Graph Signal Processing: Sparse Representation and Applications

THÈSE N^{O} 6925 (2016)

PRÉSENTÉE LE 5 FÉVRIER 2016
À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR
LABORATOIRE DE TRAITEMENT DES SIGNAUX 4
PROGRAMME DOCTORAL EN GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Ntorina THANOU

acceptée sur proposition du jury:

Prof. D. N. A. Van De Ville, président du jury
Prof. P. Frossard, directeur de thèse
Dr Ph. Chou, rapporteur
Dr P. Borgnat, rapporteur
Prof. P. Vandergheynst, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2016

To my father

Acknowledgements

The work presented in this thesis is the result of a long journey that goes beyond the pages of the current manuscript. Doing a PhD is not only an academic challenge but also a personal development one. Thanks to many people, struggling with these challenges has been much more fun and exciting than I was initially expecting.

The biggest thanks I owe to my thesis advisor, Prof. Pascal Frossard, for his countless support during all these years that went far beyond what I could have ever expected from an advisor. His vision and his intuition were always there to challenge me with new research problems and encourage me to find my research path by pushing my limits a little further each time. His flexibility and kind personality are two important ingredients for making the LTS4 lab such a warm and pleasant working environment. I am extremely happy to have Pascal as my thesis advisor and I deeply appreciate his belief in me and my research.

I would like to thank my committee members Dr. Pierre Borgnat, Dr. Philip Chou, Prof. Pierre Vandergheynst, and the president Prof. Dimitri Van De Ville, for spending time reading my thesis and giving me valuable comments and suggestions. Their interesting and promising ideas made my exam a very nice and pleasant experience to remember.

During my PhD I had the opportunity to work closely with a number of people who contributed significantly to the success of this thesis. My deepest gratitude goes to Dr. Philip Chou for being a true mentor during my internship in Microsoft Research. I feel very fortunate to have worked with Phil, and get exposed to his way of thinking. Our inspiring discussions, followed always by a smile, made my stay in Seattle a unique experience. Special thanks goes to Dr. Xiaowen Dong for his enormous help during my thesis. His enthusiasm, and his persistency were always there to remember me that research is fun. I owe to Xiaowen an even huger thanks for being a wonderful friend and making me the honor to become my Chinese brother. I would like to thank also Prof. David Shuman for his help with significant parts of my thesis and his valuable advice. Special thanks to Dr. Effrosyni Kokiopoulou for her help in the beginning of my PhD and my “neighbor” Prof. Pierre Vandergheynst for our fruitful discussions on graphs, some of them even in the LTS corridors.

I feel extremely lucky to have found one of the best working environments one could ever hope for, known as “the best lab in the world”. Many thanks to the former and the current LTS4 lab mates Ana, David, Eirini, Elif, Francesca, Hyunggon, Hussein, Jacob, Laura, Luigi, Mattia, Nikos, Renata, Pinar, Seyed, Sofia, Stefano, Tamara, Thomas, Vijay, Xiaowen, Ye, Zafer. They have all contributed to make the life in the lab more fun and exciting. I would also like to thank all the people I have met in the LTS corridor (Alessandra, Alia, Anil, Anna, Elda, Eleni, Vasilis to name a few) and the interesting time we have spent together all these years. Many thanks also to Rosie

for all her administrative support and the efficient arrangements of all my trips.

Life and in particular PhD without friends is a mission impossible. I have been lucky enough to have a lot of old and new friends along this journey whose presence has made my life more complete. Many thanks, with all my heart, goes to my friend Giorgos who is always there for me to share the bad and the good moments. I thank him for being a true friend and making the distance between Netherlands and Lausanne a piece of cake. I would also like to thank Sofia for supporting me in an almost everyday basis. It feels really nice to work in the same environment with a friend that can always listen and share all your feelings and concerns. A special thanks goes to my friend Valia for making my stay in Seattle a wonderful experience. All these years wouldn't have been the same without the presence of many friends that I met in Lausanne: Christina, Iris, Marina, Giorgos, Tasos, Skylos, Lorenzo, Emre, Panagiotis, Anthi, Pantelis, Giorgos, Ypatia, Yiannis, Vasilis and many more that are not mentioned here. I thank them all for the nice moments we had together.

I would also like to extend my deepest thanks to kurios Nikos and his family for their love and support all these years. I want to specifically thank him for always encouraging me to pursue academic studies.

A huge thanks goes to Boston, my "family" in Switzerland. All my gratitude goes to my roomy Vicky for her immense support and understanding during all these years, for tolerating all my PhD dependent mood swings and for always giving me reasons to cheer. I feel very lucky to have met her since my first days in Lausanne and share with her every single moment and emotion of this journey. A special thanks goes to Vagos for emitting so much positive energy, and transforming dull or stressful days into funny and joyful ones. I owe a big thanks to Yiorgos for being a very special person in my life, always supporting my decisions and caring about my wishes. He was always there to listen to my concerns with infinite patience, and make me smile in bad times. I would like to thank him also for his addiction to convex optimization that helped my research many times.

Finally, I am indebted to my family for always being there for me, and their unconditional love and support that makes me feel the happiest person in the world. The values that they have transmitted to me have made me what I am and will always guide me through life.

Abstract

Over the past few decades we have been experiencing an explosion of information generated by large networks of sensors and other data sources. Much of this data is intrinsically structured, such as traffic evolution in a transportation network, temperature values in different geographical locations, information diffusion in social networks, functional activities in the brain, or 3D meshes in computer graphics. The representation, analysis, and compression of such data is a challenging task and requires the development of new tools that can identify and properly exploit the data structure.

In this thesis, we formulate the processing and analysis of structured data using the emerging framework of graph signal processing. Graphs are generic data representation forms, suitable for modeling the geometric structure of signals that live on topologically complicated domains. The vertices of the graph represent the discrete data domain, and the edge weights capture the pairwise relationships between the vertices. A graph signal is then defined as a function that assigns a real value to each vertex. Graph signal processing is a useful framework for handling efficiently such data as it takes into consideration both the signal and the graph structure. In this work, we develop new methods and study specific applications related to the representation and structure-aware processing of graph signals in both centralized and distributed settings.

First, we study a novel yet natural application of the graph signal processing framework for the representation of 3D point cloud sequences. We exploit graph-based transform signal representations for addressing the challenging problem of compression of data that is characterized by dynamic 3D positions and color attributes. As temporally successive point cloud frames are similar, motion estimation is key to effective compression of these sequences. Performing motion estimation and compensation in this type of data is however challenging as the point cloud frames have varying numbers of samples without explicit correspondence information. We represent the time-varying geometry of these sequences with a set of graphs, and we then cast motion estimation as a feature matching problem between successive graphs. The estimated motion is successfully used for removing the temporal redundancy in the predictive coding for the 3D positions and the color characteristics of point cloud sequences.

Next, we depart from graph-based transform signal representations to design new overcomplete representations, or dictionaries, which are adapted to specific classes of graph signals. In particular, we address the problem of sparse representation of graph signals residing on weighted graphs by learning graph structured dictionaries that incorporate the intrinsic geometric structure of the irregular data domain and are adapted to the characteristics of the signals. We then model the graph signals as processes evolving on networks and we extend our dictionary learning framework to signals living on different graphs. The structural properties of the proposed dictionaries lead to compact representations that can be implemented in a computationally efficient manner in different signal processing tasks.

Then, we move to the efficient processing of graph signals in distributed scenarios, such as sensor or camera networks, which brings important constraints in terms of communication and computation in realistic settings. In particular, we study the effect of quantization in the distributed processing of graph signals that are represented by graph spectral dictionaries and we show that the impact of the quantization depends on the graph geometry and on the structure of the spectral dictionaries. Motivated by that observation, we propose a new dictionary learning solution that permits to control robustness to quantization noise in the distributed sparse representation of graph signals.

Finally, we focus on a widely used graph process, the distributed average consensus algorithm, which is typically addressed through the successive application of local graph filtering operators. We consider in particular the problem of distributed average consensus in a sensor network where sensors exchange quantized information with their neighbors. We propose a novel quantization scheme that depends on the graph topology and exploits the increasing correlation between the values exchanged by the sensors throughout the iterations of the consensus algorithm. A low complexity uniform quantizer is implemented in each sensor, and refined quantization is achieved by progressively reducing the quantization intervals along with the convergence of the consensus algorithm to a very smooth graph signal. The proposed quantization scheme is simple to implement in practical systems.

In summary, we address in this thesis several important problems in the processing of structured data that lives on networks or other irregular domains. We provide novel solutions for processing and analyzing graph signals in both centralized and distributed settings. We focus in particular in the theory of sparse graph signal representation and its applications and we bring some insights towards better understanding the interplay between graphs and signals on graphs. We hope that the research efforts in this thesis would benefit modern data processing applications such as inference in social networks, analysis of medical data or transmission and compression of high dimensional signals in vision sensor networks.

Keywords: signal processing on graphs, sparse representation, dictionary learning, distributed processing in sensor networks, quantized communication, 3D point cloud compression.

Résumé

Les dernières décennies ont vu une explosion de la quantité d'information générée par de grands réseaux de capteurs ou de multiples autres sources de données. Une grande partie de ces données sont naturellement structurées, comme par exemple l'évolution du trafic sur le réseau routier, les valeurs de températures à différentes positions géographiques, la diffusion d'information dans les réseaux sociaux, l'activité fonctionnelle dans le cerveau ou les maillages 3D en infographie. La représentation, l'analyse et la compression de telles données est une tâche difficile qui requiert le développement de nouveaux outils pour identifier et exploiter correctement le structure de l'information d'intérêt.

Dans cette thèse, nous regardons le traitement et l'analyse de données structurées du point de vue du domaine émergent du traitement de signal sur graphes. Les graphes sont des formes génériques pour la représentation de données, qui sont adaptés à la modélisation de la structure de signaux qui vivent dans des domaines topologiquement compliqués. Les nœuds du graphe représentent le domaine discret du signal, et les poids des liens du graphe caractérisent la relation entre des nœuds voisins. Un signal graphique est ensuite défini comme une fonction qui attribue une valeur réelle à chaque nœud du graphe. Le traitement de signal sur graphes est un cadre théorique très utile pour manipuler efficacement de telles données puisqu'il permet de prendre en considération à la fois le signal et la structure irrégulière définie par le graphe. Nous développons dans ce travail de nouvelles méthodes et nous étudions des applications particulières liées à la représentation et à l'analyse géométrique de signaux graphiques dans des configurations centralisées et distribuées.

