in case of node failures.

Some works considered the graph learning task in the context of distributed algorithms. For instance, the authors in [71, 72] consider networks that perform decentralized processing tasks. They propose to observe the evolution of the signals in order to infer the underlying

[1] This chapter contains work which has been published in:
Nobre, I.C.M., El Gheche, M. and Frossard, P., 2022, May. Distributed graph learning with smooth data priors. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 5852-5856).

graph that lead to the observed signal evolution. The graph learning tasks in itself, however, requires a central processor that gathers all observed signals. Similarly, the authors in [73] present a fusion center with the objective of joint recovery of network transmitted data and connection topology in a collaborative network, with a tensor-based approach. The nodes perform in-network coding in order to robustify data recovery. The coded signals are then communicated directly to this fusion center. In a different type of setting, the work in [74] tackles the social learning problem, limited to directed weakly-connected graphs, where the network is partitioned into sending and receiving sub-networks. Given the beliefs collected at a receiving agent, the task is to discover the influence that any sending sub-network might have exerted on this receiving agent. The work does not however retrieve the topology of the network. Closer to our work, the works [75, 76] aim to learn a graph on a graph-based filtering multivariate model, differently from our work that focuses on the smoothness model. None of the works cited above, that consider the graph learning task in the context of distributed algorithms, takes into account the communication costs in the process of learning the graph.

In this work, we propose a distributed graph learning framework that carefully considers the communication costs for exchanging data between the different nodes of the graph. We assume that there is a communication graph that connects the nodes, and we search for a data graph that appropriately describes the structure of the data. We formulate a graph learning problem under the assumption that the data varies smoothly over the target data graph. Then, we propose a new distributed optimisation algorithm to solve the graph learning problem while limiting the communication costs. This is achieved by a combination of local and distributed processing in the network. In particular, we propose to compute the gradient step and most of the optimization constraints in the nodes, and then to satisfy the symmetry constraint with a distributed projection. We propose an objective function with low number of parameters and easy to optimize, which facilitates distributed implementations since the parameters can be pre-set before the algorithm runs distributively. Our problem is solved in two steps, a first one, an initialisation, where nodes exchanges their signals in order to calculate the quadratic differences. The algorithm lowers redundant communication steps by using the more central nodes in a cluster of neighbours to calculate all the quadratic differences in their local vicinity and to share the results with the neighbours, without the information ever leaving the vicinity. In the second step, the distributed optimisation step, the nodes jointly try to find the minimum of the objective function. We use one Adam optimiser for each node, which operate independently from each other. To take into account three out of the four constraints, we project the solution in the constraint sets. For the symmetry projection, specifically, nodes first independently solve their local optimisation cost until convergence, then communicate and jointly apply the symmetry constraint projection. For the other two constraints, the projections are done after each update step of the global algorithm. Finally, to solve the sparsity constraint, we propose a differentiable regularisation term, that also brings stability to the solution. Our experiments show that the communication costs of our distributed algorithm is lowered without compromising the accuracy of the inferred graph, compared to the communication costs of a centralized graph learning algorithm. Besides, that

cost scales better with an increase of the network size or of the size of the data. We also show that sparse networks benefit more from a distributed solution than dense ones. It outperforms the baseline algorithm both in accuracy and communication costs for sparse settings. In our distributed solution, the total communication costs is around 70% in the optimisation algorithm and 30% in the initialisation, which indicates that we save mostly in the initialisation. In these experiments, for the centralised case we assume the costs of transmitting the data from the nodes to a central unity and back. Finally, we propose an incremental bit allocation scheme for our distributed optimisation algorithm, with a marginal analysis technique. We propose a data-driven solution that learns the bit allocation that minimizes the Frobenius norm of the difference between the ground truth and the quantized learned data graph, while obeying a given bit budget. Our solution presents a better trade-off between accuracy and communication costs than implementing a uniform bit allocation scheme for all exchanged messages. Our new algorithm certainly opens the way to other distributed graph learning solutions for dynamic settings.

5.2 Problem formulation

5.2.1 Graph Learning

The goal of the graph learning task is to minimize the Laplacian quadratic form such that the input signals can be represented as smooth signals[2] on the learned graph. Equipped with the smoothness assumption, one can formulate the graph learning problem as follows:

$$\underset{\mathscr{L}^c \in \mathscr{C}}{\text{minimize}} \ \text{tr}(\boldsymbol{X}^T \mathscr{L}^c \boldsymbol{X}), \tag{5.1}$$

where $\mathscr{C}$ is the space of valid combinatorial graph Laplacian matrices defined as

$$\mathscr{C} = \{\mathscr{L}^c \in \mathbb{R}^{N \times N} \mid \mathscr{L}^c_{i,j} = \mathscr{L}^c_{j,i} \leq 0, \mathscr{L}^c_{i,i} = -\sum_{j \neq i} \mathscr{L}^c_{i,j}\}. \tag{5.2}$$

To facilitate the optimization defined in Equation (5.1) with respect to the graph structure, we seek for a valid weight matrix instead of a Laplacian. It is more intuitive, simpler and equivalent to search in the valid weight matrix space [3] [37]. We can thus reformulate the problem as

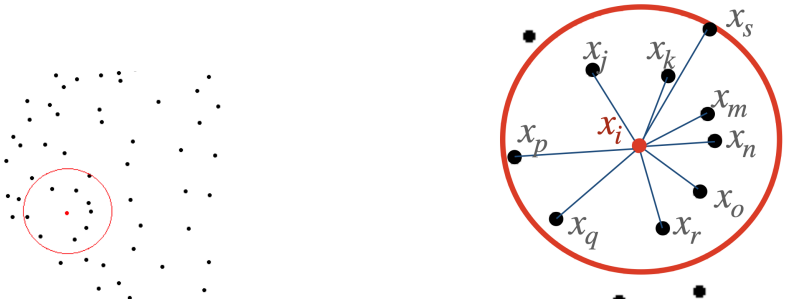
$$\underset{\boldsymbol{W} \in \mathscr{W}}{\text{minimize}} \ \sum_{i,j} \boldsymbol{W}_{ij} ||\boldsymbol{x}_i - \boldsymbol{x}_j||^2. \tag{5.3}$$

[2] See Chapter 2 for definition.

[3] The space of all valid weight matrices is defined by (assuming no self-loops, directed graphs and at least one neighbor per node)

$$\mathscr{W} = \{\boldsymbol{W} \in \mathbb{R}^{N \times N} \mid \boldsymbol{W} = \boldsymbol{W}^\top, \text{diag}(\boldsymbol{W}) = 0, \boldsymbol{W}_{ij} \geq 0, d_i > 0\}.$$

5.2.2 Distributed setup and problem formulation



(a) (b)

Figure 5.1 – Communication settings. a) A wireless sensor network, where each dot represents a sensor/node. Node i is colored in red, along with its communication range defined by the wireless communication radius. b) A zoom into the neighbourhood of i. The nodes inside the communication range define the communication graph edges (in blue). We also represent the graph signal values $\{\boldsymbol{x}_i\}$.

We now consider a network, for instance a wireless sensor network, whose nodes have a communication range that defines which other nodes they can directly exchange messages with, such as the one in Fig. 5.1a. We consider a communication graph $\mathcal{G}_c$ (unweighted), defined over the vertex set $\mathcal{V}$ representing these nodes, and an edges set that describes possible communication links that exist between nodes, such as in Fig. 5.1b. We also assume that the unknown data graph $\mathcal{G}_d$ is a spanning subgraph of the communication graph. A spanning subgraph is a subgraph that contains all the vertices of the original graph but whose edge set is formed from a subset of edges of the original graph. The weights may differ between the edges in the original graph and the edges on the subgraph. In our case, the communication graph is unweighted (weights' values are restricted to 0 or 1), whereas the data graph is weighted. The nodes do not have direct access to a central unit: in order to learn the data graph $\mathcal{G}_d$, they have to use $\mathcal{G}_c$ for communication, under the restriction that the nodes can only communicate directly with 1-hop neighbors on $\mathcal{G}_c$. Finally, each message transmitted on the communication graph eventually induces communication costs.

We propose to learn the data graph in a distributed way, in order to lower the communications costs compared to a centralised solution. It means that each node n should learn which neighbor they have on the data graph $\mathcal{G}_d$ and the corresponding weight of these edges. This should be done with the lowest communication cost possible. We define the communication cost in terms of the total number of messages exchanged between nodes. Since the data graph is a spanning subgraph of the communication graph by assumption, the potential neighbours j, for node i in the data graph, should be picked up from $\mathcal{G}_c$. Equivalently, it has to belong to the neighborhood of i in the communication graph, represented by $\mathcal{N}_i^c$. We can then rewrite the graph learning problem presented above, but in distributed settings, using the

above assumptions. For the node i, the distributed graph learning problem thus reads:

$$\underset{\boldsymbol{W}_{ij}}{\text{minimize}} \sum_{j \in \mathcal{N}_j^c} \boldsymbol{W}_{ij} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 \tag{5.4}$$

$$\text{s.t.} \quad \boldsymbol{W}_{ij} \geq 0 \tag{5.5}$$

$$\boldsymbol{W}_{ii} = 0 \tag{5.6}$$

$$d_i > 0 \tag{5.7}$$

$$\boldsymbol{W}_{ij} = \boldsymbol{W}_{ji}. \tag{5.8}$$

In a distributed setting, solving Equation (5.4) while minimizing the communication cost between the nodes is not trivial. Hence, we first reformulate the problem as:

$$\underset{\boldsymbol{W}_{ij} \geq 0, \boldsymbol{W}_{ii}=0}{\text{minimize}} \ \frac{1}{N} \sum_{j \in \mathcal{N}_j^c} \boldsymbol{W}_{ij} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 \tag{5.9}$$

$$+ \max\{0, \eta - \sum_{j \in \mathcal{N}_j^c} \boldsymbol{W}_{ij}\}^2 \tag{5.10}$$

$$\text{s.t.} \quad \boldsymbol{W}_{ij} = \boldsymbol{W}_{ji}, \tag{5.11}$$

where $\eta > 0$ is the desired minimum degree for every node. Then, we propose to learn $\mathcal{G}_d$ while limiting the communication costs. In particular, we propose an algorithm that consists of a collaborative learning between the graph nodes with local minimization to compute the edge weights $\boldsymbol{W}_{ij}$, $\forall i \in \{1, \ldots, N\}$ at each node (Eqs. (5.9) and (5.10)) and a global optimization to maintain the symmetry constraint as given in (5.11). We will go into details in the next section.

5.3 Distributed Graph Learning Algorithm

We now present the algorithm for solving the above graph learning problem in a distributed way. Let's define the variable that captures the quadratic differences between nodes as $z_{ij} = ||\boldsymbol{x}_i - \boldsymbol{x}_j||^2$. The problem can be locally solved by two steps: an initialization step, where each node i communicates its signal $\boldsymbol{x}_i$ to the neighbors $j \in \mathcal{N}_i^c$, so that the z_{ij} values are obtained at each node, as in Fig. 5.2a; and a distributed optimization step, where the nodes jointly minimize the objective function (5.4). Specifically, every node locally performs gradient descent on the objective function, with the regularization term that targets the degree constraint (5.10), and locally compute the positivity (5.5) and zero self-loop (5.6) constraints. Every node shares these values with their neighbours, but only the information of the weight related to the edge that connects the neighbours. For instance, node i only sends to node s the value $\boldsymbol{W}_{is}^0$, and not $\boldsymbol{W}_{iq}^0$, as in Fig. 5.2c. With these values, the nodes can simultaneously perform a projection to solve the symmetry constraint (5.8), such as in Fig. 5.2d. These steps are repeated until convergence. We present the solution in details next.

5.3.1 Initialization

We propose an initialization algorithm for obtaining the values of $z_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2$ at each node i (Fig. 5.2a). The algorithm takes advantage of the properties of the communication graph in trying to limit communication costs, by making use of situations where nodes share many neighbors. The more central nodes in a cluster of neighbors can calculate all the quadratic differences in their local vicinity and share the results with the neighbors. The information from node i never leaves the neighborhood of this node, which seems appropriate from a privacy perspective, as locality is one of advantages of distributed solutions. It is constructed as follows. While there are still z_{ij}'s to be calculated, every node i sends $\boldsymbol{x}_i$ to its neighbour with the highest degree, out of those whose corresponding z_{ij} has not yet been determined, as long as it is larger or equal to the degree of node i and as long as i did not receive the signals from that node. If these conditions are not fulfilled, $\boldsymbol{x}_i$ is not sent. The nodes that have received the $\boldsymbol{x}_i$ values then calculate all possible combinations of z_{ij} for themselves and for their neighbours. The corresponding results are sent to these neighbours. A similar round happens again, for the nodes that still do not have the z_{ij} of all its neighbours, and repeats until all z_{ij} combinations, of edges that belong to $\mathcal{G}_c$, are calculated. The algorithm lowers the communication costs by reducing the amount of times the $\boldsymbol{x}_i$ values are sent. The algorithm proposed is the one in Algorithm 2.

while *there are still* z_{ij} *to be calculated* **do**
  for *all nodes* i **do**
    if *node* i *doesn't have the* z_{ij} *of all its neighbours* **then**
      Determine the neighbouring node (out of those that z_{ij} is not yet determined) with the highest degree;
      if *the degree of this neighbouring node is bigger or equal than the degree of* i *(and* i *didn't receive the signals from that node yet)* **then**
        Node i sends $\boldsymbol{x}_i$ to this neighbouring node
  for *all nodes* i **do**
    Given all the received values from neighbouring nodes from the previous step (in addition to $\boldsymbol{x}_i$), calculate all possible combinations of z_{ij} and send to the corresponding neighbours