Tout d'abord, nous étudions une application nouvelle et pourtant naturelle du traitement du signal sur graphes pour la représentation de séquences de nuages de points 3D. Nous exploitons des représentations basées sur des transformées sur graphes pour résoudre le problème non-trivial de la compression de données dynamiques qui correspondent à des positions 3D et des informations de couleur. La compression de telles données repose sur l'estimation efficace de la redondance temporelle par estimation de mouvement. Il est toutefois difficile d'effectuer de l'estimation et de la compensation de mouvement pour ce type de données, parce que les ensembles de points à différents instants sont de tailles différentes et n'ont pas de relations de correspondance explicites. Nous représentons la géométrie temporellement variable de ces séquences avec un ensemble de graphes, et nous formulons ensuite l'estimation de mouvement comme un problème de mise en correspondance de caractéristiques des graphes successifs. Le mouvement ainsi estimé est ensuite utilisé avantageusement pour réduire la redondance temporelle dans le codage prédictif des positions des points et de leur couleur dans les séquences de nuages de données 3D.

Ensuite, nous quittons les représentations basées sur des transformées sur graphes pour construire des nouvelles représentations redondantes, aussi appelées dictionnaires,

qui sont adaptées à des classes spécifiques de signaux graphiques. En particulier, nous nous intéressons au problème de la représentation parcimonieuse de signaux graphiques qui résident sur des graphes pondérés en apprenant des dictionnaires graphiques structurés qui incorporent la structure géométrique intrinsèque des domaines de données irréguliers, et qui sont adaptés aux caractéristiques des signaux. Nous modélisons ensuite les signaux graphiques comme des processus qui évoluent sur des réseaux et nous étendons notre cadre théorique pour l'apprentissage de dictionnaires à des signaux qui vivent sur des graphes différents. Nous montrons finalement que les dictionnaires appris avec les algorithmes proposés contiennent des atomes localisés sur le graphe, et qu'ils peuvent être implémentés de façon efficace dans des tâches de traitement de signal telles que la compression, le débruitage ou la classification.

Ensuite, nous continuons notre étude avec le traitement efficace de signaux graphiques dans des scénarios distribués, tels que les réseaux de capteurs ou de caméras, qui amènent en pratique des contraintes importantes en termes de calcul et de communication. En particulier, nous étudions les effets de la quantification des données sur le traitement distribué de signaux graphiques qui sont représentés par des dictionnaires graphiques et nous montrons que l'impact de la quantification dépend de la géométrie du graphe et de la structure des dictionnaires graphiques. Motivés par cette observation, nous proposons une nouvelle solution pour l'apprentissage de dictionnaires qui permet de contrôler la robustesse au bruit de quantification dans la représentation distribuée et parcimonieuse de signaux graphiques.

Finalement, nous nous concentrons sur un processus communément utilisé sur les graphes, l'algorithme distribué de consensus moyen, qui est généralement implémenté par l'application successive d'opérations locales de filtrage sur graphe. Nous considérons en particulier le problème de consensus moyen distribué dans un réseau de capteurs, où les capteurs échangent de l'information quantifiée avec leurs voisins. Nous proposons un nouvel algorithme de quantification qui dépend de la topologie du graphe et exploite la corrélation croissante entre les valeurs échangées par les capteurs au long des itérations de l'algorithme de consensus. Un quantificateur uniforme simple est implémenté dans chaque capteur, et un raffinement de la quantification s'opère en réduisant progressivement les intervalles de quantification au fur et à mesure que l'algorithme de consensus moyen converge vers un signal graphique très lisse. L'algorithme de quantification proposé est très simple à déployer en pratique.

En résumé, nous avons étudié dans cette thèse plusieurs problèmes importants liés au traitement de données structurées qui vivent sur des réseaux ou d'autres domaines irréguliers. Nous fournissons des solutions nouvelles pour le traitement et l'analyse de signaux graphiques dans des environnements centralisés ou distribués. Nous nous focalisons en particulier sur la théorie de la représentation parcimonieuse de signaux graphiques et ses applications, et nous apportons un éclairage sur les relations entre les graphes et les signaux graphiques. Nous espérons que la recherche proposée dans cette thèse va contribuer au développement d'applications modernes de traitement de données, telles que l'inférence dans des réseaux sociaux, l'analyse de données médicales ou la transmission et la compression de signaux de grandes dimensions dans des réseaux de capteurs d'images.

Mots clés : traitement de signal sur graphe, représentation parcimonieuse, apprentissage de dictionnaire, traitement distribué de données dans les réseaux de capteurs, communication quantifiée, compression de nuages de points 3D.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Thesis outline . . . 4
- 1.3 Summary of contributions . . . 6

2 Graph Signal Processing Overview **9**
- 2.1 Introduction . . . 9
- 2.2 Graphs and signals on graphs . . . 9
- 2.3 Signal representation on graphs . . . 11
  - 2.3.1 Spectral graph representation . . . 11
  - 2.3.2 Localized graph transforms . . . 14
- 2.4 Applications of graph-based signal processing . . . 15
  - 2.4.1 Processing with graph-based priors . . . 15
  - 2.4.2 Distributed processing of graph signals . . . 17
  - 2.4.3 Graph-based multimedia processing . . . 19
- 2.5 Summary . . . 20

3 Graph-based Compression of Dynamic 3D Point Cloud Sequences **23**
- 3.1 Introduction . . . 23
- 3.2 Structural representation of 3D point clouds . . . 26
  - 3.2.1 Octree representation of 3D point clouds . . . 26
  - 3.2.2 Graph-based representation of 3D point clouds . . . 26
- 3.3 Motion estimation in 3D point cloud sequences . . . 27
  - 3.3.1 Multi-resolution features on graphs . . . 27
  - 3.3.2 Finding correspondences on dynamic graphs . . . 28
  - 3.3.3 Computation of the motion vectors . . . 29
  - 3.3.4 Implementation details . . . 32
- 3.4 Compression of 3D point cloud sequences . . . 32
  - 3.4.1 Coding of motion vectors . . . 32
  - 3.4.2 Motion compensated coding of 3D geometry . . . 33
  - 3.4.3 Motion compensated coding of color attributes . . . 34
- 3.5 Experimental results . . . 35
  - 3.5.1 Motion estimation . . . 36

3.5.2 3D geometry compression 37
3.5.3 Color compression 40
3.5.4 Discussion 41
3.6 Related work 42
3.7 Conclusions 44

4 Learning Parametric Dictionaries for Signals on Graphs 45
4.1 Introduction 45
4.2 Parametric dictionary learning on a single graph 47
4.2.1 Translation of polynomial graph kernels 47
4.2.2 Dictionary structure 48
4.2.3 Dictionary learning algorithm 50
4.3 Computational efficiency of the learned polynomial dictionary 53
4.4 Experimental results 54
4.4.1 Synthetic signals 55
4.4.2 Approximation of real graph signals 61
4.5 Parametric dictionary learning from multiple graphs 64
4.5.1 Experimental results for multi-graph dictionary learning 68
4.6 Conclusion 74

5 Distributed Graph Signal Processing with Quantization 77
5.1 Introduction 77
5.2 Polynomial graph dictionaries for distributed signal representation 78
5.2.1 Distributed sensor network topology 78
5.2.2 Distributed computation of the graph operators 79
5.2.3 Illustrative application of distributed graph signal processing 81
5.3 Distributed processing with quantization 82
5.3.1 Distributed computation of $\mathcal{D}^T y$ with quantization 82
5.3.2 Distributed computation of $\mathcal{D}x$ with quantization 84
5.3.3 Distributed computation of $\mathcal{D}^T\mathcal{D}x$ with quantization 86
5.4 Distributed sparse graph signal regularization with quantization 87
5.5 Polynomial dictionary learning with quantization 90
5.6 Experimental results 91
5.7 Conclusions 95

6 Graph-based Quantization Refinement for Distributed Consensus 97
6.1 Introduction 97
6.2 Progressive quantizer for distributed average consensus 98
6.3 Design of the parameters of the progressive quantizer 100
6.3.1 Average case analysis 100
6.3.2 Exponential quantization range 102
6.4 Simulation results 104
6.4.1 Performance of the approximate exponential model 104
6.4.2 Comparison to uniform quantization 105

6.4.3 Comparison to existing quantization schemes for average consensus 106
6.4.4 Convergence of the consensus algorithm 108
6.5 Related Work 109
6.6 Conclusions 111

7 Conclusions **113**
7.1 Thesis achievements 113
7.2 Future directions 114

A Analysis of the sparse coding step **117**

B Continuous kernel approximation on graphs from discrete samples **121**

C Illustrative application of polynomial dictionaries: Image segmentation **123**

D Supplementary material for Chapter 6 **125**
D.1 **Proof of Proposition 2** 125
D.2 **Proof of Proposition 3** 126
D.3 **Proof of Proposition 4** 128
D.4 **Proof of Proposition 5** 128

List of Figures

1.1 Examples of graphs and signals on graphs: (a) traffic bottlenecks on the transportation graph, (b) average temperature on a geographical graph, (c) fMRI brain signal in a structural, anatomical graph, (d) gender attributes on a social network graph, and (e) 3D color attributes on a mesh graph. The size and the color of each disc in (a)-(c) indicate the value of the signal at the corresponding vertex. The red and the blue square lines in (d) correspond to female and male respectively. 2

1.2 Decomposition of a graph signal (a) in four localized simple components (b), (c), (d), (e). Each component is a heat diffusion process ($e^{-\tau\mathcal{L}}$) at time τ that has started from different network nodes (δ_n). 3

1.3 A wireless sensor network where the graph captures the communication pattern in the network. Each sensor exchanges information only with its one-hop neighbors. The exchanged messages are quantized i.e., they are transmitted only using a limited number of bits. 4

2.1 A randomly generated graph of 30 vertices, with edges connecting geographically nearby vertices. The illustrative signal on the graph is represented by black and blue bars, whose height represents the signal value at each vertex, with blue denoting the positive values and black negative ones. 11

2.2 Translation of the same signal y to three different nodes on the graph. The size and the color of each disk represent the signal value at each vertex. Due to the irregular topology, the translated signals appear different but contain the same graph spectral information. 13

3.1 Sequence of point cloud frames captured at different time instances in the 'man' sequence. 24

3.2 Example of a point cloud of the 'yellow dress' sequence (a). The geometry is captured by a graph (b) and the r component of the color is considered as a signal on the graph (c). The size and the color of each disc indicate the value of the signal at the corresponding vertex. 25

3.3 Schematic overview of the encoding architecture of a point cloud sequence. Motion estimation is used to reduce the temporal redundancy for efficient compression of the 3D geometry and the color attributes. 25

3.4 Octree decomposition of a 3D model for two different depth levels. The points belonging to each voxel are represented by the same color. 26

3.5 Schematic overview of the motion vector coding scheme. The motion vectors v_t^* between two consecutive frames of the sequence are transformed in the graph Fourier domain, quantized uniformly, and sent to entropy coded. The decoder performs the reverse procedure to obtain $\widehat{v_t^*}$. 33

3.6 Illustration of the geometry compression of the target frame (TF) based on the motion compensated reference frame (RF). (a) Differential encoding of the consecutive frames where structural changes within octree occupied voxels are extracted during the binary serialization process and encoded using the XOR operator. The bit stream of the XOR operator is sent to the decoder. The figure is inspired by [1]. (b) Schematic overview of the overall 3D geometry coding scheme. 34

3.7 Schematic overview of the predictive color coding scheme. The color residual in each block of the octree is projected in the graph Fourier domain. The graph Fourier coefficients are uniformly quantized and entropy coded based on the scheme of [2]. The bit stream is sent to the decoder where the inverse operations are performed to decode the color of the target frame. 35

3.8 Illustrative frames of the upper body sequence. 36

3.9 Example of motion estimation and compensation in the yellow dress, man and upper body motion sequences. The superimposition of the reference ($\mathcal{I}_t$) and target frame ($\mathcal{I}_{t+1}$) is shown in (a), (d), and (g) while in (b), (e), (h) we show the correspondences between the target (red) and the reference frame (green). The superposition of the motion compensated reference frame ($\mathcal{I}_{t,mc}$) and the target frame ($\mathcal{I}_t$) is shown in (c), (f), and (i). Each small cube corresponds to a voxel in the motion compensated frame. 37

3.10 Performance comparison of the average signal-to-quantization noise ratio (SQNR) versus bits per vertex (bpv) for coding the motion vectors in the graph Fourier domain and in the signal domain. 38

3.11 Effect of the coding rate of the motion vectors on the overall coding rate of the geometry for the motion compensated dual octree algorithm. By sending the motion vectors at low bit rate (≈ 0.1 bpv), the motion compensated dual octree scheme performs slightly better than the static octree and the dual octree compression algorithm. 39

3.12 Color compression performance (dB) vr. bits per vertex for independent and differential coding on the three datasets for a quantization stepsize of $\Delta = [32, 64, 256, 512, 1024]$. 41

3.13 Rendering results of a point cloud frame from the yellow dress sequence compressed at a quantization stepsize of $\Delta = 1024$, and $\Delta = 32$; (a) original point cloud, (b) voxalized and decoded frame for $\Delta = 1024$, and (c) voxalized and decoded frame for $\Delta = 32$. 42

4.1 Illustrative example: The three signals on the graph are the minutes of bottlenecks per day at different detector stations in Alameda County, California, on three different days. The detector stations are the nodes of the graph and the connectivity is defined based on the GPS coordinates of the stations. The size and the color of each ball indicate the value of the signal at each vertex of the graph. Note that all signals consist of a set of localized features positioned on different nodes of the graph. 46

4.2 Comparison of the generating kernels $\{\widehat{g}_s(\cdot)\}_{s=1,2,\dots,S}$ (shown in (a)) with the kernels learned by the polynomial dictionary learning algorithm for $M = 400$, $M = 600$ and $M = 2000$ training signals. 56

4.3 Comparison of the learned polynomial dictionary to the SGWT [3], K-SVD [4] and the graph structured dictionary [5] in terms of approximation performance on test data generated from a polynomial generating dictionary, for different sizes of the training set. 57

4.4 Kernels $\{\widehat{g}_s(\cdot)\}_{s=1,2,3,4}$ learned by the polynomial dictionary algorithm for (a) $K = 5$, (b) $K = 10$, and (c) $K = 20$. 58

4.5 Learned atoms centered on vertex $n = 1$, from each of the subdictionaries. 59

4.6 Comparison of the average approximation performance of our learned dictionary on test signals generated by the non-polynomial synthetic generating dictionary, for $K = \{5, 10, 20, 25\}$. 60

4.7 Comparison of the learned polynomial dictionary to the SGWT [3], K-SVD [4] and the graph structured dictionary [5] in terms of approximation performance on test data generated from a non-polynomial generating dictionary, for different sizes of the training set. 61

4.8 (a) An example of a graph signal from the testing set and its atomic decomposition with respect to (b) the K-SVD dictionary, (c) the dictionary learned by [5] and (d) the learned polynomial graph dictionary. 62

4.9 Comparison of the learned polynomial dictionaries to the SGWT, K-SVD, and graph structured dictionaries [5] in terms of approximation performance on testing data generated from the (a) Flickr, (b) traffic, and (c) brain datasets, for $T_0 = 6$. 64

4.10 Examples of atoms learned from training data from the brain dataset with (a) K-SVD and (b) the polynomial dictionary. The six atoms illustrated here are the ones that were most commonly included by OMP in the sparse decomposition of the testing signals. 65

4.11 Generating kernels $\{\widehat{g}_s(\cdot)\}_{s=1,2,\dots,S}$ as a function of the eigenvalues. 69

4.12 Comparison of multi-graphs learning, with SGWT and single-graph learning in terms of SNR in the synthetic data. 70

4.13 Generating kernels of the four graph models 71

4.14 Learned kernels of the four graph models. Learning is based on (a) and (b). . . . 72

4.15 Learned kernels of the four graph models. Learning is based on (a) and (b). . . . 73

4.16 Comparison of multi-graphs learning, with SGWT and single-graph learning in terms of SNR in the traffic delay dataset for three different counties. 74

5.1 Distributed computation of $\mathcal{D}_s y$ in a sensor network. Sensor n exchanges messages with its one-hop neighbors for K iterations. After each iteration, the received messages are filtered with weights defined by the graph Laplacian i.e., $z_k(n) = (\mathcal{L}z_{k-1})(n)$, and are transmitted locally in the network. $(\mathcal{D}_s y)(n)$ is computed as a linear combination of the messages exchanged during the K iterations. 80
5.2 Distributed computation of $\mathcal{D}_s y$ with quantization in a sensor network. Sensor n exchanges quantized messages with its one-hop neighbors for K iterations. After each iteration $k-1$, the received messages $\tilde{z}_{k-1}$ are filtered with weights defined by the graph Laplacian (i.e., $z_k(n) = (\mathcal{L}\tilde{z}_{k-1})(n)$), quantized (i.e., $\tilde{z}_k(n) = Q(z_k(n))$), and finally transmitted locally in the network. $(\mathcal{D}_s y)(n)$ is computed as a linear combination of the messages exchanged during the K iterations. 83
5.3 Distributed approximation performance (SNR) versus sparsity level, achieved with different polynomial graph dictionary learned with different values of η in distributed settings, with a bit rate of 6 bits per message. 92
5.4 Illustration of the kernels recovered for different values of η in the dictionary learning problem. 93
5.5 Polynomial coefficient values, for each of the dictionary kernels, for different values of η in solving the optimization problem (5.22). 93
5.6 Approximation performance (SNR) versus sparsity level, achieved with the polynomial graph dictionary for different values of η in centralized settings. 94
5.7 Denoising performance (in dB) versus bits per message using dictionaries learned with different values of η. The SNR of the input signals is 10 dB. 95
5.8 Denoising performance (in dB) versus bit rate per message for the traffic dataset using dictionaries learned with different values of η. The initial SNR is 10 dB. . . . 96

6.1 (a) Evolution of the β_t values over the iterations. (b) Comparison of the average consensus performance of the Progressive Quantizer with parameters generated (i) recursively and (ii) with a linear model. 105
6.2 Average consensus performance of the proposed quantization scheme (ProgQ) vs uniform quantizer with a constant range (UnifQ) for 2, 4 and 6 bits. 106
6.3 Average consensus performance of the proposed quantization scheme (ProgQ) vs the adaptive quantizer (AdaptQ) [6] for 2, 4 and 6 bits. 107
6.4 Average consensus performance of the proposed quantization scheme (ProgQ) vs the zoom-in, zoom-out uniform quantizer (ZoomQ) [7] for 2, 4 and 6 bits. 108
6.5 Average consensus performance of the proposed quantization scheme (ProgQ) vs the quantization scheme proposed in [8] (Li et al.) for 2, 4 and 6 bits. 109
6.6 (a) Evolution of the average consensus performance for an exponentially reducing quantization range. (b) Evolution of the variance of the quantization noise over the iterations for 2, 4 and 6 bits. 110

C.1 Learned atoms (a) and segmentation results (d,e) obtained using the polynomial dictionary on the house (b) and cameraman (c) images. 123