Algorithm 2: Initialization

5.3.2 Optimization

We solve this problem with gradient descent. The positivity (5.5) and zero self-loop (5.6) constraints are local and describe closed sets, so each node can independently project the solution into the set defined by these constraints using ReLU function $f(x) = \max(0, x)$ and setting $x_{i,i} = 0$. Both operations can be done locally. The other two constraints are less trivial to solve. The symmetry constraint (5.8) also represents a closed set, but cannot be locally and independently solved by each node, since it requires communication. As for the

degree constraint (5.7), besides representing an open set, it involves all optimization variables belonging to node i. Next we show how we address these last two constraints.

a) Solving the symmetry constraint

The symmetry constraint also represents a closed set, so it can be solved by projection. It cannot be independently solved by each node, and requires communication. Each node shares its $\boldsymbol{W}_{ij}$ values with the corresponding neighbours and updates them with $\boldsymbol{W}_{ij}^{projected} = \frac{\boldsymbol{W}_{ij}+\boldsymbol{W}_{ji}}{2}$. From an optimization point of view, the projection should be done at every step of the algorithm. But since this projection results in communication costs, we choose a compromise solution where the nodes first solve their local optimization problem (Eqs. (5.9) and (5.10)) , without the symmetry constraint; and, once the local problem converges at every node (figs. 5.2b and 5.2e), they communicate and jointly apply the symmetry constraint (figs. 5.2c and 5.2d). Symmetry projection and optimization alternate until global convergence. [4]

b) Solving the degree constraint

The degree constraint is necessary so that the solution of our problem does not become the trivial solution. It represents an open set, i.e. a set that does not contain its borders, so we replace it by the constraint $\sum_j \boldsymbol{W}_{ij} \geq \eta$, with $\eta > 0$, in order to have an equivalent closed set. We propose to add the regularization term $\max\{0, \eta - \sum_j \boldsymbol{W}_{ij}\}^2$ to the objective function. The term is differentiable, which enables a solution with gradient descent. The sparsity in the learned graph is not explicitly taken into account in the problem formulation. However, the regularisation term $\max\{0, \eta - \sum_j \boldsymbol{W}_{ij}\}^2$ prevents the degree to become too small, by penalizing the solution when it becomes smaller than η. Without this term (or if $\eta = 0$), the solution of the minimization problem would be the trivial solution ($\boldsymbol{W}_{ij} = 0$ for all i, j), which is the sparsest possible. It does not affect the solution when d is equal or bigger than η, since the regularization term becomes zero in that case. This means that, if we choose a small η, the solution tends to be sparse; alternatively, for big η, we tend to produce denser solutions, since the penalization will be stronger. One advantage compared to other solutions (e.g., logarithm function) is that when d approaches zero, the regularization term does not go to infinity, bringing stability to the solution. The cost function we solve at node i becomes then Eqs. (5.9) and (5.10), with the data fidelity term normalized by the size of the network.

[4] It is important to establish the projections order, since we also solve $\boldsymbol{W}_{ii} = 0$ and $\boldsymbol{W}_{ij} \geq 0$ with projection. The $\boldsymbol{W}_{ii} = 0$ projection does not affect the variables outside the diagonal of the weight matrix, so it can be done at any point. As for the $\boldsymbol{W}_{ij} \geq 0$ constraint, that we project with the use of the ReLU function, it is important that it is applied after the symmetry projection, which is what is done in this work. The reason for that is that the symmetry projection step would make variables $\boldsymbol{W}_{ij}$ and $\boldsymbol{W}_{ji}$ either become both positive, both negative or go to the origin. In the origin and positive cases, if the ReLU function is applied after the symmetry projection, nothing changes. As for the negative case, the ReLU operation will bring them to origin. These correspond to the correct euclidean projections into the intersection set between the symmetry and positivity constraints. The other way around, that is, first applying ReLU, then symmetry projection would sometimes result in wrong projections into the intersection set, specifically for the cases where $\boldsymbol{W}_{ij}$ and $\boldsymbol{W}_{ji}$ have opposite signs.

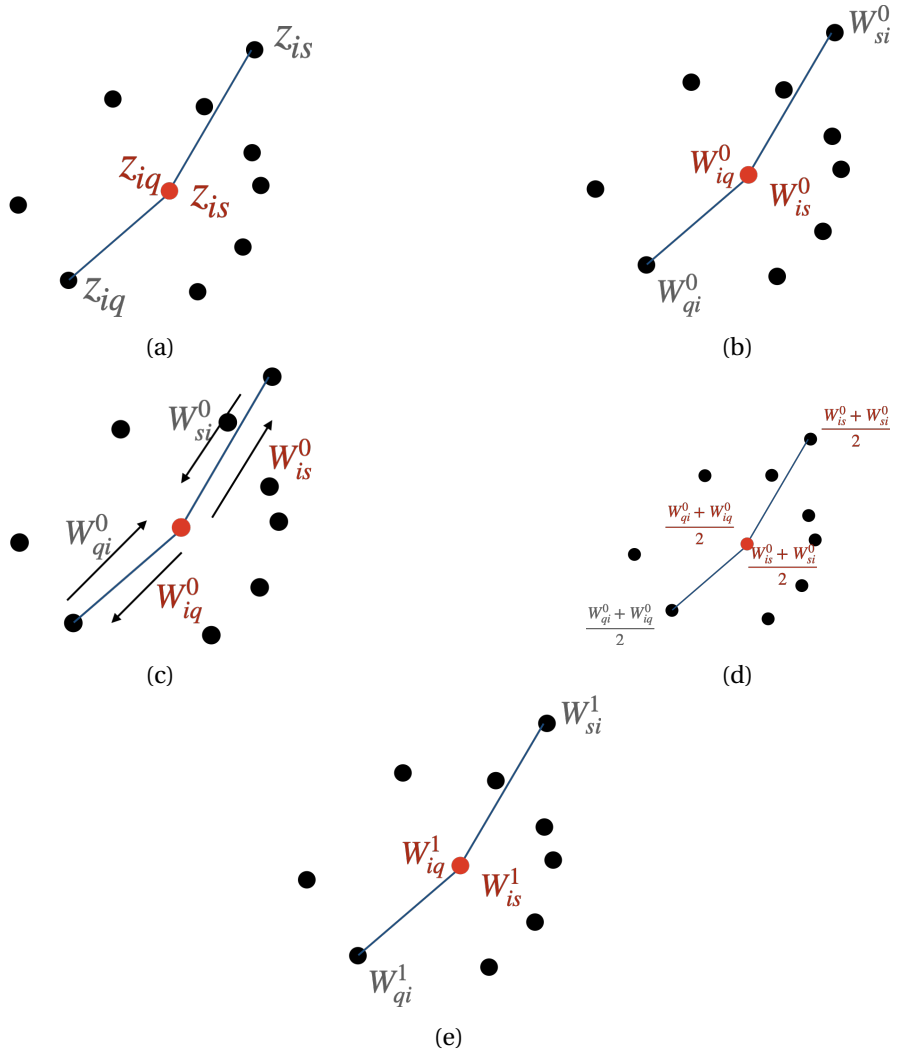


Figure 5.2 – Toy network illustrating the distributed learning algorithm. 5.2a: After initialization, the node i obtains the difference vectors z_{ij}. 5.2b: First estimation of the data graph weights. 5.2c: Sharing of data graph weights. 5.2d: Weights averaging, or symmetry constraint projection. 5.2e: Gradient descent.

One of the advantages of our method is that is has a very simple objective function, without any parameters to be set in an online way. This is appropriate for a distributed scenario, since the task of distributively setting up parameters is a problem on its own, which would also involve extra communication costs. Note that the variable η is not to be optimized, but rather an input of the problem that can be directly adapted to the desired graph sparsity level. In Algorithm 3 we can see the global solution for the distributed graph learning algorithm.

Initialization algorithm (Fig. 5.2a and Algorithm 2)
Initialize all weights $\boldsymbol{W}_{ij} = 1$
while *the global algorithm does not converge* **do**
  for *every node i* **do**
    while *The local algorithm in node i does not converge* **do**
      Relu
      set $\boldsymbol{W}_{ii} = 0$
      Gradient step: minimize Eqs. (5.9) and (5.10)
  After local convergence, local data graph weights are estimated (Figs. 5.2b and 5.2e)
  for *every node i* **do**
    Node i gives $\boldsymbol{W}_{ij}^t$ to every node j that is a neighbor in the communication graph $\mathcal{G}_c$ (Fig. 5.2c);
    Node i receives $\boldsymbol{W}_{ji}^t$ from every node j that is a neighbor in the communication graph $\mathcal{G}_c$ (Fig. 5.2c);
    Node i makes: $\boldsymbol{W}_{ij}^t \leftarrow \frac{\boldsymbol{W}_{ij}^t + \boldsymbol{W}_{ji}^t}{2}$ for all j (Fig. 5.2d);
  for *every node i* **do**
    Relu
    set $\boldsymbol{W}_{ii} = 0$

Algorithm 3: Distributed graph learning algorithm

5.4 Experiments on Distributed Graph Learning

5.4.1 Experimental Settings

We consider synthetic settings and create the communication graph by randomly distributing N nodes uniformly inside the unit square (a square with side lengths 1), and assigning the same communication radius value for every node, equal to $\frac{2}{\sqrt{N}}$[5]. The communication radius is the variable that determines to which neighbours the nodes can communicate to: communication is only possible for nodes whose distances are smaller than the radius. We repeat the graph generation process until producing a connected graph, discarding graphs that are unconnected. A variable called removal-rate defines the ratio of edges in the communication graph that will be randomly deleted, for the data graph generation, with random weights assigned to the edges. This data graph is also our ground truth. We then assign 5000 smooth signals from a probabilistic generative model to the graph. To generate a smooth signal, we interpret graphs as key elements that drive probability distributions of signals. The smooth

[5] In this experiment, we increase N in order to observe the algorithm's performance at different settings. Since these nodes are distributed in a square with side lengths 1 (the unit square), the area that these nodes are distributed into is fixed, regardless of the number of nodes. If we use the same communication radius for all the different values of N, the communication graph would be extremely dense for higher N (it could even produce complete graphs), or alternatively, it would produce disconnected graphs for lower N. These would make conditions very different for distinct values of N, making it hard to compare them. We therefore preferred to change the communication radius, thus maintaining a pretty constant number of neighbours per node, regardless of N. The square root is used to account for the 2-dimensional nature of the problem.

signal follows the normal distribution with zero mean and $\mathscr{L}^{\dagger}$ as covariance matrix ($\mathscr{L}^{\dagger}$ is the pseudoinverse of $\mathscr{L}^{c}$), more details in [34]. We finally fix the removal-rate at 0.5, and we test different number of nodes N, from 150 to 950 [6].

We compare our distributed algorithm with the centralized state-of-the-art approach in [36] for smoothness based topology inference, as well as with a centralized version of our algorithm, where (5.9) is solved directly in a central node of the network, with the symmetry projection done at every step of the optimization algorithm. The communication cost is the number of messages exchanged between two nodes, where one scalar number equals one message. For our distributed framework, we consider the total cost as the sum of the initialization and the optimization costs. For the centralized methods, it is the cost to aggregate all signals into a node with lowest eccentricity, assuming this node would centrally make all computations, plus the costs of bringing the results back to the nodes. The accuracy metrics are defined over the difference between the ground truth data graph and the learned ones. The expected output of the algorithm is that each node possess the learned data graph weights from links that connect their communication graph neighbours. If there is no connection on the data graph, but there is connection on the communication graph, the node should receive a weight with value zero. In all experiments the accuracy metrics are defined over the difference between the ground truth and the solution (proposed or baseline). We use the Pytorch optimizer Adam, with an independent optimizer for each node in the distributed case, and only one optimizer for the centralized cases. That way, the experiments become more realistic and coherent with the theoretical assumptions, that each graph node is separate and makes computations independently from each other.

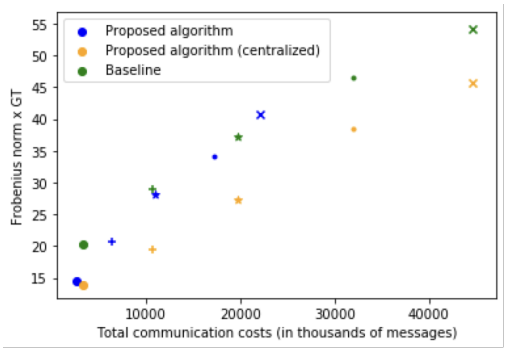
All methods are assumed to know that the data graph should be a spanning subgraph of the communication graph. This is already done in the distributed version by design. For the centralized versions, that means the algorithms set the links that do not belong to the communication graph to zero. In all cases, after we obtain our data graph, we clean any tiny edges (due to numerical error), smaller than 1e-5, to obtain a sparse weight matrix.

5.4.2 Graph Learning Performance

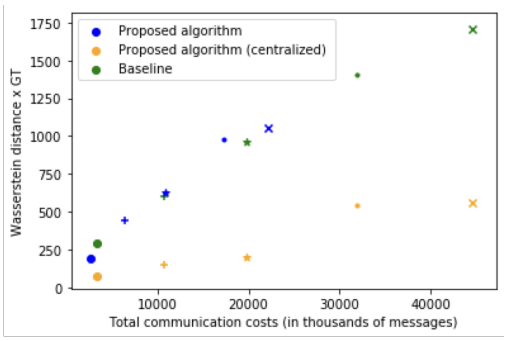
In Figs. 5.3a and 5.3b we plot performance in terms of both total communication costs and accuracy (Frobenius and Wasserstein) for the three algorithms under comparison. These experiments have been run in the sparse settings (average of 11.57 neighbours/node in the communication graph). We notice that our distributed algorithm scales better in communication costs with the increase of N. The distributed version presents better communication costs for all cases. In terms of accuracy, the distributed algorithm is only sometimes slightly worse than the centralized counterparts, due to the approximations inherent to its design. The main difference between both is that the symmetry projection is done at every step of the optimization algorithm in the centralized case. The distributed algorithm represents then a

[6] Sampled every 200

good trade-off between communication costs and graph estimation accuracy.