List of Tables

3.1 Color Prediction Error (SNR in dB) 40
4.1 Parameters of Real Graph Signals 63

Chapter 1

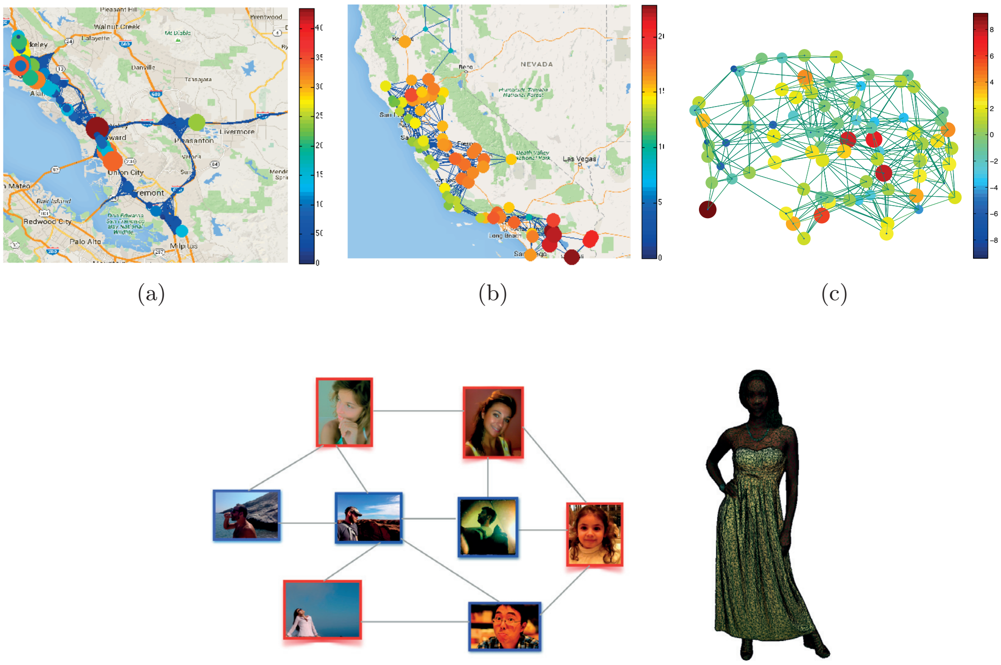
Introduction

1.1 Motivation

Over the past few decades, we have witnessed a deluge of information generated by numerous data sources, in a large variety of applications. For example, sensor networks have been widely deployed to measure a plethora of physical entities like temperature and solar radiation, traffic volumes in transportation networks, brain activities in biological networks. Online social networks such as Twitter, Facebook have turned into a significant means of communication and contain a lot of information. 3D depth cameras are yet becoming more powerful and widely used to capture dynamic 3D scenes in emerging applications such as gaming, immersive communication and virtual reality. Such data is usually very complex for different reasons. First, it is extremely high-dimensional and occupies a large amount of storage space. For example, 3D depth cameras provide rich information about the environment, that consists of 3D models with millions of points. Second, most data is intrinsically and possibly irregularly structured. For instance, wireless sensor networks are irregularly deployed in space and their measurements depend on their geographical positions. Third, the data and the structure may be generated by different sources of information. For example, the information spread in social networks may be influenced by the relationships between the entities, as well as the type of data itself. The representation, analysis, and compression of such data is a challenging task that requires the development of new tools that can identify and properly exploit data structures.

In this thesis, we study the representation and analysis of structured data in the context of the emerging graph signal processing framework. Graphs are generic data representation forms that are suitable for modeling the geometric structure of signals that live on topologically complicated domains. These signals are either intrinsically discrete (e.g., attributes of entities in social networks) or sampled from a continuous process (e.g., heat diffusion on a manifold). Typically, the vertices of the graph represent the discrete data domain and carry the data values. The edge weights of the graph capture the pairwise relationships between the vertices, like geographical distance or biological connections, for example. A graph signal is then defined as a function that assigns a real value to each vertex. Examples of graphs and signals on graphs are illustrated in Fig. 1.1.

Graph representations lead to rich data description on irregular domains and, if properly ex-

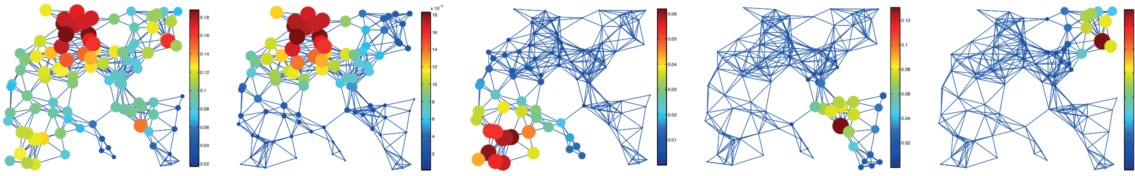


(a) (b) (c)

(d) (e)

Figure 1.1: Examples of graphs and signals on graphs: (a) traffic bottlenecks on the transportation graph, (b) average temperature on a geographical graph, (c) fMRI brain signal in a structural, anatomical graph, (d) gender attributes on a social network graph, and (e) 3D color attributes on a mesh graph. The size and the color of each disc in (a)-(c) indicate the value of the signal at the corresponding vertex. The red and the blue square lines in (d) correspond to female and male respectively.

ploited, permit to efficiently capture the evolution of signals in a priori complex high-dimensional data sets. Signals and graphs are usually defined using different types of information which, if combined properly, can be quite helpful in analyzing or inferring information in the datasets. Moreover, graph signal representations provide a natural way to handle signals that cannot be easily processed with classical tools due to their irregular structure. The price to pay for this flexibility is the fact that one has to develop new tools and algorithms that handle efficiently the graph structure, possibly by leveraging intuition from classical signal processing in Euclidean spaces. Although the foundations of graph theory date back to the 18^{th} century with the celebrated problem of the Seven Bridges of Königsberg, most of the research in the machine learning and computer science community has so far focused in understanding and analyzing the graph structure, without paying particular attention to signals residing on the vertices of those graphs. Adapting classical signal processing tools to signals defined on graphs has however raised significant interest in the last few years [9], [10]. It requires the combination of different fields such as algebraic and spectral graph theory, harmonic analysis, and application domain expertise. Even if this research area looks highly promising because it provides a framework for modeling complex and irregularly structured discrete datasets, the challenges are many and the field is still in its infancy.



(a) Graph signal (b) $e^{-15\mathcal{L}}\delta_4$ (c) $e^{-15\mathcal{L}}\delta_{30}$ (d) $e^{-5\mathcal{L}}\delta_{80}$ (e) $e^{-5\mathcal{L}}\delta_{50}$

Figure 1.2: Decomposition of a graph signal (a) in four localized simple components (b), (c), (d), (e). Each component is a heat diffusion process ($e^{-\tau\mathcal{L}}$) at time τ that has started from different network nodes (δ_n).

A first challenge consists in properly using the graph-based signal representations in problems that cannot be easily solved with classical signal processing tools. For example, graph representations have recently started to gain some attention in the computer graphics and computer vision communities, where they have been used for representing and organizing the irregular geometry of 3D models captured with 3D depth sensors. These 3D point clouds are not regularly organized and even have little explicit spatial structure, which prevents the use of classical signal processing tools. Graphs seem to be a promising framework for processing and compressing this new type of data. Then, the analysis of data that lives on networks is certainly a very good fit for graph signal processing. For example, the graph structure and a proper signal representation on the graph are expected to provide two useful sources of information for mining and inferring typical behaviors or patterns in social networks, detecting the spread of a disease in biological networks, predicting physical changes in the environment, or reducing the energy consumption in energy networks.

One of the fundamental challenges when dealing with graph signals actually amounts to the design of signal representations and appropriate signal models that incorporate the graph structure and can be efficiently implemented. Representing graph signals using classical signal processing transforms such as DCT and wavelets, or data-driven and structure-agnostic dictionaries, is definitely suboptimal as it ignores key data dependencies and geometry arising from the irregular graph domain. Graph signal representations have definitely to be adapted to the graph structure. This can be done by either adapting traditional signal models such as smoothness and sparsity to the graph setting or by designing new models that can reveal particular characteristics of the graph signals. An example is given in Fig. 1.2 that shows a graph signal that is a linear combination of a sparse set of localized components on the graph generated by a heat diffusion process starting at four different network nodes. A good data representation should therefore be able to identify the core components of the graph signals i.e., the underlying processes evolving on the graph and their localization on the network.

Finally, the effective implementation of graph signal processing operators represents another important research challenge. In particular, as a natural application of graph signal processing lies in the vast field of sensor networks, it is important that signal processing methods can be effectively distributed. In such networks, each sensor typically communicates short messages only with a small number of neighbor nodes due to energy constraints. As a result, the information exchanged by the network nodes is quantized prior to transmission, which poses further challenges in terms of the proper convergence of the distributed graph signal processing algorithms. An illustration

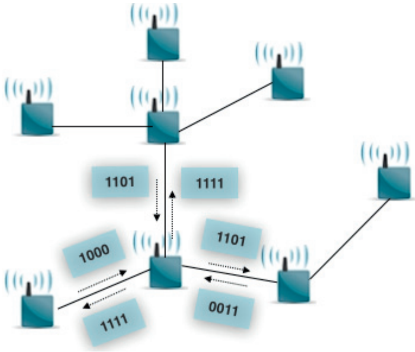


Figure 1.3: A wireless sensor network where the graph captures the communication pattern in the network. Each sensor exchanges information only with its one-hop neighbors. The exchanged messages are quantized i.e., they are transmitted only using a limited number of bits.

of quantized communication in a sensor network is given in Fig. 1.3 where sensor nodes have to exchange their information with only a limited number of bits. Topology-aware quantization algorithms could increase the accuracy of the exchanged messages given a limited bit budget.

The above research questions are only some of the numerous open theoretical and application challenges in the promising field of signal processing on graphs. From the theoretical point of view, there is a need for novel signal representation methods that can reveal the main characteristics of the graph signals, and at the same time, can be efficiently implemented. From an application perspective, the main challenge consists in using appropriately the graph signal representations for handling and analyzing complex dataset. In this thesis, we study various problems related to the representation and processing of graph signals in both centralized and distributed settings and provide new solutions to the design of effective graph signal representation methods and their implementation in various illustrative applications.

1.2 Thesis outline

The goal of this thesis is to present solutions as well as in-depth analyses of a few of the most important issues that arise in the emerging field of graph signal processing.

Initially, we review in Chapter 2 the current state-of-the-art methods for graph signal representations and their applications in both centralized and distributed settings. First, we give the basic definitions and notation used in this thesis for graphs and signals on graphs, and we review the generalization of classical transforms such as Fourier and wavelets to the irregular graph domain. Then, we give an overview of the use of graph-based regularizers in solving ill-posed inverse problems such as graph signal denoising and inpainting in centralized and distributed settings. The related work chapter concludes with applications of graph signal processing in visual data representation, processing and compression.

Chapter 3 studies a novel yet natural application of graph signal processing in the representation of 3D point cloud sequences and exploits graph-based transform signal representations for addressing the challenging problem of the compression of data that is characterized by dynamic

3D positions and color attributes. As temporally successive point cloud frames are similar, motion estimation is key to effective compression of these sequences. This however represents a challenging problem as the point cloud frames have varying numbers of points without explicit correspondence information. We design a new solution to this problem by building on the interesting properties of the graph signal processing framework. We represent the time-varying geometry of these sequences with a set of graphs, and consider 3D positions and color attributes of the point clouds as signals on the vertices of the graphs. We then cast motion estimation as a feature matching problem between time consecutive graphs. The motion is estimated on a sparse set of representative vertices using new spectral graph wavelet descriptors. A dense motion field is eventually interpolated at every vertex by solving a graph-based regularization problem. The estimated motion is finally used for removing the temporal redundancy in the predictive coding of the 3D positions and the color characteristics of the point cloud sequences. Experimental results demonstrate that our novel method is able to accurately estimate the motion between consecutive frames. Moreover, motion prediction is shown to bring significant improvement in terms of the overall compression performance. To the best of our knowledge, this is the first work that exploits both the spatial correlation inside each frame (through the graph-based transform) and the temporal correlation between the frames (through the graph-based motion estimation) to effectively compress the color and the geometry of 3D point cloud sequences.

Next, in Chapter 4, we depart from the graph-based transform signal representations used in Chapter 3 to new overcomplete representations, or dictionaries, that are adapted to both the graph structure and the signals at hand. In particular, we study the notion of sparsity in the graph settings and propose a novel parametric dictionary learning algorithm to design data-adapted parametric dictionaries that can sparsely represent graph signals. More specifically, we model graph signals as combinations of overlapping local patterns on graph. We impose the constraint that the global dictionary is a concatenation of subdictionaries, where each subdictionary is constructed on a polynomial of the graph Laplacian matrix and represents a single pattern translated to different areas of the graph. The learning algorithm adapts the patterns to a training set of graph signals. We further extend our algorithm to the effective representation of graph signals that live on different graph topologies. We exploit the Laplacian polynomial form to learn dictionaries that provide sparse representations for classes of signals that share common spectral characteristics but reside on the vertices of different graphs. Experimental results on both synthetic and real datasets demonstrate that the dictionaries learned by the proposed algorithm are competitive with and often better than unstructured dictionaries learned by state-of-the-art numerical algorithms in terms of sparse approximation of graph signals. We then perform experiments on graph signals that represent common processes on different graphs and show that our dictionary learning method is able to recover the actual components of these signals. Importantly, in contrast to the structure-agnostic dictionaries, the dictionaries learned by the proposed algorithm feature localized atoms and can be implemented in a computationally efficient manner in different signal processing tasks.

We move to the problem of processing graph signals in distributed settings under communication rate constraints in Chapter 5. We study in particular the distributed processing of graph signals that are well represented by graph spectral dictionaries. We first analyze the impact of quantization noise in the distributed computation of polynomial dictionary operators that are commonly used in various signal processing tasks. We show that the impact of quantization depends on the graph geometry and on the structure of the spectral dictionaries. Then, we focus on the problem of distributed sparse signal representation that can be solved with an iterative soft thresholding algorithm. We define conditions on the dictionary structure to ensure the convergence of the distributed algorithm and finally propose a new dictionary learning solution that permits to control the robustness of the distributed signal processing tasks to quantization noise. Experimental results for reconstruction and denoising of both synthetic and realistic signals illustrate the tradeoffs that exist between accurate signal representation and robustness to quantization error in the design of dictionary operators for distributed processing.

Finally, in Chapter 6, we focus on the distributed average consensus problem, which has been typically addressed in the literature through the successive application of local graph filtering operators. We consider in particular the problem of distributed average consensus in a sensor network where sensors exchange quantized information with their neighbors. We propose a novel quantization scheme that exploits the increasing correlation between the values exchanged by the sensors throughout the iterations of the consensus algorithm. A low complexity, uniform quantizer is implemented in each sensor, and refined quantization is achieved by progressively reducing the quantization intervals along with the convergence of the consensus algorithm. We propose a recurrence relation for computing the quantization parameters that depend on the network graph topology, the graph process itself, and the communication rate. We further show that the recurrence relation can lead to a simple exponential model for the value of the quantization step size over the iterations. Finally, simulation results demonstrate the effectiveness of the progressive quantization scheme that leads to the consensus solution even at low communication rate.

In summary, we address in this thesis several important problems related to the emerging field of signal processing on graphs, and provide novel solutions for processing and analyzing graph signals in both centralized and distributed settings. We focus in particular on the theory of sparse graph signal representation and its applications and bring some insights towards understanding the interplay between graphs and signals on graphs. Moreover, the research effort presented in this thesis provides evidence of the usefulness and the relevance of the graph signal processing framework in solving a broad category of real science and engineering problems.

1.3 Summary of contributions

The main contributions of this thesis are summarized below.

- We propose a novel graph-based algorithm for solving the challenging problem of motion estimation and compensation on 3D point cloud sequences. The estimated motion is used to design a new compression framework that is based on removing the temporal redundancy and performing predictive coding of the 3D positions and the color attributes. The proposed framework is tested and validated in real 3D point cloud sequences and is shown to provide significant gain with respect to intra frame coding.

- We introduce a new dictionary learning algorithm for constructing flexible forms of graph structured dictionaries, which sparsely represent graph signals and are adapted to the signals at hand. The novel algorithm is tested in the approximation of signals representing (i) Flicker users that have been taking photos in the geographical area of London, (ii) bottleneck duration in the transportation network of California, and (iii) fMRI signals in the brain network acquired on different subjects.

- We extend the proposed dictionary learning framework to signals that live on different graph topologies but share some common spectral characteristics. We show that our dictionary learning method is able to recover the common characteristics and core components of these signals.
- We study the distributed processing of graph signals that are well represented by graph spectral dictionaries, and the effect of quantization on different distributed processing tasks performed with such dictionaries. Furthermore, we propose a new dictionary learning solution for the sparse representation of graph signals that permits to control the robustness of distributed processing tasks to quantization noise.
- We propose a novel quantization scheme for distributed average consensus that exploits the increasing correlation between the values exchanged by the sensors throughout the successive application of the local graph filtering operator. Our quantization scheme is driven by the characteristics of the graph topology and the communication rate, and is shown to provide significant gain in terms of rate distortion performance.

Chapter 2

Graph Signal Processing Overview

2.1 Introduction

In order to represent efficiently graph signals, one needs to account for the intrinsic geometric structure of the underlying graph. Signal characteristics such as smoothness depend on the irregular topology of the graph on which the signal resides. Classical signal processing tools that have been designed for regular signal structures are therefore inappropriate for the irregular structures in the graph setting. A lot of effort has been recently dedicated to designing new tools and algorithms that can handle efficiently the new challenges arising from the irregular nature of networks or other graph supports. These tools are based on a combination of computational harmonic analysis with algebraic and spectral graph theoretical concepts [9].

In this chapter, we review principal graph signal processing methods from the literature, which are related to the problems studied in this thesis. First, we give the basic definitions and notation for graphs and signals on graphs, that will be used in the rest of the thesis. Next, we review the generalization of classical transforms such as Fourier and wavelet transforms to the irregular graph domain. In the sequel, we focus on the use of graph-based signal processing tools in different applications. In particular, we review the use of graph-based regularizers in solving inverse problems such as graph signal denoising and inpainting. We also provide an overview of tools for processing graph signals in a distributed fashion. Finally, we quickly review the use of graph-based signal processing tools for image and 3D data, which represent a popular application area for this emerging framework.

2.2 Graphs and signals on graphs

In this section, we briefly recall a few basic definitions for signals on graphs. We generally consider a weighted and undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$ where $\mathcal{V}$ and $\mathcal{E}$ represent the vertex and edge sets of the graph, and W represents the matrix of edge weights, with $W_{nm} = W_{nm}$ denoting the positive weight of an edge connecting vertices n and m; $W_{nm} = 0$ if there is no edge between vertices m and n. The degree of a node n is defined as the sum of the degree of the incoming edges, that can be computed as the sum of the weight values in the n^{th} row of the matrix W. We assume that the graph is connected and that it consists of N nodes. The j-hop neighborhood $\mathcal{N}_{j,n} = \{v \in \mathcal{V} : d(v, n) \leq j\}$

of node n is the set of all nodes that are at most j-hop away from node n.

The combinatorial graph Laplacian operator is defined as

$$L = D - W, \tag{2.1}$$

where D is the diagonal degree matrix whose n^{th} diagonal element is equal to the sum of the weights of all the edges incident to vertex n [11]. It is a positive semi-definite matrix that has a complete set of real orthonormal eigenvectors with corresponding nonnegative eigenvalues. We denote its eigenvectors by $\chi = [\chi_0, \chi_1, ..., \chi_{N-1}]$, and the sorted spectrum of eigenvalues by

$$\sigma(\mathcal{L}) := \Big\{0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_{N-1}\Big\}.$$

The smallest eigenvalue of the combinatorial Laplacian is always zero, and the corresponding eigenvector is a constant vector. The largest eigenvalue depends on the maximum degree of the graph. Moreover, the combinatorial Laplacian is associated with the incidence matrix, as shown in [11].