(a) Frobenius norm (of the difference between learned and ground truth weight matrices) versus total communication costs for the three methods in sparse settings for different numbers of nodes N. Big circle: $N = 150$, + symbol: $N = 350$, star: $N = 550$, small circle: $N = 750$, X symbol: $N = 950$.



(b) Wasserstein distance (between learned and ground truth weight matrices) versus total communication costs for the three methods in sparse settings for different numbers of nodes N. Big circle: $N = 150$, + symbol: $N = 350$, star: $N = 550$, small circle: $N = 750$, X symbol: $N = 950$.

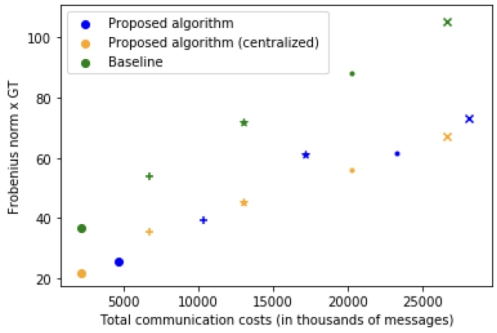
Figure 5.3 – Results for sparse setting.

In the same configuration as before, we now assign 1000 signals to each node. We vary N from 50 to 150. We also assign 4 different values, 0.2, 0.3, 0.4 and 0.5, for the communication radius, and average the results over these values. We compare the communication costs of calculating the differences z_{ij} with both our initialization algorithm and the naive approach where each node sends to the other nodes its respective signals in order to calculate the difference values z_{ij}. We note that our initialization algorithm not only presents lower communication costs, it also scales well with the size of the network.

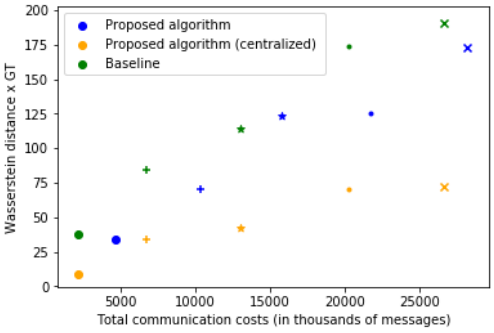
5.4.3 Analysis

Here we study in detail the performance of the algorithm with respect to different parameters.

a) Dense settings In the same conditions as before, we use a bigger communication radius (radius= $\frac{3}{\sqrt{N}}$), which brings an average of 25.01 neighbours per node. We call these the dense conditions. We do the same experiments for the Frobenius norm and Wasserstein norm. The higher density on the communication graph increases the communication costs for the distributed version and decreases costs for both centralised versions. This happens because the distances to the central node are lower when the nodes have more connections (for the centralized case). For the distributed case, it means the nodes have more neighbours to communicate to, which increases communication costs. In this experiment, the distributed method performs similarly to centralized methods, both in accuracy and communication costs. We can see the results for the Frobenius norm in dense settings versus the communication costs in Fig. 5.4a, and for the Wasserstein norm in Fig. 5.4b.



(a) Frobenius norm (of the difference between learned and ground truth weight matrices) versus total communication costs for the three methods in dense settings for different numbers of nodes N. Big circle: $N = 150$, + symbol: $N = 350$, star: $N = 550$, small circle: $N = 750$, X symbol: $N = 950$.



(b) Wasserstein norm (between learned and ground truth weight matrices) versus total communication costs for the three methods in dense settings for different numbers of nodes N. Big circle: $N = 150$, + symbol: $N = 350$, star: $N = 550$, small circle: $N = 750$, X symbol: $N = 950$.

Figure 5.4 – Results for dense setting.

b) Varying sparsity We now use 5000 signals, removal-rate equal to 0.7, we fix $N = 500$ and we vary the communication radius from $\frac{2}{\sqrt{N}}$ to $\frac{3}{\sqrt{N}}$, consequently changing the communication graph sparsity. We plot the difference between the communication costs in the centralized and distributed versions against the average degree in the communication graph in Fig. 5.5. We observe that until 20 neighbours per node, the communication cost in the distributed solution is lower. The observed differences between the Frobenius norms of centralized and distributed versions, in Fig. 5.5, are very small.

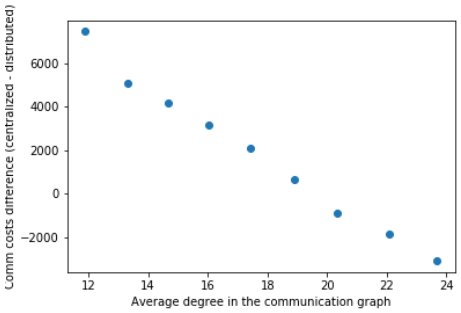


Figure 5.5 – Communication costs difference between centralized and distributed methods against the average degree in the communication graph.

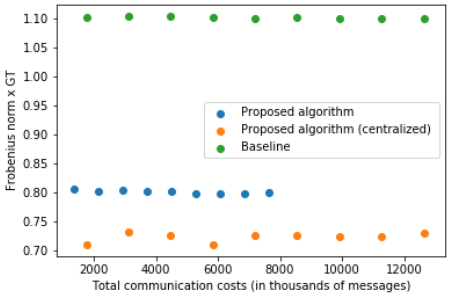


Figure 5.6 – Normalized Frobenius norm (of the difference between learned and ground truth weight matrices) versus total communication costs for the baseline method and both centralized and distributed proposed methods.

c) Number of training signals In the same settings as subsection 5.4.1, we now fix N in 300, and vary the number of signals instead, from 1000 to 10000. In the results (Fig. 5.6), we see that the normalized Frobenius does not change with the variation on the number of signals for the same 3 algorithms considered. For the baseline [36], the normalized Frobenius norm equals around 1.10, the centralized version of the proposed algorithm equals around 0.73 and the distributed version, 0.81. The number of signals directly impacts the communication costs for all methods, as expected. The cost grows linearly with this number. We also note that the distributed version scales better with the growth of the number of signals.

5.5 Quantization for Distributed Graph Learning

The distributed graph learning algorithm proposed above is built under the assumption that messages are transmitted with infinite precision and that communication costs directly relate to the number of messages sent over the network. In realistic scenarios, messages are not sent with infinite precision and are rather quantized in some way. In this section we analyse the quantization effect in our distributed graph learning algorithm, and propose a bit allocation strategy.

5.5.1 Problem Formulation

Our proposed distributed graph learning algorithm, described previously in this Chapter, has two major steps: the initialization and the optimization step. Both require communication in order to be implemented. In total, there are three different stages when messages are exchanged in the algorithm: two in the initialization step, namely when nodes share the original graph signals $\boldsymbol{x}_i$ (Fig. 5.1b), and when they send back their quadratic differences z_{ij} (Fig. 5.2a) according to the initialization algorithm (Algorithm 2); and one in the optimization step, specifically when nodes share their intermediate estimated weight matrix values $\boldsymbol{W}_{ij}$ (Fig. 5.2c). The messages sent on these different stages normally require different communication costs for the same bit rate and impact the quantization error differently, given the different purpose of each stage.

We call S_A the set containing all messages sent in the the first stage, where the $\boldsymbol{x}_i$ values are exchanged. We also call the amount of messages exchanged in this stage as a, which is also the size of the set S_A. There are, in total, $N \cdot M$ values of $\boldsymbol{x}_i$, where N is the size of the network, and M is the number of features, but not all are transmitted and a few are transmitted more than once, depending on the initialization algorithm and the number of edges. Therefore, a is not necessarily equal to $N \cdot M$. Similarly, we call S_E the set containing all messages sent in the second stage, where the z_{ij} values are exchanged. We also call the amount of messages exchanged in this stage as e. The total amount of z_{ij} values is equal to the number of edges in the communication graph. Again, this is not necessarily equal to e, as the number of z_{ij} that are transmitted, depends on the initialization algorithm. Finally, for the third stage, where nodes exchanges the $\boldsymbol{W}_{ij}$ values, we call S_D the set containing all d exchanged messages. This

quantity depends on the number of edges in the communication graph and on the number of rounds needed to converge in the optimization algorithm.

We make use of uniform quantizers to represent every message[7]. We use b_A, b_E and b_D bits, per message, to represent messages from the sets S_A, S_E and S_D, respectively. Within a set, all messages use the same number of bits. We are interested in determining the best allocation $\mathscr{B} = [b_A, b_E, b_D]$, in order to allow the distributed graph learning algorithm to infer the data graph as best as possible, and at the same time obey a total bit budget B. Specifically, given $\mathscr{F}(\mathscr{B})$, the Frobenius norm of the difference between the inferred (with quantization) and the ground truth weight matrices, and $C(\mathscr{B})$ the total bit cost, our objective can be written as the following optimization problem

$$\underset{\mathscr{B}=[b_A,b_E,b_D]}{\text{minimize}} \quad \mathscr{F}(\mathscr{B}) \tag{5.12}$$

$$\text{s.t.} \quad C(\mathscr{B}) \leq B. \tag{5.13}$$

We decide for allocating the same amount of bits to all messages within each set (as opposed to a specific message-based allocation) in order to make a general quantization scheme for the distributed graph learning algorithm, regardless of specific settings. If we would rather target individual messages, the learned allocation rules would heavily depend on the communication graph structure, the input conditions and the data graph (which we do not have a priori), and we are more interested in general rules. We identify that the three sets S_A, S_E and S_D represent the biggest intrinsic differences, in terms of communication costs and impact on the algorithm, between the transmitted messages. That is, those differences are higher for different sets than within sets.

A challenge in solving Eq. (5.12) is that we do not know a priori how the distributed graph learning optimization behaves in every aspect. For instance, we do not know a priori the number of rounds needed for convergence, which can also be changed with quantization. Thus, finding an analytical solution for (5.12) is challenging. Therefore, we approach this problem with an approximated, learning-based solution instead, which will be detailed next.

5.5.2 Training Strategy

We now detail the solution that we propose for the problem in Eq. (5.12). We train the best allocation $\mathscr{B} = [b_A, b_E, b_D]$ that gives the lowest Frobenius norm for the given budget B, using a supervised learning approach. Specifically, we make use of a training dataset that contains all ground truth data graphs [8].

[7] We opt for a learning solution in this chapter (not analytical), thus we do not need to use the assumptions used in the previous chapters. Instead, we let our algorithm learn the suitable allocation based on the available training data.

[8] These graphs have in common: the node set, the communication graph they are based on and the removal-rate value. The communication graphs edges (total number tested between 10 and 25 edges/node, in average) that are randomly deleted are different and the randomly assigned weights are also different.

In order to learn a bit allocation $\mathscr{B}$ that will give the best trade-off between bit costs and accuracy in the inferred graph, we propose an incremental bit allocation scheme with a marginal analysis optimization [77], using the training dataset. An incremental algorithm is a greedy algorithm, which works by allocating one bit at a time to the set S_A, S_E or S_D that performs the best for that partial allocation. Since the best performance is the one with the lowest Frobenius norm and the lowest bit cost, for every incremental bit, we select the set that maximizes the ratio of the change in Frobenius norm to the change in total bit cost. Specifically, we start the algorithm with a 1-allocation[9] (i.e., $\mathscr{B} = [1,1,1]$ for sets S_A, S_E and S_D respectively), and incrementally allocate an additional bit to every single set at a time, temporarily obtaining $\mathscr{B}_A = [2,1,1]$, $\mathscr{B}_E = [1,2,1]$ and $\mathscr{B}_D = [1,1,2]$. We then run our quantized distributed graph learning algorithm (Algorithm 3, with quantization) three times, one for each set s, and respectively use the $\mathscr{B}_A$, $\mathscr{B}_E$ and $\mathscr{B}_D$ allocations to quantize messages. We then proceed to compute $\mathscr{F}(\mathscr{B}_s)$ and $C(\mathscr{B}_s)$ for each set (alternatively, for each of the three runs). Finally, we choose to allocate the incremental bit to the set that maximizes the ratio of the change in Frobenius norm to the change in bit cost. For example, if by using $\mathscr{B}_A$ the algorithm performs best, we would assign $\mathscr{B} \leftarrow \mathscr{B}_A = [2,1,1]$. These steps are then repeated until $C(\mathscr{B})$ attains the bit budget. The training strategy is better detailed in Algorithm 4.

Input: B
Generate nodes
Generate $\mathscr{G}_c$
Generate training dataset (multiple $\mathscr{G}_d$ and signals)
Run Algorithm 3 with training dataset, without quantization, and compute typical quantization ranges
for *every training* $\mathscr{G}_d$ **do**
  $\mathscr{B} \leftarrow [1,1,1]$
  Run Algorithm 3, quantizing with $\mathscr{B}$
  Compute $\mathscr{F}(\mathscr{B})$ and $C(\mathscr{B})$
  while $C(\mathscr{B}) < B$ **do**
    $\mathscr{B}_A \leftarrow \mathscr{B} + [1,0,0]$
    $\mathscr{B}_E \leftarrow \mathscr{B} + [0,1,0]$
    $\mathscr{B}_D \leftarrow \mathscr{B} + [0,0,1]$
    Run Algorithm 3, quantizing with $\mathscr{B}_A$, compute $\mathscr{F}(\mathscr{B}_A)$ and $C(\mathscr{B}_A)$
    Run Algorithm 3, quantizing with $\mathscr{B}_E$, compute $\mathscr{F}(\mathscr{B}_E)$ and $C(\mathscr{B}_E)$
    Run Algorithm 3, quantizing with $\mathscr{B}_D$, compute $\mathscr{F}(\mathscr{B}_D)$ and $C(\mathscr{B}_D)$
    Choose set index $s \in [A,E,D]$ so that $\frac{\mathscr{F}(\mathscr{B})-\mathscr{F}(\mathscr{B}_s)}{C(\mathscr{B}_s)-C(\mathscr{B})}$ is maximized
    $\mathscr{B} \leftarrow \mathscr{B}_s$
    $\mathscr{F}(\mathscr{B}) \leftarrow \mathscr{F}(\mathscr{B}_s)$
    $C(\mathscr{B}) \leftarrow C(\mathscr{B}_s)$
$\mathscr{B}_{trained} \leftarrow$ average of learned $\mathscr{B}$'s for all training datasets

Algorithm 4: Incremental Bit Allocation Training Strategy.