For connected graphs, the normalized graph Laplacian is closely related to the combinatorial Laplacian and is defined as

$$\mathcal{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}, \tag{2.2}$$

where I is the identity matrix. As in the case of the combinatorial Laplacian, the eigenvalues are non-negative, with the smallest one corresponding to zero. A nice property of the eigenvalues of the normalized Laplacian is that they are contained between the interval [0, 2], which makes it easier to compare the distribution of the eigenvalues between different graphs, especially if there is a large difference in the number of vertices. Furthermore, its eigenvalues are consistent with the eigenvalues in the spectral geometry and in stochastic processes, such as random walks [11]. The combinatorial and the normalized graph Laplacians are both examples of generalized graph Laplacians [12] and they are both popular in many graph related applications. In general, when the graph is almost regular, the combinatorial and the normalized Laplacian have similar spectra. Although the combinatorial Laplacian has been widely used in the literature, in applications that have a random walk or diffusion interpretation, or in those where weighting vertices by their degrees is more natural, the normalized Laplacian can give in general better results. For these reasons, in this thesis, we mainly use the normalized graph Laplacian. We focus only on undirected graphs. For the sake of completeness though, we note that the definition of the Laplacian can be easily extended to directed graphs [13].

A graph signal y in the vertex domain is a real-valued function defined on the vertices of the graph $\mathcal{G}$, such that $y(n)$ is the value of the function at vertex $n \in \mathcal{V}$. An example of a graph and a signal on the graph is given in Fig. 2.1. In this particular example, vertices that are connected by an edge have similar signal values. Thus, the variation of the signal on the graph depends on the graph connectivity. It is therefore important to process graph signals by considering both the signal values and the irregular graph support.

Different frameworks have been developed for processing signals on the graph. The first one [9] relies on spectral graph theory [11], and is based on the spectral decomposition of the eigenvalues

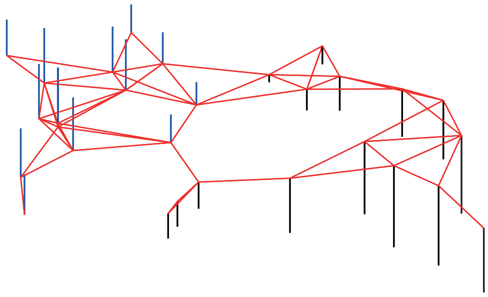


Figure 2.1: A randomly generated graph of 30 vertices, with edges connecting geographically nearby vertices. The illustrative signal on the graph is represented by black and blue bars, whose height represents the signal value at each vertex, with blue denoting the positive values and black negative ones.

of the graph Laplacian matrix. The second one relies on the algebraic signal processing theory, and builds upon graph shift operators that are based on the adjacency matrix of the graph [10]. Its main advantage is that it can be easily generalized to directed graphs. Recently, the authors in [14] build on the first framework and provide a probabilistic interpretation for signals and operators on graphs, by modeling the graph signals as Gaussian Markov Random Fields. In the rest of this thesis, we adopt the spectral graph theory framework, which provides direct connections with the framework of harmonic analysis in regular signal domains. By analogy to the continuous Laplacian operator whose eigenfunctions are the classical Fourier modes and its eigenvalues are the frequencies, the eigenvectors and the eigenvalues of the graph Laplacian are related to the graph Fourier notion of frequency. We present it in details in the next section.

2.3 Signal representation on graphs

2.3.1 Spectral graph representation

The fundamental analogy between traditional signal processing and graph signal processing is established through the spectral graph theory [11]. In particular, the generalization of the classical Fourier transform to graph settings has been established through the eigenvectors and the eigenvalues of the graph Laplacian matrix [3], which carry a notion of frequency for graph signals. In particular, the graph Laplacian eigenvectors associated with small eigenvalues correspond to signals that vary slowly across the graph, hence they can be associated with the notion of low frequency. In other words, if two vertices are connected by an edge with a large weight, the values of the low frequency eigenvectors at those locations are likely to be similar. The eigenvectors associated with larger eigenvalues take values that change more rapidly on the graph; they are more likely to have dissimilar values on vertices connected by an edge with high weight. The eigenvectors of the graph Laplacian are therefore considered to represent a Fourier basis for graph signals. For any function y defined on the vertices of the graph, the graph Fourier transform $\hat{y}(\lambda_\ell)$ at frequency λ_ℓ is thus defined as the inner product with the corresponding eigenvector χ_ℓ [3]

$$\hat{y}\left(\lambda_\ell\right) = \langle y, \chi_\ell \rangle = \sum_{n=1}^{N} y(n)\chi_\ell^*(n), \tag{2.3}$$

where the inner product is conjugate-linear in the first argument, and $\chi_\ell^*(n)$ is the conjugate value of the eigenvector χ_ℓ at node n. The inverse graph Fourier transform is

$$y(n) = \sum_{\ell=0}^{N-1} \hat{y}\left(\lambda_\ell\right)\chi_\ell(n), \quad \forall n \in \mathcal{V}.$$

The Fourier bases can be chosen as the eigenvectors of either the combinatorial or the normalized graph Laplacian matrices[1]. Both spectrums have a frequency-like interpretation [9]. We notice that, as in the classical Euclidean settings, the spectral domain representation provides important information about the graph signals. For example, analogously to the classical case, the graph Fourier coefficients of a smooth signal decay rapidly. Such signals are compressible as they can be closely approximated by just a sparse set of Fourier coefficients [15]. As we will see, this property is used in many applications such as compression or regularization of graph signals.

Besides its use in spectral analysis, the graph Fourier transform is also useful in generalizing traditional signal processing concepts such as convolution, translation, or modulation to graph settings. In particular, the relation between the vertex and the spectral graph domain has been used to define the convolution on the graph. Given two graph signals y, h the result of the convolution of these two signals on vertex n is defined as [16, 3]

$$(y * h)(n) = \sum_{\ell=0}^{N-1} \hat{y}(\lambda_\ell)\hat{h}(\lambda_\ell)\chi_\ell(n), \tag{2.4}$$

which imposes the property that the convolution in the vertex domain is equivalent to a multiplication in the graph spectral domain. The generalized translation operator of a graph signal y to a node n can be defined as a generalized convolution with a Kronecker δ function centered at vertex n [17, 16, 3]:

$$T_n y = \sqrt{N}(y * \delta_n) \overset{(a)}{=} \sqrt{N} \sum_{\ell=0}^{N-1} \hat{y}(\lambda_\ell)\chi_\ell^*(n)\chi_\ell, \tag{2.5}$$

where the normalizing constant $\sqrt{N}$ ensures that the translation operator preserves the mean of the signal[2]. The Kronecker function δ_n is an N-dimensional signal that is zero everywhere on the graph except from node n, where it takes the value of one. Moreover, the equality (a) follows from Eq. (2.4). An example of the translation of a signal y in different nodes of the graph is illustrated

---

[1] For the sake of simplicity we use the same notation $\chi = [\chi_0, \chi_1, ..., \chi_{N-1}]$ for the vectors of both the combinatorial and the normalized Laplacian matrix. However, in each chapter we specify the Laplacian matrix that this notation refers to.

[2] When the Fourier basis is formed by the eigenvector of the normalized Laplacian, $\sqrt{N}$ is substituted by $\|\sqrt{d}\|$, where d is a vector containing as elements the degree of each node.

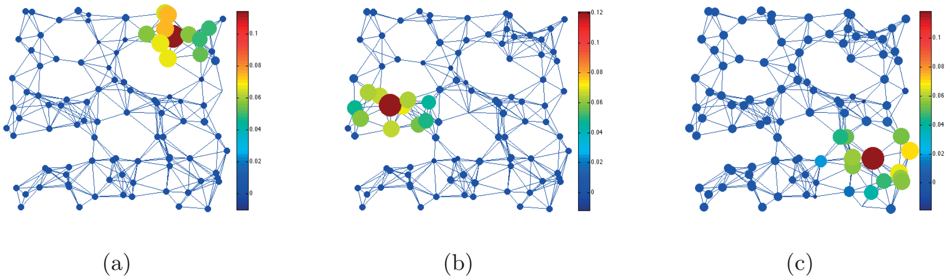

(a) (b) (c)

Figure 2.2: Translation of the same signal y to three different nodes on the graph. The size and the color of each disk represent the signal value at each vertex. Due to the irregular topology, the translated signals appear different but contain the same graph spectral information.

in Fig. 2.2. We notice that the classical shift in the classical definition of the translation does not apply on graphs.

Filtering is another fundamental operation in graph signal processing. Similarly to classical signal processing, the outcome y_{out} of the filtering of a graph signal y with a graph filter h is defined in the spectral domain as the multiplication of the graph Fourier coefficient $\hat{y}(\lambda_\ell)$ with the transfer function $\hat{h}(\lambda_\ell)$ such that

$$\hat{y}_{out}(\lambda_\ell) = \hat{y}(\lambda_\ell)\hat{h}(\lambda_\ell), \quad \forall \lambda_\ell \in \sigma(\mathcal{L}). \tag{2.6}$$

The filtered signal y_{out} at node n is given by taking the inverse graph Fourier transform of Eq. (2.6), such that

$$y_{out}(n) = \sum_{\ell=0}^{N-1} \hat{y}(\lambda_\ell)\hat{h}(\lambda_\ell)\chi_\ell(n). \tag{2.7}$$

Eq. (2.7), can be expressed in matrix notation as follows

$$y_{out} = \hat{h}(\mathcal{L})y, \tag{2.8}$$

where

$$\hat{h}(\mathcal{L}) = \chi \begin{bmatrix} \hat{h}(\lambda_0) & & 0 \\ & \ddots & \\ 0 & & \hat{h}(\lambda_{N-1}) \end{bmatrix} \chi^T,$$

is a graph filter or kernel defined in the spectral domain of the graph.

Interestingly, when the graph filter is a polynomial of order K with coefficients $\{\alpha_k\}_{k=0}^K$ such that

$$\hat{h}(\lambda_\ell) = \sum_{k=0}^{K} \alpha_k \lambda_\ell^k, \tag{2.9}$$

filtering in the spectral domain of the input signal $y(n)$ at node n can be interpreted as a linear combination of the components of the input signal at vertices that are within a K-hop neighborhood of n. Combining Eqs. (2.8), (2.9), we obtain that

$$\begin{aligned} y_{out}(n) &= \sum_{\ell=0}^{N-1} \hat{y}(\lambda_\ell)\hat{h}(\lambda_\ell)\chi_\ell(n) \\ &= \sum_{m=1}^{N} y(m) \sum_{k=0}^{K} \alpha_k \sum_{\ell=0}^{N-1} \lambda_\ell^k \chi_\ell^*(m)\chi_\ell(n) \\ &= \sum_{m=1}^{N} y(m) \sum_{k=0}^{K} \alpha_k (\mathcal{L}^k)_{n,m}, \end{aligned} \tag{2.10}$$

where $(\mathcal{L}^k)_{n,m} = 0$ if the shortest-path distance between vertices n and m is greater than k [3]. This property can be quite useful for designing signals that are localized in the vertex domain of the graph. A detailed overview of these basic operations can be found in [9].