[9] The original algorithm starts with a 0-allocation for all messages, but our distributed optimization does not converge in this case, so me make a slight modification on the algorithm, by starting it with a 1-allocation instead. We assume we have enough bit budget to get started.

5.5.3 Experiments of Distributed Graph Learning with Quantization

a) Experimental Setup

We consider synthetic settings and create the communication graph by randomly distributing $N = 50$ nodes uniformly inside the unit square, and assigning the same communication radius value for every node, equal to $\frac{2}{\sqrt{N}}$, which is equivalent to the sparse settings in Subsection 5.4.2. We repeat the graph generation process until producing a connected communication graph, discarding graphs that are unconnected. We then proceed to generate 40 different data graphs, with the fixed communication graph. Each one of these data graphs is generated by removing edges from the communication graph, according to the removal-rate fixed at 0.5, and assigning random weights to the remaining edges. We additionally generate 1000 smooth[10] signals, for each data graph, from a probabilistic generative model. We then split these datasets (40 data graphs, each with 1000 signals) into training/testing according to the ratio 50/50. Thus, the shared settings for the training and testing datasets are: the node set, the communication graph and the removal-rate value. The data graphs are different since the communication graph edges are removed randomly and the weights are also assigned randomly every time a data graph is generated. The signals from the training and testing datasets are also different. In contrary to the training dataset, the algorithm only knows the graph signals in the testing dataset (which will be used to learn the data graphs), but not the data graphs themselves.

In order to compute the quantization ranges for the transmitted messages, we use the typical empirical values. Specifically, we run our distributed graph learning algorithm (Algorithm 3), without quantization, in our training dataset. We then observe the maximum absolute value $V_s^{\max}$ for every set index s, and we define the range, per set, as the interval $[-V_s^{\max}, V_s^{\max}]$. These ranges are the ones used in our quantizers, both in training and testing stages.

b) Initial Observations

We first run experiments where we quantize only one of the sets (we test 1 to 4 bits/message), and leave the others without quantization. We observe that in general, the quantization of sets S_A and S_E impacts the accuracy more than the quantization of set S_D. Probably because the initialization (sets S_A and S_E) is the first step of the distributed graph learning algorithm, and has a stronger impact on its performance. If the input values of the optimization step are not very correct, the optimization step can perform badly. A proper initialization is thus important. Another consideration is that the set S_D has a low impact on the performance since it is only responsible for the symmetry projection part of the optimization step. When the symmetry projection happens, the nodes already have the estimation of the $\boldsymbol{W}_{ij}$ values, and the communication only happens in order to synchronize those estimations. Thus it is not necessary to spend a lot of bits in this step. Additionally, we observe that the set S_A induces more communication costs than the other two sets. Thus, we expect that the best allocation

[10] Smooth on the data graph.

scheme would allocate more bits to set S_E, as it has a strong impact on the accuracy, without compromising communication costs as the set S_A does.

c) Performance

We then proceed to learn the bit allocation $\mathscr{B}$ with the incremental algorithm described in Subsection 5.5.2, for different bit budgets. We then average the resulting $\mathscr{B}$ for all 20 training data graphs. For a bit budget of 50kB, specifically, we obtain a learned allocation of $\mathscr{B} = [3.1, 7.2, 2.1]$, which confirms our initial observation to allocate more bits to set S_E. We round to integers the bit rate $\mathscr{B}$ learned in the previous step and run our distributed graph learning algorithm on the test set, quantizing messages according to $\mathscr{B}$. We average the Frobenius norms and communication costs for the 20 testing datasets and plot those on Fig. 5.7. For the same testing datasets, we also perform the quantized graph learning task with a uniform bit allocation ($b_A = b_E = b_D$), average the results and plot them on Fig. 5.7, in order to compare with our proposed solution.

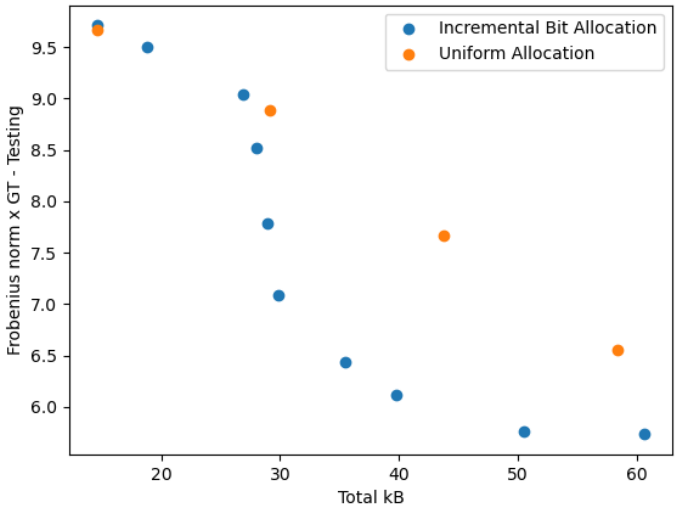


Figure 5.7 – Average Frobenius norm $\mathscr{F}(\mathscr{B})$ versus average total bit costs $C(\mathscr{B})$ for 20 testing datasets (20 different data graphs, each with 1000 signals). The training is done beforehand on 20 different training data graph (each with 1000 training signals).

The results on Fig. 5.7 show that the proposed solution presents a better trade-off between accuracy and communication costs than the uniform bit allocation scheme. In a very low bit rate condition (up to 2 bits per message, in average), the performance is similar (in average) to that of the uniform bit allocation, as we can see with the proximity of both curves for a bit budget up to roughly 30kB. This happens because in this case, with such a small budget, every message should have enough bits to be communicated. This means that the learned solution allocates the same amount of bits for all three sets in these low rate conditions, so the optimized solution learns the uniform solution, and both perform the same. Once

the algorithm has a higher budget to spend, it starts allocating bits differently for the sets, distancing itself in the plotted image from the uniform bit allocation scheme curve. This can be seen with the separation of the two plotted curves around 30kB. In these higher rate conditions, our algorithm has more flexibility to allocate bits, thus the Frobenius norm drops compared to the one of the uniform bit allocation.

d) Additional Experiments

In this subsection, we perform additional experiments that complement the one from the previous subsection. We maintain the same settings that resulted in Fig. 5.7, but now we generate a different set of nodes, and consequently, a different communication graph[11]. Similarly to before, the shared settings for the training and testing datasets are: the node set, the communication graph and the removal-rate value. The data graphs and graph signals of the testing and training datasets are different. We run the same experiments with this new node set and plot the results in Fig. 5.8. We can see in Fig. 5.8 that the same trend of Fig. 5.7 is observed for a different node set and that the incremental bit allocation again outperforms the uniform bit allocation.

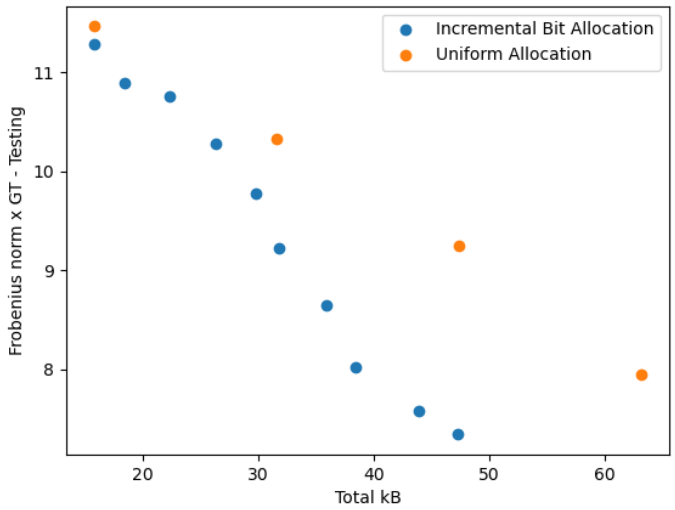


Figure 5.8 – Average Frobenius norm $\mathscr{F}(\mathscr{B})$ versus average total bit costs $C(\mathscr{B})$ for 20 testing datasets (20 different data graphs, each with 1000 signals). The training is done beforehand on 20 different training data graph (each with 1000 training signals). The same settings used for Fig. 5.7 are used here, but with a different node set and communication graph.

We now change the network size for $N = 100$, we maintain the other settings constant[12] and

[11] A new set of $N = 50$ nodes is randomly distributed uniformly inside the unit square. This generates a different communication graph, with the same communication radius value.

[12] Similarly as before, the shared settings for the training and testing datasets are: the node set, the communication graph and the removal-rate value. The data graphs and graph signals are different.

run the experiments with the incremental bit allocation strategy once more. We plot the results in Fig. 5.9. Finally, we test our strategy in the dense settings of Subsection 5.4.3, that is, we use a communication radius value of $\frac{3}{\sqrt{N}}$, and sequentially plot these results in Fig. 5.10. We observe in Figs 5.9 and 5.10 that the same trends from previous experiments are kept, particularly that the incremental bit allocation outperforms the uniform bit allocation.

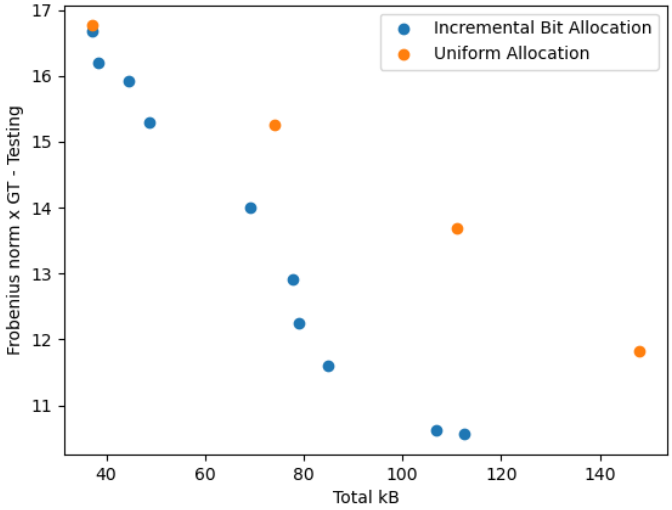


Figure 5.9 – Average Frobenius norm $\mathscr{F}(\mathscr{B})$ versus average total bit costs $C(\mathscr{B})$ for 20 testing datasets (20 different data graphs, each with 1000 signals). The training is done beforehand on 20 different training data graph (each with 1000 training signals). Same settings as for Fig. 5.7, but with $N = 100$.

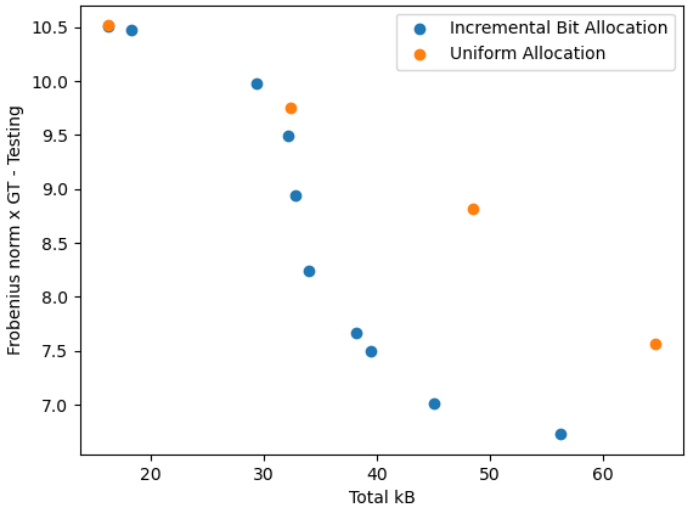


Figure 5.10 – Average Frobenius norm $\mathscr{F}(\mathscr{B})$ versus average total bit costs $C(\mathscr{B})$ for 20 testing datasets (20 different data graphs, each with 1000 signals). The training is done beforehand on 20 different training data graph (each with 1000 training signals). Same settings as for Fig. 5.7, but with denser graphs, i.e., communication radius value of $\frac{3}{\sqrt{N}}$.

5.6 Conclusion

We have proposed a distributed graph learning framework that considers communication costs. We model the problem with an objective function that has a low number of parameters and is simple to optimize. We solve it in two steps, an initialisation step and an optimisation step. In the latter, we use one independent optimiser per node, and we incorporate constraints with a mix of projection methods and a regularisation term. Our experiments show that the communication cost is lowered in the distributed algorithm without compromising the accuracy. Our solution scales better in terms of communication costs with the increase of the network size and the number of signals, compared to centralised solutions. We also show that sparse networks benefit more from a distributed solution than dense ones. Moreover, distributed solutions have other benefits by design, such as allowing the information to stay local, improving privacy and robustness against central node failure. Finally, we have also proposed an incremental bit allocation scheme for our distributed optimisation algorithm, with a marginal analysis technique. We have shown that our proposed solution presents a better trade-off between accuracy and communication costs than the uniform bit allocation scheme. Additionally, compared to a message-specific solution, our solution does not depend on the data graph and is a good trade-off between performance and computational complexity.

6 Conclusion

6.1 Summary

In this thesis we proposed new methods to quantize and allocate bits in the different steps of messages exchanges between nodes, while implementing diverse distributed signal tasks. Our methods aim to find the best trade-off between tasks performance and communication costs. In these distributed settings, the graph nodes send messages that have limited precision in realistic scenarios, hence they undergo quantization. We mostly considered the high rate regime and uniform quantizers for each message. We introduced original methods for adaptive quantization and illustrate their implementation in three specific tasks: graph filtering, graph neural networks and graph learning. Below, we summarize the achievements of this thesis, for each one of these tasks.

Firstly, we considered the problem of minimizing the quantization error in distributed graph filtering tasks. We accomplished that by initially bounding the transmitted messages, and consecutively by optimizing the network bit allocation. Our technique proved to be advantageous when we have a limited bit budget. In the analysis of our optimized bit allocation algorithm, we further found that the nodes near the network's edge and the first steps of the iterative algorithm require more bits for representing transmitted messages. Additionally, the proper bit allocation variance tends to be more heterogeneous with sparse and irregular topologies, than for dense ones. We then experimentally validated the proposed distributed processing algorithm method. We found that, in comparison to the uniform bit allocation technique, our solution greatly reduces the quantization error.

Second, we focused on the implications of message quantization in distributed implementations of Graph Convolutional Neural Networks (GCNNs), a specific type of Graph Neural Networks. When the necessary messages for distributed implementation are quantized, errors occur in the received signals, which accumulate and eventually degrade the accuracy of the GCNN model. We posed an optimized bit allocation problem, and proposed an analytical solution with the use of the Karush–Kuhn–Tucker (KKT) conditions. We then carried out experiments on synthetic and real datasets for the tasks of distributed denoising (regression) and

distributed source localisation (classification). The results confirmed the benefits of the optimized bit allocation algorithm, which reduces quantization error significantly compared to a uniform allocation and alternative baselines. Interestingly, our analysis on the optimized bit allocation showed that messages in the middle layers of the model tend to be more important with respect to the accuracy of the GCNN.

Finally, we considered the problem of minimizing communication costs in distributed graph learning tasks. We consider a communication graph that allows data exchanges between nodes, and we learn a data graph that not only is a spanning subgraph of this communication graph, but also describes the graph signal structure. We modelled this problem with an objective function that has a small number of parameters. In order to solve it, an initialization step and an optimization step were proposed. For the optimization step, one independent optimizer was used for each node, and optimization constraints were addressed by using a mix of projection methods and the addition of a regularisation term. The experimental results demonstrated that the distributed approach reduced transmission costs in terms of number of exchanged messages without sacrificing accuracy compared to a centralized solution. We also found that, our method scales better in terms of communication costs as the network size and the volume of signals increases. Additionally, we showed that a distributed solution works best for sparse networks than dense ones. The proposed algorithm outperforms baseline algorithms, both in accuracy and communication costs for sparse settings. Finally, we consider the problem of quantization of messages in the distributed graph learning algorithm. Using a marginal analysis method, we presented an incremental bit allocation scheme which, compared to the uniform bit allocation technique, offers a superior trade-off between accuracy and transmission costs counting in number of bits.

6.2 Future Work

While this thesis showed many different and efficient methods to quantize messages for distributed graph signal processing and learning tasks, there remain many opportunities for extending this work. We discuss a few of them in this section.

First of all, we believe that an interesting extension of this study would be to investigate the use of non-uniform quantization for each message, to further enhance the performance of our previous results, especially towards low-rate regime. We could also extend this study by exploring the use of different quantizers for different nodes, and by investigating whether that would provide additional gains. Considering different devices could be used in the network, with different communication and processing capacities, it would also be pertinent to study cases where node-specific budgets are imposed, instead of an overall budget imposed for the whole network. Specifically for the distributed graph neural networks study, an interesting future direction would be to train models that are intrinsically robust to quantization, which could then be used with the optimal bit allocation strategy for additional accuracy improvement. Besides quantization, it would be interesting to study settings where messages can be

lost during the distributed algorithm implementation, and its impact on model performance. We believe it is important to design algorithms and architectures that are also robust to such message loss, since these settings are realistic and in line with real world implementation conditions. A future direction specific for the graph learning work could be to extend it to data graphs that are not spanning subgraphs of the communication graph, and to exploit the benefits of a distributed graph learning framework for such case. We could also extend the graph learning framework to different signal models, that are not necessarily smooth on the graph.

In conclusion, this thesis suggests novel approaches for quantizing and allocating bits in the distributed implementation of various distributed graph signal tasks. Our mathematical investigations and empirical findings show that it is possible to achieve efficient trade-offs between accuracy and communication costs. Our research efforts will hopefully speed up the development of intelligent distributed algorithms for network data that balance performance, transmission rate, and computational complexity. In the wide range of possible applications such as social, environmental, energy, transportation, and other domains, such technology will be critical for our increasingly interconnected future society.

A Appendix of Chapter 3

A.1 Bit allocation in the network

We now extend the experiments on Subsection 3.5.2 to observe the relationship between $F_k[n]$ and $b[n,k]$. In the same experiment of Subsection 3.5.2, we show in Fig. A.1 the values of $b[n,k]$ plotted as the colors of the graph. Comparing Figs. 3.9 and A.1, we notice that, when the values of $b[n,k]$ are high, they are also high in $F_k[n]$; but the opposite does not necessarily hold true. This happens because the values of $F_k[n]$ have the tendency of being higher on the more isolated nodes, but the values of $b[n,k]$ take into consideration communication costs too. Thus they tend to be higher on those nodes that are not only at the border of the graph but have also small degrees.

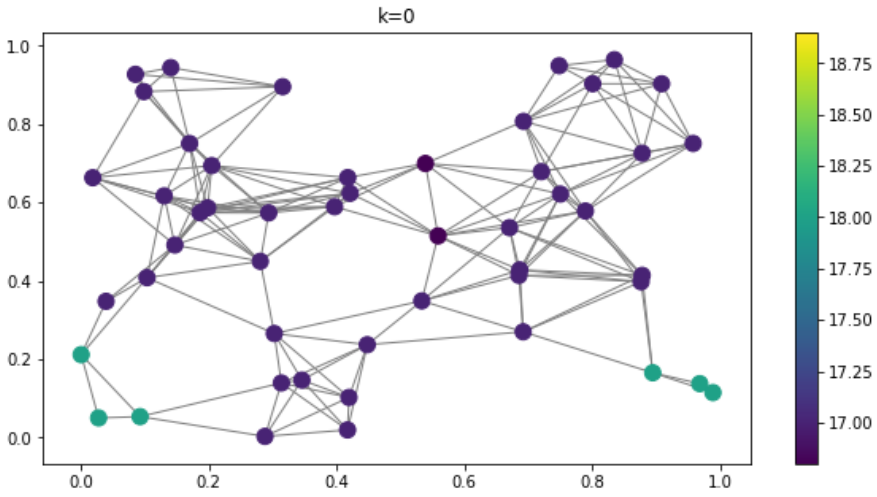


Figure A.1 – $b[n,k]$ at step $k=0$. The colors represent the values of $b[n,0]$ at different nodes ($K=9$, $\theta=2$, $\kappa=0.2$, $H(\lambda)=\frac{3}{3+5\lambda}$, $f_{in}[n]=a[n]^2+b[n]^2-1$).

A.2 Error Propagation

We now extend the experiments in subsection 3.5.2 to do an analysis of the error propagation for illustration purpose. Three graphs are generated in the same way as in subsection 3.5.1. The same input graph signal is filtered by distributively applying the same low-pass filter of Eq. (3.36) to the signal, with its Chebyshev polynomial approximation of order $K = 17$. We fix $\theta = 2$ and vary κ. When κ is small, the resulting graph is sparse. Thus, the three generated graphs have the same set of nodes but have different sparsity values, according to the chosen value of κ. Figs. A.2, A.3 and A.4 show the three graphs for the values of $\kappa = 0.18$, 0.25 and 0.3 respectively.

We are interested in observing how the individual error $\epsilon_0[n]$, at a specific value of n and generated in the first iteration step $k = 0$, is propagated through the network. To achieve this, we perform an experiment where all the other nodes send their messages without quantization (that is, as the perfect representation of their true value) and node n only quantizes at step $k = 0$, while for the next values of k it sends the messages unquantized. By doing this we can assure that all observed errors in the distributed signal processing stem from $\epsilon_0[n]$ and its propagation. The bounded scheme proposed in this chapter is used for all cases.

The chosen node n is highlighted within an orange circle. The colors represent the absolute difference between $z_{17}[n]$ in an unquantized processing (true value) and its value in the experiment mentioned where only $z_0[n]$ is quantized but all the other values (different k's or different n's) are not. We choose to represent this difference at k=17 since it is the filtering final step and we can thus see the accumulated propagation of $\epsilon_0[n]$. The difference is plotted in log scale.

We can see in Fig. A.2 that the node n is only connected to another node and disconnected from the rest. Thus, $\epsilon_0[n]$ does not propagate to most of the nodes resulting in these unconnected nodes having no errors on $z_{17}[n]$. On the other hand, node n and its neighbor present very high error, which suggests that the error was amplified among them. When we compare to Figs. A.3 and A.4, where the node n is connected to the rest of the network, $\epsilon_0[n]$ propagates to the other nodes and thus the value of the error at n drops considerably. When we consider the last Figure in particular, we notice that there is barely any error in any node. The global MSE for each case is $6.97e-07$ for $\kappa = 0.18$, $1.10e-07$ for $\kappa = 0.25$ and $0.47e-7$ for $\kappa = 0.3$, respectively. Notice that Fig. A.2 has the highest eccentricity for node n, which seems to suggest that nodes with high eccentricity, that is, more isolated nodes, tend to have their errors amplified, whereas the ones that are more central tend to have their errors dissolved into the network. This is coherent to what we previously observed in Section 3.5.2.

A.3 Variance of the bit allocation for binomial distribution

We now extend the experiments in subsection 3.5.2 to analyse the variance of the bit allocation for binomial distribution. Another 300 graphs were generated considering node degrees that

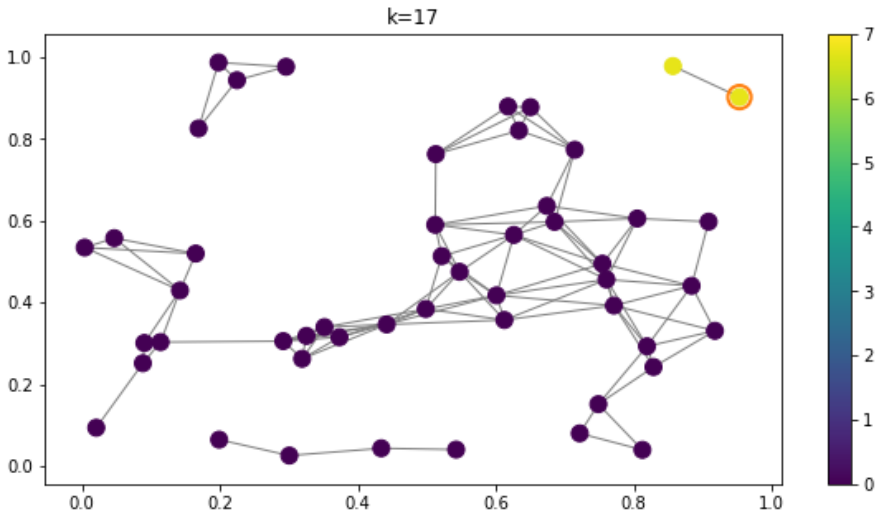


Figure A.2 – Graph built with $\kappa = 0.18$. The colors represent the absolute difference between $z_{17}[n]$ in an unquantized processing (true value) and its value in the experiment where only $z_0[n]$ is quantized but all the other values (different ks or different n's) are not. The chosen node n is highlighted within an orange circle. The global MSE is $6.97e-07$.

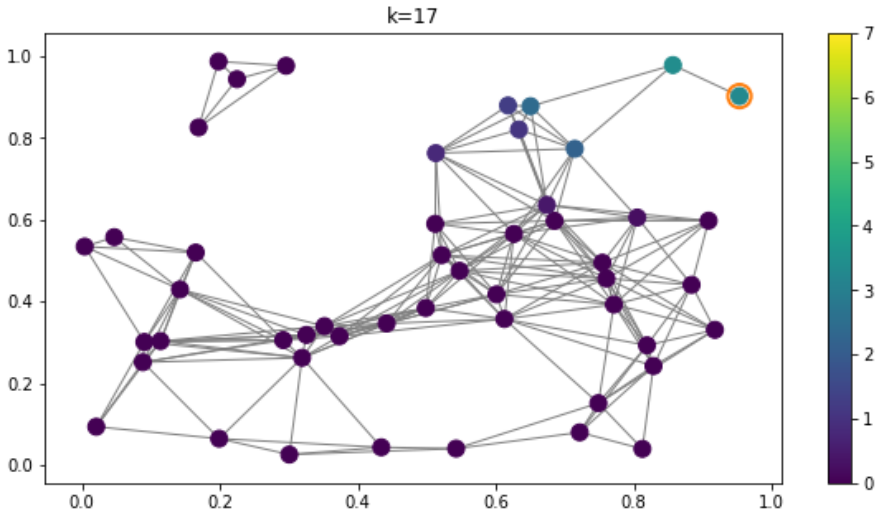


Figure A.3 – Graph built with $\kappa = 0.25$. The colors represent the absolute difference between $z_{17}[n]$ in an unquantized processing (true value) and its value in the experiment where only $z_0[n]$ is quantized but all the other values (different ks or different n's) are not. The chosen node n is highlighted within an orange circle. The global MSE is $1.10e-07$.

follow a binomial distribution. This distribution was chosen was because the mean and variance of this distribution can be chosen independently (as long as the variance is below the mean). The graphs were chosen with $N = 50$ nodes. The same graph signal as in Subsection 3.5.1 is generated and filtered with the same filter of Eq. (3.36) with polynomial approximation

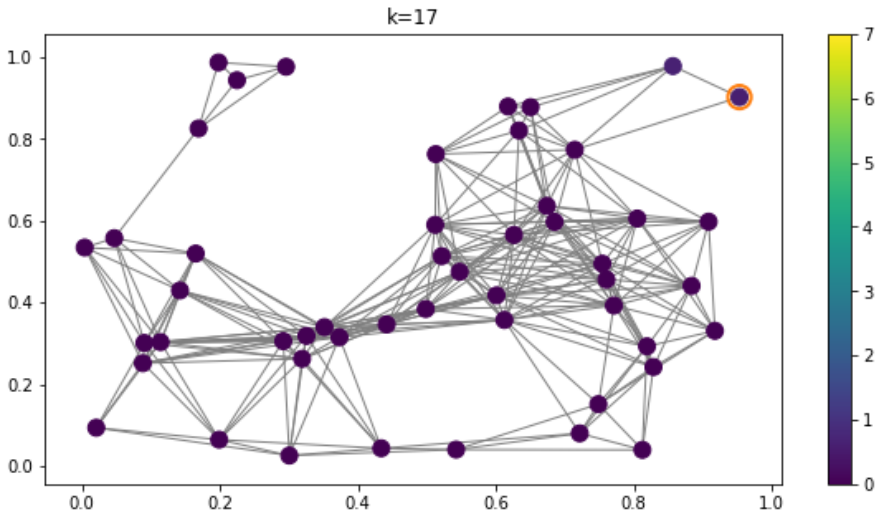


Figure A.4 – Graph built with $\kappa = 0.3$. The colors represent the absolute difference between $z_{17}[n]$ in an unquantized processing (true value) and its value in the experiment where only $z_0[n]$ is quantized but all the other values (different ks or different n's) are not. The chosen node n is highlighted within an orange circle. The global MSE is $0.47e-7$.

order $K = 9$. We use Eq. (3.33) for computing $b[n,k]$.