2.3.2 Localized graph transforms

The graph Fourier transform is global as most of the eigenvectors of the graph Laplacian are not localized on the vertices of the graph. However, there exist many applications such as analysis of graph signals, source localization, detection, where localization is a desired property. A lot of research has been dedicated to developing localized transform methods specifically designed for analyzing graph signals. In particular, the authors in [16] have extended the classical short-time Fourier transform to a windowed graph Fourier transform, which enables vertex-frequency analysis. Wavelet-like transforms have also received particular attention mainly due to their multiresolution properties that are obtained with the definition of translation and dilation operators on the graph. They can thus be used to analyze graph signals at different vertices and scales. These wavelet construction methods can be divided into two types, the vertex domain designs and the graph spectral domain designs.

The vertex domain designs of graph wavelet transforms are based on the spatial features of the graph, such as the k-hop connectivity between the nodes. Such examples are the multiscale wavelets of [18], which are Haar-like bases for tree graphs; the critically sampled generalized tree-based wavelets transform [19] that extends the transform of [18] to general wavelet filters, and its extension to a redundant wavelet transform [20] that exploits the tree structure of the data to represent signals defined on weighted graphs. In all these methods the construction of the tree is a difficult design problem. In addition, the graph wavelets proposed in [21] have been specifically designed in the vertex domain to analyze computer network traffic by relying on the geodesic or the shortest path distance on the graph. The algorithm however applies only to unweighted graphs. The wavelets on unidirectional routing trees proposed in [22] are extensions of the lifting wavelets, and have been used for distributed data gathering in wireless sensor networks.

The graph spectral domain designs of graph wavelets are based on the spectral features of the graph, which are encoded in the eigenvalues and eigenvectors of the graph Laplacian. Their goal is to design bases that are localized in both the vertex and the graph spectral domain. Examples such as diffusion wavelets [23], spectral graph wavelets [3], and critically sampled two-channel wavelet filter banks [24] target piecewise-smooth graph signals. In particular, the diffusion wavelets [23] are based on compressed representations of the powers of a diffusion operator to capture different resolutions. To make the transform orthogonal, the localized basis functions at each resolution are downsampled and orthogonalized. The spectral graph wavelet [3] is a redundant transform that is designed in precise analogy to the classical continuous time wavelet transform, with the translation and the dilation of band-pass filters defined in the graph spectral domain. The graph wavelet filter banks [24] are inspired by the traditional multiresolution analysis based on filter banks in the Euclidean domain, and they result, under some conditions, in orthogonal filter banks that are critically sampled. While the spectral graph wavelets form a frame, the authors in [25] propose the design of filters that are adapted to the actual distribution of the graph Laplacian eigenvalues and lead to a tight frame. Similar ideas has been used in [26] to design filters that extend the spectral graph wavelets to a tight frame on multilayer graphs. Finally, a multiscale pyramid transform for graph signals that outputs both a multiresolution representation of the graph and a graph signal multiresolution decomposition is proposed in [27]. The algorithm is based on the four fundamental steps of the Laplacian pyramid transform for signals in the Euclidean domain that are graph downsampling, graph reduction, filtering and interpolation of graph signals.

The above transforms feature pre-defined structures merely derived from the graph and some of them can be efficiently implemented; however, they are generally not adapted to the signals at hand. Some exceptions are the diffusion wavelet packets of [28], the wavelets on graphs via deep learning [29], and the tree-based wavelets [19, 20], which feature extra adaptivity. In particular, the diffusion wavelet packets consist of a large number of bases, each adapted to represent functions with different space and frequency localization properties. The choice of the basis from the bases library depends on the task at hand. The authors in [29] introduce a machine learning framework to design a lifting scheme that resembles a deep auto encoder network. Desired properties of the transform are introduced in the training of the neural network. In both works however training signals living on the graph are not taken into account. The definition of the tree in [19, 20] captures the geometry and the structure of the input data and the adaptivity is obtained by permutations derived from the tree. The performance of the scheme however depends on the tree construction and the reordering involved. One of the objectives of this thesis is exactly to design structured graph transforms, which can sparsely represent graph signals and can be efficiently implemented.

2.4 Applications of graph-based signal processing

In this section, we present a few applications of graph-based signal processing. First, we review some of the works from the literature that use graph-based tools to process graph signals in both centralized and distributed settings. Then, we rely on graphs to model the geometrical structure of high-dimensional signals and we focus on multimedia applications that use graph-based transforms in image and video compression, or 3D shape processing.

2.4.1 Processing with graph-based priors

Many of the representation methods of the previous section have been applied to different signal processing tasks such as denoising, semi-supervised learning, and classification. Similar to the traditional Euclidean domain, notions such as smoothness and sparsity have been used as regularizers for solving many inverse graph-based problems in both centralized and distributed settings.

The smoothness of the signals on the graph has been one of the core assumptions in semi-supervised learning with applications in classification, link prediction, and ranking problems. A signal is considered to be smooth on the graph if it exhibits little variations between strongly connected vertices. Typically, the notion of global smoothness $S_p(y)$ of a signal y is defined through the discrete p-Dirichlet norm of y as [30]

$$S_p(y) = \frac{1}{p}\sum_{v\in\mathcal{V}} \|\nabla_v y\|_2^p = \frac{1}{p}\sum_{v\in\mathcal{V}} \Big[\sum_{u\in\mathcal{N}_v} W_{vu}[y(v)-y(u)]^2 \Big]^{\frac{p}{2}}, \tag{2.11}$$

where $\mathcal{N}_v$ denotes the one-hop neighborhood of node v, and W_{vu} is the edge weight between nodes v, u. When $p = 1$, Eq. (2.11) defines the total variation of the signal y on the graph. When $p = 2$, we have a widely-used Laplacian based form of smoothness defined as

$$\begin{aligned} S_2(y) &= \sum_{u,v\in\mathcal{E}} W_{vu}[y(v)-y(u)]^2 \\ &= y^T L y = \sum_{\ell=0}^{N-1} \lambda_\ell \hat{y}(\lambda_\ell)^2. \end{aligned} \tag{2.12}$$

Eq. (2.12) implies that the signal y is smooth, i.e., $S(y)$ is small only if the graph Fourier coefficients corresponding to big eigenvalues are small. This definition of smoothness or similar notions have been imposed as regularizers in the graph-based semi-supervised learning literature, where the goal is to compute the unknown signal entries by exploiting the assumption that the signal values vary slowly between nodes that are connected by strong edges [31], [32], [33], [34]. The extension to more sophisticated regularization techniques has been developed through the definition of kernels on graphs that are typically of the form of the power series of the graph Laplacian [35]. Recently, a framework for active semi-supervised learning based on sampling theory for graph signals has been introduced in [36] and is based on the above notion of smoothness of signals on the graph. Furthermore, we remark that the authors in [37], introduce the quadratic Laplacian based regularizer of Eq. (2.12) to the dictionary learning problem with the purpose of image classification. They improve the classification performance by imposing that the obtained sparse coding coefficients vary smoothly along the geodesics of the manifold that is captured by the graph. Finally, a smooth regularizer based on Eq. (2.12) has been used in [38] for matrix completion on graphs, with applications to recommendation systems.

While smoothness priors have been widely employed, the use of sparse prior for graph signals has been mostly overlooked so far. The reason is that the link between sparsity and signal structure is not well understood in graph settings. However, there are still some works that try to exploit sparsity in learning applications. For example, the sparsity of the Fourier coefficients has been exploited for the reconstruction of bandlimited graph signals in [39]. The authors in [18] construct multi-scale wavelet-like orthonormal bases on hierarchical trees and relate the smoothness of the graph signals with the decay of the coefficients in these bases. The proposed bases have been applied to transductive semi-supervised learning problems. Finally, sparsity of the spectral graph wavelets coefficients has been used as a regularizer for semi-supervised learning in [40] and [41].

For the sake of completeness, we mention a few more works that use graph-based transforms and filters for processing graph signals in a wide range of applications. The authors in [42] have used the graph Fourier transform for matched signal detection through hypothesis testing. Regularization through localized graph filters have been proposed in [43] to perform graph signal interpolation with partial observations. In [44], adaptive graph filtering has been used for multi-resolution classification with application to health monitoring. Spectral graph wavelets have been successfully applied in applications such as community detection [45] and inference on graphs [46]. In particular, in [45], the correlation between wavelets centered at different nodes has been used to define a notion of correlation, and eventually design a hierarchical clustering scheme that finds the best partitioning in communities at each scale. Finally, the authors in [46] analyze the characteristics of graph signals through their spectral representations using wavelets defined on graphs, which enables them to build efficient classifier of mobility patterns in the spectral domain. While in all these applications graph transforms have been shown to provide significant benefits, the design of adaptive transforms that are particularly adapted to the graph signals is expected to improve even more the performance in such applications.

2.4.2 Distributed processing of graph signals

The processing of graph signals in centralized settings has received considerable attention, but less work has been devoted to solving similar tasks in distributed settings like sensor networks. Many distributed processing tasks consider the graph signal to be the result of the application of a linear graph-based operator to an initial input signal. When the signal can be represented as a filtering operation in the vertex domain of the graph, distributed processing of the signal is significantly simplified. More formally, given an initial signal y, every signal y_{out} that can be expressed as filtering of y in the graph vertex domain with a graph operator $P \in \mathbb{R}^{N\times N}$, such that

$$y_{out}(n) = \sum_{m\in\mathcal{N}_n} P_{n,m} y(m), \tag{2.13}$$

can be computed by local exchange of information only within the neighborhood of node n. $P_{n,m}$ is the filtering weight corresponding to the edge between nodes n and m. The operator or graph filter P is then defined according to the model of the signal.