In Fig. A.5, we can see the relationship between the degrees' variance and mean with the variance of $b[n,k]$ with respect to n ($b[n,k]$ is averaged with respect to k). Each dot corresponds to the experiment resulted with the use of a different graph. We can notice that the results of Fig. A.5 are similar to those of Fig. 3.11, which means our observations are consistent for different types of graphs. That is, regular and dense graphs tend to have a more uniform bit distribution. Conversely, it means that when the graph is both sparser and irregular, it will require a more irregular bit distribution hence optimal bit allocation is important.

In Fig. A.6, we can further see the relationship with the variance of the nodes' eccentricities. In orange, we have graphs generated from binomial distribution node degrees and in blue we have Gaussian kernel generated graphs. Here, the same tendency is again noticed for both types of graphs.

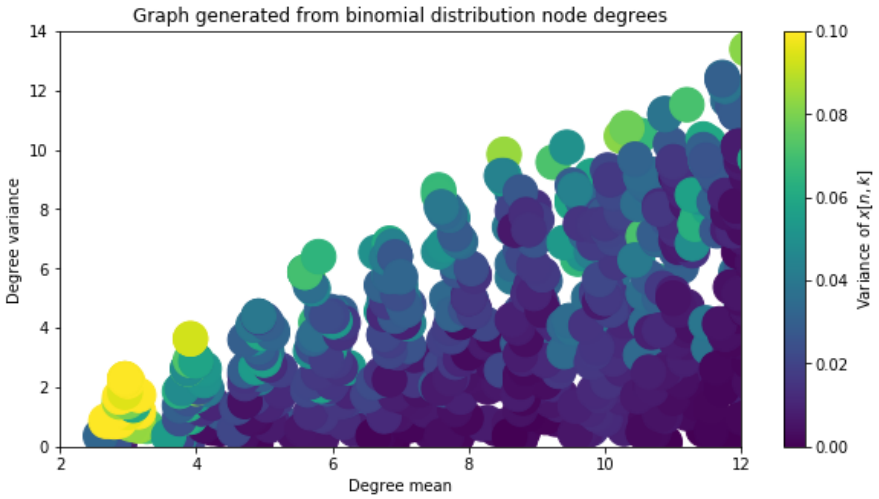


Figure A.5 – Relationship between the degrees' variance and mean with the variance of $b[n,k]$ for graphs generated with node degrees following a binomial distribution. The colors represent the variance of $b[n,k]$ along n and k. Each color dot represents a different graph.

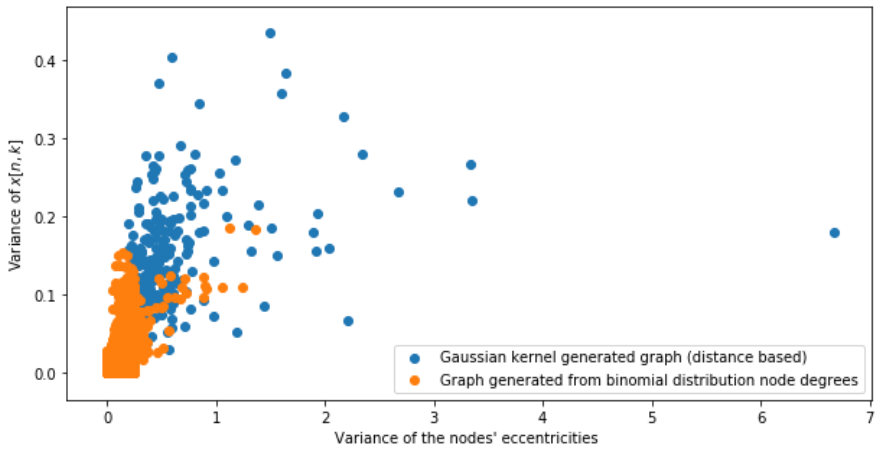


Figure A.6 – Variance of $b[n,k]$ versus variance of the nodes' eccentricities for two different type of graphs. In orange, we have graphs generated from binomial distribution node degrees and in blue we have Gaussian kernel generated graphs.

B Appendix of Chapter 4

B.1 Quantization for Graph Filter

In this section, we develop the quantization analysis for graph filters, for a general graph filter shift operator. We isolate the total quantization error caused by the accumulated effects of the individual quantization errors sent at every message. Additionally, we develop an expression of the MSE as a function of the bit allocation.

By taking into account the quantization errors that accumulate through all iterations of the distributed processing task, the filtered signal can be written as

$$\boldsymbol{H}^i(\boldsymbol{S})\boldsymbol{x} = \sum_{k=0}^{K} h_k^i \boldsymbol{S}^k \boldsymbol{x} + \sum_{k=0}^{K-1} \left[\sum_{j=1}^{K-k} h_{k+j}^i \boldsymbol{S}^j \right] \boldsymbol{\epsilon}_k, \tag{B.1}$$

as opposed to Eq. (4.1) for the perfect settings. We denote for each specific filter $\boldsymbol{H}^i(\boldsymbol{S})$ the coefficients h_k^i in Eq. (4.1).

Since the first term of the above expression of Eq. (B.1), $\sum_{k=0}^{K} h_k^i \boldsymbol{S}^k \boldsymbol{x}$, is the filtered signal in a setting without quantization, we can define the second term,

$$\boldsymbol{Q}^i = \sum_{k=0}^{K-1} \left[\sum_{j=1}^{K-k} h_{k+j}^i \boldsymbol{S}^j \right] \boldsymbol{\epsilon}_k, \tag{B.2}$$

as the total error caused by the accumulated effects of the quantization errors.

At iteration k, the maximum value (in an absolute sense) of the messages to be transmitted is $||\boldsymbol{S}^k \boldsymbol{x}||_\infty$. If the eigenvalues of the shift operator are bounded in $[-1, 1]$, the range of the values being transmitted at step k will not surpass the range of the original signal.

For the sake of clarity, we now write

$$\boldsymbol{W}_k^i = \sum_{j=1}^{K-k} h_{k+j}^i \boldsymbol{S}^j \tag{B.3}$$

and

$$F_j^i = \left(\boldsymbol{W}_k^T \boldsymbol{W}_k\right)[n,n] = \sum_{j=1}^{K-k}\sum_{p=1}^{K-k} h_{k+j}^i h_{k+p}^i \boldsymbol{S}^{j+p}[n,n]. \tag{B.4}$$

Hence,

$$\boldsymbol{Q}^i = \sum_{k=0}^{K-1} \boldsymbol{W}_k^i \boldsymbol{\epsilon}_k, \tag{B.5}$$

and we can finally calculate the mean squared error

$$E\left\{\| \boldsymbol{Q}^i \|^2\right\} = \sum_{k=0}^{K-1}\sum_{l=0}^{K-1} E[\boldsymbol{\epsilon}_k^T (\boldsymbol{W}_k^i)^T \boldsymbol{W}_l^i \boldsymbol{\epsilon}_l]. \tag{B.6}$$

Hence the expected value of the crossed terms in Eq. (B.6) is zero and we finally obtain the expected value of the total mean squared error as

$$E\left\{\| \boldsymbol{Q}^i \|^2\right\} = \sum_{k=0}^{K-1}\sum_{n=0}^{N-1} \left((\boldsymbol{W}_k^i)^T \boldsymbol{W}_k^i\right)[n,n] E\{\boldsymbol{\epsilon}_k[n]^2\}, \tag{B.7}$$

which is equivalent to

$$E\left\{\| \boldsymbol{Q}^i \|^2\right\} = \sum_{k=0}^{K-1}\sum_{n=0}^{N-1} F_j^i E\{\epsilon_j^2\}, \tag{B.8}$$

with Eq. (2.16), the MSE finally reads

$$E\left\{\| \boldsymbol{Q}^i \|^2\right\} = \frac{(\Gamma^i)^2}{12} \sum_{k=0}^{K-1}\sum_{n=0}^{N-1} F_j^i 2^{-2b_j^i}. \tag{B.9}$$

This result is used in Subsection 4.3.1.

B.2 Quantization Step in Molène Experiments

We now extend the experiments in Subsection 4.5.1 to observe the quantization step on the Molène dataset. In Fig. B.1, we plot the quantization step under optimal bit allocation for the 5 models trained on the Molène dataset, a non-synthetic dataset supported on a graph of N=355 from Subsection 4.5.1. Similarly to Fig. 4.5, we see a tendency of a lower quantization step (i.e., a higher relative bit allocation) in the second layer, while the quantization step is higher in the first and third layers. This means the messages sent in the middle layers are more important than the ones sent in the first and last layers, from a quantization perspective.

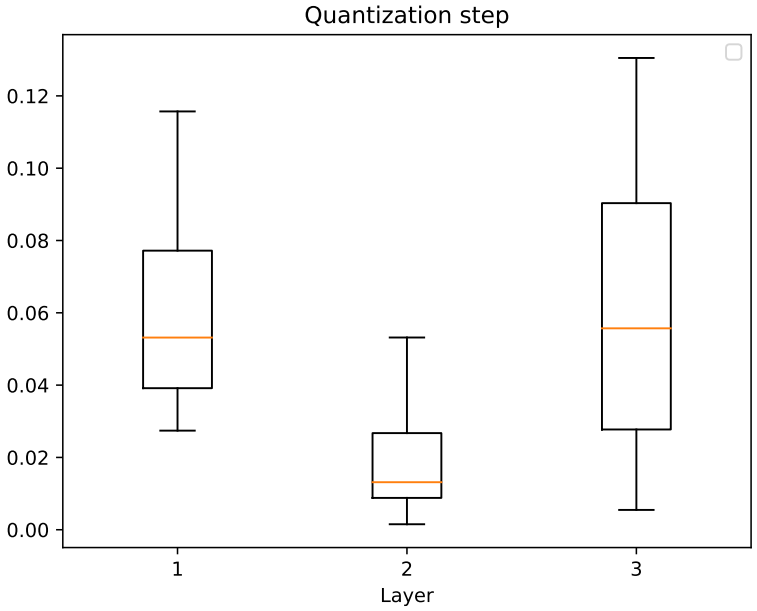


Figure B.1 – Quantization step per layer using the optimal allocation scheme for Molène experiments in Subsection 4.5.1. Lower values mean a higher relative bit allocation. Each datapoint represents the average quantization step of a filter in that layer of a model. Filters of 5 models are represented in the box plot, with 1, 2 and 2 filters in layers 1, 2 and 3 respectively. Bit budget of 8 bits/message on average.

B.3 Source Localization Examples

To better illustrate the task of source localization (Section 4.5.2), in Fig. B.2 we visualize three examples of a source signal and its corresponding diffused signal, on three different supporting graphs. The source signal, $\mathbf{y}^{\text{original}}$, is diffused in the graph for $k = 3$ steps, as $\mathbf{x} = \mathbf{S}^k\mathbf{y}^{\text{original}}$. The diffused signal $\mathbf{x}$ is the input to the GCNN, which outputs a prediction $\boldsymbol{y} = \sigma(f_{\boldsymbol{\theta}}(\boldsymbol{x}))$ on the binary class of each node, either originally active (1) or inactive (0).

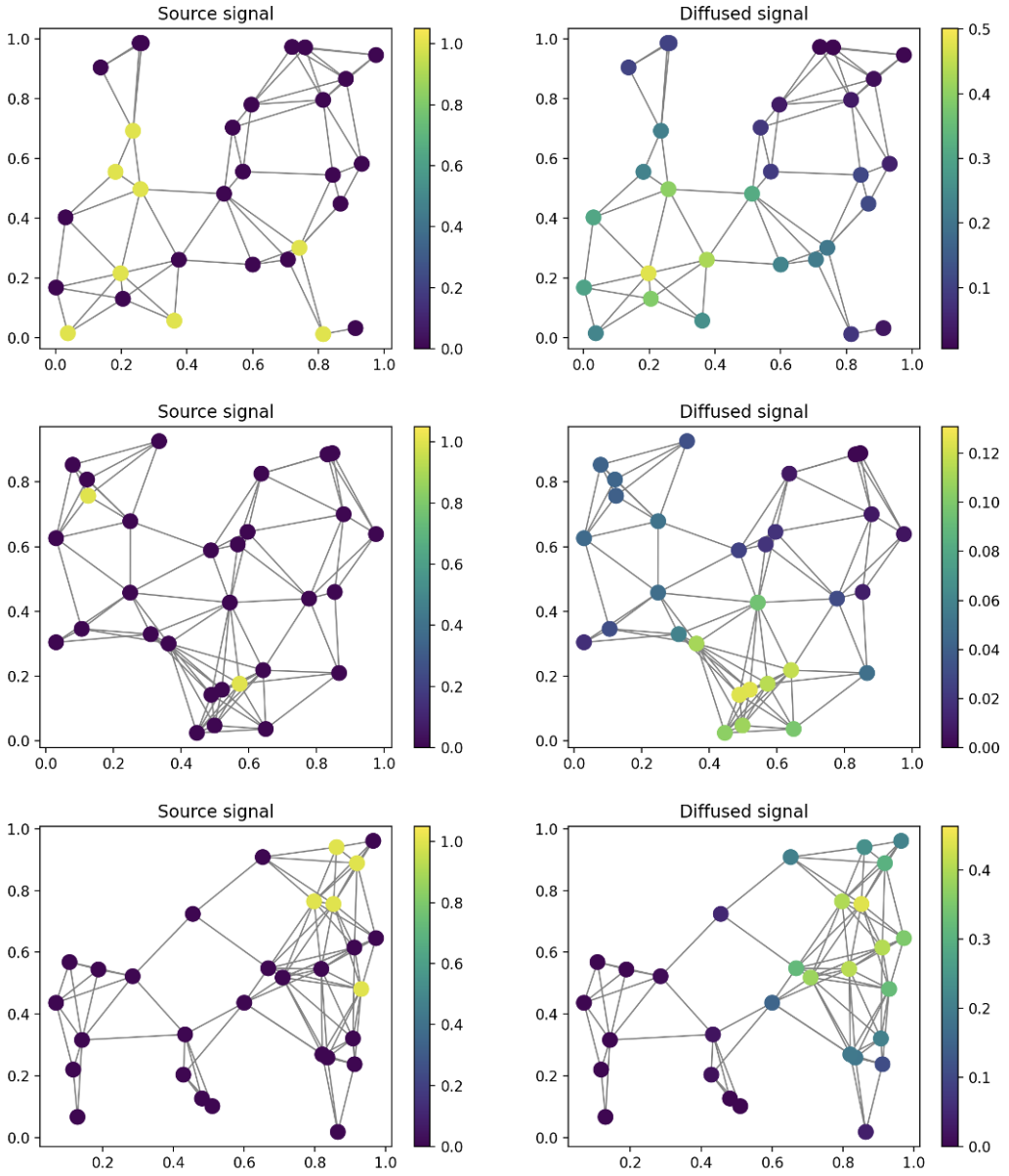


Figure B.2 – Three example samples for the source localization task, supported on three different graphs of $N = 30$. A GCNN predicts for each node whether it was active or not in the source signal, given the diffused signal, as shown in Section 4.5.2.

B.4 Tables for Standard Deviations

In order to illustrate the standard deviations on experiments performed on Section 4.5, we make use of tables. Table B.1 is linked to Fig. 4.3, table B.2 is linked to Fig. 4.4a, table B.3 is linked to Fig. 4.4b, and finally, table B.4 is linked to Fig. 4.10.

Avg. Bits/message	8 bits	9 bits	10 bits
Uniform	2.8e−4 ± 1.9e−4	7.0e−5 ± 4.8e−5	1.7e−5 ± 1.2e−5
Optimal	**9.2e−5** ± 5.7e−5	**2.3e−5** ± 1.3e−5	**5.8e−6** ± 3.5e−6
Graph-level	1.8e−4 ± 1.2e−4	4.6e−5 ± 3.2e−5	1.1e−5 ± 8.1e−6
Layer 1 (+1, 0, -1)	6.7e−4 ± 4.3e−4	1.6e−4 ± 1.0e−4	4.2e−5 ± 2.8e−5
Layer 3 (-1, 0, +1)	2.3e−4 ± 1.2e−4	5.8e−5 ± 3.e−5	1.4e−5 ± 7.7e−6
Range-based	3.2e−4 ± 2.5e−4	8.1e−5 ± 6.2e−5	2.0e−5 ± 1.5e−5

Table B.1 – Quantization MSE on the Molène dataset for distributed denoising, supporting graph of N=355 nodes. Average of 25 runs (5 models and 5 runs per model). **Lowest** and second-lowest highlighted. Table for Fig. 4.3

Avg. Bits/message	8 bits	9 bits	10 bits
Uniform:	1.7e−5 ± 4.6e−6	4.4e−6 ± 1.2e−6	1.1e−6 ± 3.0e−7
Optimal:	**5.4e−6** ± 1.6e−6	**1.2e−6** ± 4.1e−7	**3.2e−7** ± 9.0e−8
Graph-level:	1.2e−5 ± 3.7e−6	3.0e−6 ± 1.0e−6	7.4e−7 ± 2.5e−7
Layer 1 (+1, 0, -1):	2.5e−5 ± 7.8e−6	6.2e−6 ± 1.8e−6	1.5e−6 ± 4.6e−7
Layer 3 (-1, 0, +1):	4.9e−5 ± 1.8e−5	1.2e−5 ± 4.7e−6	3.1e−6 ± 1.2e−6
Range-based:	1.2e−5 ± 2.4e−6	3.1e−6 ± 6.2e−7	7.8e−7 ± 1.3e−7

Table B.2 – Quantization MSE for different bit allocation schemes on the distributed denoising task. Average across 5 supporting graphs of N=30 nodes, error bars represent standard deviation. Results for each graph are averaged across 20 models of L=2. **Lowest** and second-lowest highlighted. Table for Fig. 4.4a

Avg. Bits/message	8 bits	9 bits	10 bits
Uniform:	5.5e−5 ± 9.6e−6	1.4e−5 ± 2.5e−6	3.4e−6 ± 6.5e−7
Optimal:	**2.0e−5** ± 9.6e−6	**5.0e−6** ± 3.4e−6	**1.4e−6** ± 6.6e−6
Graph-level:	3.9e−5 ± 8.8e−6	9.8e−6 ± 2.1e−6	2.4e−6 ± 5.1e−7
Layer 1 (+1, 0, -1):	9.2e−5 ± 2.2e−5	2.3e−5 ± 5.3e−6	5.8e−6 ± 1.2e−6
Layer 3 (-1, 0, +1):	8.7e−5 ± 2.1e−5	2.1e−5 ± 5.2e−6	5.5e−6 ± 1.4e−6
Range-based:	4.8e−5 ± 8.8e−6	1.2e−4 ± 2.3e−6	3.0e−6 ± 5.8e−7

Table B.3 – Quantization MSE for different bit allocation schemes on the distributed denoising task. Average across 5 supporting graphs of N=30 nodes, error bars represent standard deviation. Results for each graph are averaged across 20 models of L=4. **Lowest** and second-lowest highlighted. Table for Fig. 4.4b

Avg. Bits/message	8 bits	9 bits	10 bits
Uniform	1.9e−3 ± 1.6e−3	4.9e−4 ± 4.1e−4	1.2e−4 ± 9.9e−5
Optimal	**5.4e−4** ± 7.6e−4	**1.7e−4** ± 2.6e−4	**2.3e−5** ± 1.5e−5
Graph-level	<u>9.6e−4</u> ± 8.9e−4	<u>2.2e−4</u> ± 2.1e−4	<u>5.7e−5</u> ± 5.1e−5
Layer 1 $(+1, 0, -1)$	2.0e−3 ± 1.3e−3	5.2e−4 ± 3.4e−4	1.3e−4 ± 8.7e−5
Layer 3 $(-1, 0, +1)$	4.8e−3 ± 4.7e−3	1.1e−3 ± 1.0e−4	2.9e−4 ± 2.8e−4
Range-based	1.4e−3 ± 1.0e−3	3.4e−4 ± 2.2e−4	8.9e−5 ± 5.7e−5

Table B.4 – Quantization MSE for different bit allocation schemes on a source localization task. Mean and standard deviation across 5 supporting graphs of N=30 nodes. Results for each graph are averaged across 20 models. **Lowest** and <u>second-lowest</u> highlighted.Table for Fig. 4.10

Bibliography

[1] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José MF Moura, and Pierre Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.

[2] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[3] David I Shuman, Pierre Vandergheynst, Daniel Kressner, and Pascal Frossard, "Distributed Signal Processing via Chebyshev Polynomial Approximation," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 4, pp. 736–751, Dec. 2018.

[4] Aliaksei Sandryhaila, Soummya Kar, and José MF Moura, "Finite-time distributed consensus through graph filters," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 1080–1084.

[5] Sam Safavi and Usman A Khan, "Revisiting finite-time distributed algorithms via successive nulling of eigenvalues," *IEEE Signal Processing Letters*, vol. 22, no. 1, pp. 54–57, 2015.

[6] Elvin Isufi, Andreas Loukas, Andrea Simonetto, and Geert Leus, "Autoregressive moving average graph filtering," *IEEE Transactions on Signal Processing*, vol. 65, no. 2, pp. 274–288, 2017.

[7] Santiago Segarra, Antonio G Marques, and Alejandro Ribeiro, "Optimal graph-filter design and applications to distributed linear network operators," *IEEE Transactions on Signal Processing*, vol. 65, no. 15, pp. 4117–4131, 2017.

[8] Xuesong Shi, Hui Feng, Muyuan Zhai, Tao Yang, and Bo Hu, "Infinite impulse response graph filters in wireless sensor networks," *IEEE Signal Processing Letters*, vol. 22, no. 8, pp. 1113–1117, 2015.

[9] Aikaterini Protogerou, Stavros Papadopoulos, Anastasios Drosou, Dimitrios Tzovaras, and Ioannis Refanidis, "A graph neural network method for distributed anomaly detection in IoT," *Evolving Systems*, vol. 12, no. 1, pp. 19–36, Mar. 2021.

[10] Bianca Iancu, Luana Ruiz, Alejandro Ribeiro, and Elvin Isufi, "Graph-adaptive activation functions for graph neural networks," in *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2020, pp. 1–6.

[11] Bianca Iancu and Elvin Isufi, "Towards Finite-Time Consensus with Graph Convolutional Neural Networks," in *2020 28th European Signal Processing Conference (EUSIPCO)*. IEEE, 2021, pp. 2145–2149.

[12] Fernando Gama, Qingbiao Li, Ekaterina Tolstaya, Amanda Prorok, and Alejandro Ribeiro, "Decentralized Control with Graph Neural Networks," *arXiv:2012.14906 [cs, eess]*, June 2021.

[13] Fernando Gama and Somayeh Sojoudi, "Distributed linear-quadratic control with graph neural networks," *Signal Processing*, vol. 196, pp. 108506, 2022.

[14] Fernando Gama, Ekaterina Tolstaya, and Alejandro Ribeiro, "Graph neural networks for decentralized controllers," in *2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 5260–5264.

[15] Zhan Gao, Fernando Gama, and Alejandro Ribeiro, "Wide and deep graph neural network with distributed online learning," *IEEE Transactions on Signal Processing*, vol. 70, pp. 3862–3877, 2022.

[16] Qingbiao Li, Fernando Gama, Alejandro Ribeiro, and Amanda Prorok, "Graph neural networks for decentralized multi-robot path planning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 11785–11792.

[17] Qingbiao Li, Weizhe Lin, Zhe Liu, and Amanda Prorok, "Message-aware graph attention networks for large-scale multi-robot path planning," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5533–5540, 2021.

[18] Ekaterina Tolstaya, Fernando Gama, James Paulos, George Pappas, Vijay Kumar, and Alejandro Ribeiro, "Learning decentralized controllers for robot swarms with graph neural networks," in *Conference on robot learning*. PMLR, 2020, pp. 671–682.

[19] Zhiyang Wang, Mark Eisen, and Alejandro Ribeiro, "Learning decentralized wireless resource allocations with graph neural networks," *IEEE Transactions on Signal Processing*, vol. 70, pp. 1850–1863, 2022.

[20] Zhongyuan Zhao, Gunjan Verma, Chirag Rao, Ananthram Swami, and Santiago Segarra, "Distributed scheduling using graph neural networks," in *2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 4720–4724.

[21] Zhongyuan Zhao, Gunjan Verma, Chirag Rao, Ananthram Swami, and Santiago Segarra, "Link scheduling using graph neural networks," *IEEE Transactions on Wireless Communications*, 2022.

[22] Zhan Gao, Elvin Isufi, and Alejandro Ribeiro, "Stochastic graph neural networks," *IEEE Transactions on Signal Processing*, vol. 69, pp. 4428–4443, 2021.

[23] S. Scardapane, I. Spinelli, and P. D. Lorenzo, "Distributed Training of Graph Convolutional Networks," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 7, pp. 87–100, 2021.

[24] Alberto Natali, Mario Coutino, and Geert Leus, "Topology-aware joint graph filter and edge weight identification for network processes," in *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2020, pp. 1–6.

[25] Coutiño Minguez, E Isufi, Takanori Maehara, and GJT Leus, "State-space network topology identification from partial observations," *IEEE Transactions on Signal and Information Processing over Networks*, 2020.

[26] T Mitchell Roddenberry, Madeline Navarro, and Santiago Segarra, "Network topology inference with graphon spectral penalties," *arXiv preprint arXiv:2010.07872*, 2020.

[27] Dorina Thanou, Xiaowen Dong, Daniel Kressner, and Pascal Frossard, "Learning heat diffusion graphs," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 3, no. 3, pp. 484–499, 2017.

[28] Carlos Lassance, Vincent Gripon, and Gonzalo Mateos, "Graph topology inference benchmarks for machine learning," in *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2020, pp. 1–6.

[29] Vincenzo Matta, Augusto Santos, and Ali H Sayed, "Graph learning over partially observed diffusion networks: Role of degree concentration," *arXiv preprint arXiv:1904.02963*, 2019.

[30] Matthias Minder, Zahra Farsijani, Dhruti Shah, Mireille El Gheche, and Pascal Frossard, "Figlearn: Filter and graph learning using optimal transport," *arXiv preprint arXiv:2010.15457*, 2020.

[31] Jiaxi Ying, José Vinícius de M Cardoso, and Daniel P Palomar, "Does the l1-norm learn a sparse graph under laplacian constrained graphical models?," *arXiv preprint arXiv:2006.14925*, 2020.

[32] Xiaowen Dong, Dorina Thanou, Michael Rabbat, and Pascal Frossard, "Learning graphs from data: A signal representation perspective," *IEEE Signal Processing Magazine*, vol. 36, no. 3, pp. 44–63, 2019.

[33] Gonzalo Mateos, Santiago Segarra, Antonio G Marques, and Alejandro Ribeiro, "Connecting the dots: Identifying network structure via graph signal processing," *IEEE Signal Processing Magazine*, vol. 36, no. 3, pp. 16–43, 2019.

[34] Xiaowen Dong, Dorina Thanou, Pascal Frossard, and Pierre Vandergheynst, "Learning laplacian matrix in smooth graph signal representations," *IEEE Transactions on Signal Processing*, vol. 64, no. 23, pp. 6160–6173, 2016.

[35] Pierre Humbert, Batiste Le Bars, Laurent Oudre, Argyris Kalogeratos, and Nicolas Vayatis, "Learning laplacian matrix from graph signals with sparse spectral representation," *Journal of Machine Learning Research*, 2021.

[36] Vassilis Kalofolias and Nathanaël Perraudin, "Large scale graph learning from smooth signals," *arXiv preprint arXiv:1710.05654*, 2017.

[37] Vassilis Kalofolias, "How to learn a graph from smooth signals," in *2016 Artificial Intelligence and Statistics (AI Stats)*. PMLR, pp. 920–929.

[38] Stefania Sardellitti, Sergio Barbarossa, and Paolo Di Lorenzo, "Graph topology inference based on transform learning," in *2016 IEEE global conference on signal and information processing (GlobalSIP)*. IEEE, 2016, pp. 356–360.

[39] Dorina Thanou and Pascal Frossard, "Learning of robust spectral graph dictionaries for distributed processing," *EURASIP Journal on Advances in Signal Processing*, vol. 2018, no. 1, pp. 67, 2018.

[40] B. Widrow, I. Kollar, and Ming-Chang Liu, "Statistical theory of quantization," *IEEE Transactions on Instrumentation and Measurement*, vol. 45, no. 2, pp. 353–361, Apr. 1996.

[41] Lin Xiao, M. Johansson, H. Hindi, S. Boyd, and A. Goldsmith, "Joint optimization of communication rates and linear systems," *IEEE Transactions on Automatic Control*, vol. 48, no. 1, pp. 148–153, Jan. 2003.

[42] Aliaksei Sandryhaila and José MF Moura, "Discrete signal processing on graphs," *IEEE Transactions on Signal Processing*, vol. 61, no. 7, pp. 1644–1656, 2013.

[43] Jiani Liu, Elvin Isufi, and Geert Leus, "Filter design for autoregressive moving average graph filters," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 5, no. 1, pp. 47–60, 2018.

[44] Luiz F. O. Chamon and Alejandro Ribeiro, "Finite-precision effects on graph filters," in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Montreal, QC, Nov. 2017, pp. 603–607, IEEE.

[45] Shengyu Zhu, Mingyi Hong, and Biao Chen, "Quantized consensus admm for multi-agent distributed optimization," in *IEEE ICASSP*, 2016, pp. 4134–4138.

[46] Shengyu Zhu and Biao Chen, "Quantized consensus by the admm: probabilistic versus deterministic quantizers," *IEEE Transactions on Signal Processing*, vol. 64, no. 7, pp. 1700–1713, 2016.

[47] Huaqing Li, Shuai Liu, Yeng Chai Soh, and Lihua Xie, "Event-triggered communication and data rate constraint for distributed optimization of multiagent systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, , no. 99, pp. 1–12, 2017.

[48] Jueyou Li, Guo Chen, Zhiyou Wu, and Xing He, "Distributed subgradient method for multi-agent optimization with quantized communication," *Mathematical Methods in the Applied Sciences*, vol. 40, no. 4, pp. 1201–1213, 2017.

[49] Dorina Thanou, Effrosyni Kokiopoulou, Ye Pu, and Pascal Frossard, "Distributed average consensus with quantization refinement," *IEEE Transactions on Signal Processing*, vol. 61, no. 1, pp. 194–205, 2013.

[50] L Ben Saad, Baltasar Beferull-Lozano, and Elvin Isufi, "Quantization analysis and robust design for distributed graph filters," *IEEE Transactions on Signal Processing*, vol. 70, pp. 643–658, 2021.

[51] Leila Ben Saad, Elvin Isufi, and Baltasar Beferull-Lozano, "Graph Filtering with Quantization over Random Time-varying Graphs," in *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Nov. 2019, pp. 1–5.

[52] Carl D Meyer, *Matrix analysis and applied linear algebra*, vol. 71, Siam, 2000.

[53] Stephen Boyd and Lieven Vandenberghe, *Convex optimization*, Cambridge university press, 2004.

[54] Adrian Segall, "Bit allocation and encoding for vector sources," *IEEE Transactions on Information Theory*, vol. 22, no. 2, pp. 162–169, 1976.

[55] Morgan Levy, "Curated rain and flow data for the brazilian rainforest-savann transition zone," *HydroShare*, Mar 2017.

[56] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2020.

[57] Fernando Gama, Elvin Isufi, Geert Leus, and Alejandro Ribeiro, "Graphs, convolutions, and neural networks: From graph filters to graph neural networks," *IEEE Signal Processing Magazine*, vol. 37, no. 6, pp. 128–138, 2020.

[58] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[59] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[60] Chaim Baskin, Natan Liss, Eli Schwartz, Evgenii Zheltonozhskii, Raja Giryes, Alex M. Bronstein, and Avi Mendelson, "UNIQ: Uniform Noise Injection for Non-Uniform Quantization of Neural Networks," *ACM Transactions on Computer Systems*, vol. 37, no. 1-4, pp. 1–15, June 2021.

[61] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," *arXiv:2103.13630 [cs]*, Apr. 2021.

[62] Hongyang Liu, Sara Elkerdawy, Nilanjan Ray, and Mostafa Elhoushi, "Layer Importance Estimation With Imprinting for Neural Network Quantization," in *2021 Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2408–2417.

[63] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort, "A White Paper on Neural Network Quantization," *arXiv:2106.08295 [cs]*, June 2021.

[64] Shyam A. Tailor, Javier Fernandez-Marques, and Nicholas D. Lane, "Degree-Quant: Quantization-Aware Training for Graph Neural Networks," *arXiv:2008.05000 [cs, stat]*, Mar. 2021.

[65] Yiren Zhao, Duo Wang, Daniel Bates, Robert Mullins, Mateja Jamnik, and Pietro Lio, "Learned Low Precision Graph Neural Networks," *arXiv:2009.09232 [cs]*, Sept. 2020.

[66] Isabela Cunha Maia Nobre and Pascal Frossard, "Optimized Quantization in Distributed Graph Signal Filtering," *arXiv:1909.12725 [eess]*, Sept. 2019.

[67] Pei Li, Nir Shlezinger, Haiyang Zhang, Baoyun Wang, and Yonina C. Eldar, "Task-Based Graph Signal Compression," *arXiv:2110.12387 [eess]*, Oct. 2021.

[68] Mengyuan Lee, Guanding Yu, and Huaiyu Dai, "Decentralized inference with graph neural networks in wireless communication systems," *IEEE Transactions on Mobile Computing*, 2021.

[69] Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard, "Adaptive quantization for deep neural network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, vol. 32.

[70] Wang Zhe, Jie Lin, Vijay Chandrasekhar, and Bernd Girod, "Optimizing the Bit Allocation for Compression of Weights and Activations of Deep Neural Networks," in *2019 IEEE International Conference on Image Processing (ICIP)*, Sept. 2019, pp. 3826–3830.

[71] Vincenzo Matta, Augusto Santos, and Ali H Sayed, "Graph learning under partial observability," *Proceedings of the IEEE*, vol. 108, no. 11, pp. 2049–2066, 2020.

[72] Vincenzo Matta and Ali H Sayed, "Consistent tomography under partial observations over adaptive networks," *IEEE Transactions on Information Theory*, vol. 65, no. 1, pp. 622–646, 2018.

[73] André LF De Almeida, Alain Y Kibangou, Sebastian Miron, and Daniel C Araújo, "Joint data and connection topology recovery in collaborative wireless sensor networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 5303–5307.

[74] Vincenzo Matta, Virginia Bordignon, Augusto Santos, and Ali H Sayed, "Interplay between topology and social learning over weak graphs," *IEEE Open Journal of Signal Processing*, vol. 1, pp. 99–119, 2020.

[75] Mircea Moscu, Roula Nassif, Fei Hua, and Cédric Richard, "Learning causal networks topology from streaming graph signals," in *2019 27th European Signal Processing Conference (EUSIPCO)*. IEEE, 2019, pp. 1–5.

[76] Mircea Moscu, Ricardo Borsoi, and Cédric Richard, "Online graph topology inference with kernels for brain connectivity estimation," in *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 1200–1204.

[77] Bennett Fox, "Discrete Optimization via Marginal Analysis," *Management Science*, vol. 13, no. 3, pp. 210–216, 1966.

ISABELA CUNHA MAIA NOBRE

EPFL LTS4 Station 11 CH-1015 Lausanne

+41 76 6422303 isabela.nobre@epfl.ch

Education

PhD in Electrical Engineering
École Polytechnique Fédérale de Lausanne (EPFL)
GPA 5.5/6; 2018 - 2022
Title: Quantization for Distributed Processing and Learning of Structured Data.
Advisor: Prof. Pascal Frossard
Expertise: Graph Signal Processing, Graph Neural Networks, Machine Learning, Distributed Systems and Communication Systems.

M.S. Electrical Engineering, Communication Systems
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
GPA 9.4/10; 2015 - 2017
Title: On the Protection of Fixed Service Receivers from the Interference Generated by Non-GSO Satellite Systems Operating in the 3.7-4.2 GHz Band.
Advisor: Prof. José Mauro Pedro Fortes.

B.S. Electrical and Electronic Engineering
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
GPA 9.4/10; 2011 - 2015

Exchange Program, Electrical and Electronic Engineering
University of California, San Diego (UCSD)
GPA 4/4; Fall 2014

Experience

EPFL - LTS4
Research Internship in Signal Processing over Networks, 01/2018-06/2018
Work Description: Studied quantization error in distributed signal processing tasks over networks.

Neoenergia (Iberdrola)
Internship in the Power Systems Regulation Sector, 06/2014-08/2014
Work Description: Worked with the operational model of the Brazilian electrical power system and the procedures on energy commercialization and contracting.

PUC-Rio
Research Internship in Antenna Theory, 08/2012-07/2014
Work Description: Studied the control of electromagnetic modal propagation within a coaxial antenna in order to use it in a Space Division Multiple Access system.

Teaching Assistantship, 08/2011-12/2011
Work Description: Teaching assistant of the Department of Mathematics' course

Calculus of Several Variables. Taught at a class of approximately 40 students.

AC ROSA Engineering Consultancy
Internship in the Geotechnics Sector, 03/2012-06/2012
Work Description: Elaborated technical drawings related to geotechnical retaining wall projects with the software AutoCAD.

UFAL - LCCV
Research Internship in Computer Vision, 02/2010-12/2010
Work Description: : Developed (C) a player for viewing videos in 3D projection environments for oil and gas applications. Project funded by Petrobras.

Awards

- Awarded 3 incentive prizes (50% of tuition costs for the awarded year) from PUC-Rio for being one of the top 10 students out of 3000 in Engineering based on cumulative GPA. Awards obtained in the years 2012, 2013 and 2014;
- Won best project out of 30 students in the Introduction to Engineering class (2011) by successfully designing and building a slot car and a track;
- Selected among 868.000 participants nationwide to be part of the 5 students representing Brazil at the International Olympiad on Astronomy and Astrophysics and at the Latin-American Astronomy Olympiad in 2009 (high school level), based on theoretical and practical exams. Participated in the latter and won a silver medal;
- Won third place out of 5000 students nationwide in the II Brazilian Rocket Olympiad in 2008 (high school level) based on the range attained by a built rocket;
- National Scientific Olympiads (middle- and high- school level): 3 golden medals, 3 silver medals and 2 bronze medals (in the fields of Astronomy, Mathematics and Chemistry, from 2005 to 2009). On average, 350 thousand students nationwide participate in these competitions based on theoretical knowledge exams.

Grants

- PhD: 100% tuition + CHF 83.190,00 stipend - CAPES (2018- ongoing), based on leadership potential and academic success;
- M.S: 100% tuition + R$36.000,00 stipend - CNPq / PUC-Rio (2015-2017), based on academic performance;
- B.S: 50% tuition - PUC-Rio (2011-2015), based on entrance exam results (top 20 out of 1900);
- Study Abroad Scholarship: 100% tuition - UCSD (2014), based on academic performance.

Languages

Portuguese: Native, English: Fluent, French: Fluent, Spanish: Advanced, Italian: Basic, German: Basic