Most of the existing works in such settings focus on reaching distributively an agreement between sensors, using only local communication. In that case, the operator P is a doubly stochastic weight matrix that leads to an output y_{out} that is the average value of the components of the initial signal y. Examples of such operators are the Metropolis and the Laplacian weight matrices [47] defined respectively as:

- Metropolis weights

$$P_{n,m} = \begin{cases} \frac{1}{1+\max\{d(n),d(m)\}}, & \text{if } \{n,m\} \in \mathcal{E} \\ 1 - \sum_{(n,k)\in\mathcal{E}} P_{n,k}, & \text{if } n = m \\ 0, & \text{otherwise,} \end{cases}$$

  where $d(n), d(m)$ denotes the degree of the n^{th} and the m^{th} sensors respectively.

- Laplacian weights
$$P = I - aL$$
where L denotes the Laplacian matrix of the graph $\mathcal{G}$ and the scalar a must satisfy $0 < a < 1/d_{max}$, where d_{max} consists of the maximum degree of the graph.

Among the most common applications, distributed consensus algorithms in both synchronous (average consensus algorithms) [48] and asynchronous versions (gossip algorithms) [49] have been widely used for performing various aggregations tasks in ad-hoc sensor networks. In particular, the authors in [50] solve the problem of distributed classification of multiple observations exploiting average consensus while consensus-based distributed algorithms for SVM training for binary classification have been proposed in [51]. In addition, [52] solves a distributed field estimation problem from compressed measurements while [53] introduces an algorithm for distributed subspace estimation based on average consensus. Gossip algorithms find also numerous applications in problems such as distributed parameter estimation, source localization, distributed compression [49], and decentralized sparse approximation [54].

Distributed average consensus operators are however only a specific case of the general family of graph-based operators. In general, distributed processing of graph signals requires the definition of more sophisticated graph operators P. To that end, the authors in [55] have introduced a special category of linear graph operators called graph Fourier multipliers, which has been eventually extended to generalized graph multiplier operators in [56]. Such operators are defined with respect to a real symmetric positive semi-definite matrix $\Phi = UVU^T$, where U and V are the eigenvectors and the eigenvalues of Φ, and are expressed as

$$P = \sum_{\ell=0}^{N-1} g(V_\ell) U_\ell U_\ell^*, \tag{2.14}$$

where $g(\cdot) : [0 : V_{max}(\Phi)] \rightarrow \mathbb{R}$ is a positive function defined in the spectral domain of the graph. When the matrix Φ is the graph Laplacian matrix then

$$P = \sum_{\ell=0}^{N-1} g(\lambda_\ell) \chi_\ell \chi_\ell^*,$$

which corresponds to a graph Fourier operator. The union of such operators $P = [\chi g_1(\Lambda)\chi^T \ \chi g_2(\Lambda)\chi^T \ ... \ \chi g_S(\Lambda)\chi^T]$ represents the graph Fourier multipliers. From Eq. (2.13), a graph signal y_{out} is then the result of filtering a set of initial signals $y = [y_1; \ y_2; \ ...; \ y_S]$ in the spectral domain with each of the graph Fourier multipliers, such that

$$y_{out} = \sum_{s=1}^{S} \chi g_s(\Lambda) \chi^T y_s. \tag{2.15}$$

An example of a union of graph Fourier multipliers is the spectral graph wavelet transform [3], where each of the multipliers corresponds to a particular scale. An efficient way to apply graph Fourier multipliers in distributed settings is by approximating them with Chebyshev polynomials [3], [55]. In that case, the output signal y_{out} is the linear combination of a set of graph filtering operations (in the vertex domain) of some initial signals on the graph. Such an approximation permits the distributed approximation of y_{out} from the set of initial signals as well as the implementation of the forward and adjoint operators, which can be useful in tasks such as distributed denoising and distributed smoothing, as shown in [55].

A few more distributed processing algorithms of graph signals are based on the above mentioned ideas of graph filtering in the vertex domain. Recently, a distributed least square reconstruction algorithm of bandlimited graph signals has been proposed in [57]. The initial observations are sampled only on a subset of nodes and the algorithm is shown to be efficient in tracking the unobserved data of time-varying graph signals. The distributed graph signal inpainting algorithm of [58] uses a regularizer that minimizes a metric term related to the variation of the signal on the graph. The underlying assumption is that the signal is smooth on the graph. The problem of interpolation of bandlimited graph signals from a few samples is also studied in [43]. The reconstruction is achieved using iterative graph filtering, which can be approximated by polynomials of the graph Laplacian matrix and implemented in distributed settings. Graph filters have also been used to accelerate the convergence of the average consensus algorithm on a sensor graph [59], [60]. Finally, matrix polynomials of a graph-shift operator have been proposed in [61] to design graph filters for distributed linear network operators such as finite-time consensus or analog network coding. Most of all the above mentioned works show the potentials of graph signal processing techniques for distributed tasks, but do not explicitly consider practical aspects such as quantization, which is of significant importance in real word applications.

2.4.3 Graph-based multimedia processing

Apart from processing signals that live on networks, graphs have been used for modeling structured signals that live on other irregular domains. In particular, graph signal processing algorithms have been successfully applied in numerous multimedia applications in order to capture the geometrical structure of complex high-dimensional signals such as images, videos, and 3D data. This type of data provides a promising application domain for the emerging field of graph signal processing.

First, we note that graphs and features based on graphs have recently started to gain attention in the computer vision and shape analysis community mainly due to the fact that the graph Laplacian has been shown to approximate successfully the Laplace-Beltrami operator on a manifold [62], [63], [64]. Spectral features defined on the graph have been successfully applied in a wide variety of shape analysis tasks. The heat kernel signatures [65], their scale-invariant version [66], the wave kernel signatures [67], the optimized spectral descriptors of [68], have already been used in 3D shape processing with applications in graph matching [69] or in mesh segmentation and surface alignment problems [70]. These features have been shown to be stable under small perturbations of the edge nodes of the graph. In all these works however, the descriptors are defined based only on the graph structure, and the information about the attributes of the nodes such as color and 3D positions, if any, is assumed to be introduced in the weights of the graph. Thus, the performance of these descriptors largely depends on the quality of the defined graph.

Signal compression is a second application domain where graph signal processing tools have been applied successfully. Analogously to the classical analog case, the graph Fourier coefficients of a smooth signal decay rapidly [15], making the graph Fourier transform a good candidate for compression. In particular, the graph Fourier transform has been widely used to compress efficiently smooth images. For example, the graph-based Fourier transform has been used in [71] for the compression of image and video signals, as an alternative to the classical discrete cosine transform (DCT). The authors in [72] adapted the graph for maximally smooth signals and optimized the graph Fourier transform for better compression of 3D smooth images. A set of edge-adaptive transforms was presented as an alternative to the standard DCT and used in depth-map coding in [73]. A few steps towards the theoretical analysis of the analogy between the graph Fourier transform and the classical DCT have been taken in [74]. Under a Gaussian Markov Random Field image model, the graph Fourier transform has been shown to be optimal in decorrelating the signal and used for predictive transform coding. Graphs have also been used for compressing multiview images, where the graph is designed by connecting corresponding pixels in different views [75]. In [76] graph-based transforms have been used to code luminance values in GBR. The problem of multiview images of asymmetric quality has been studied in [77], where the construction of a graph from high quality images has led to the enhancement of low quality images. In the same line of works, a graph regularizer that imposes smoothness has been proposed in [78] to enhance the quality of quantized depth images. Thus, graph representations are an interesting tool for compression of image and video signals.

Finally, graph-based transforms have recently been used in computer graphics where the structural organization of 3D objects is captured by a graph. In particular, the authors in [79] represent a moving human body by a sequence of 3D meshes with a fixed and known connectivity represented by a graph. The geometry and the color information have then been considered as time-varying signals on a graph, which are compressed using the graph wavelet filter banks [24]. Graph representations have been also used in [2] to model the structure of 3D point clouds and connect nearby points. The graph Fourier transform, which is equivalent to Karhunen-Loève transform on such graphs, is adopted to decorrelate and eventually compress the point cloud attributes that are treated as signals on the graph.

Although graphs and in particular graph-based transforms have been shown to compress efficiently 2D and 3D signals by removing the spatial correlation across the nodes, the temporal correlation between graph signals that are changing over time is not taken into consideration. Towards that direction, informative graph descriptors could be helpful in determining the variation of signals in sequence of graphs. The design of features that combine both the graph and the signal on the graph could result in more informative and discriminative graph features. Exploiting graphs and well-defined features on graphs for removing temporal redundancy in compression is a challenging research problem that we study in this thesis.

2.5 Summary

Signal processing on graphs is a new and emerging field that is attracting the attention of many different research communities. A significant amount of work has been dedicated to the development of graph-based transforms that can represent and process graph signals. These transforms are mainly designed by leveraging intuitions from Euclidean settings and at the same time incorporating the irregular graph structure. The graph Fourier transform or wavelet-like type of transforms have found applications in different inference problems such as regularization of graph signals for semi-supervised learning, in both centralized and distributed settings. Finally, these transforms have shown some potentials in the efficient storage and compression of high-dimensional graph signals, such as images, videos, and 3D point clouds.

However, since the field is at its infancy, there are still a lot of open questions and research challenges. Existing work focuses only on exploiting the graph for capturing pairwise dependencies between the vertices. The extension of graph-based signal processing tools in capturing temporal correlation between graph signals could bring significant benefits in applications such as predictive coding and is definitely an open research question. In addition, the representation of wider and more complex families of graph signals requires the transition from transformed-based designs that follow a specific model toward overcomplete graph dictionaries that are adapted to the characteristics of the signals. To that end, notions such as sparsity and localization on graphs have to be better understood. There is also room for exploiting signal-adapted representations to solving efficiently graph-based inverse problems in a distributed fashion. In such settings, the effect of quantization that is of paramount importance in realistic networks should be taken into consideration. These are some of the challenges addressed in this thesis.

Chapter 3

Graph-based Compression of Dynamic 3D Point Cloud Sequences

3.1 Introduction

A typical application of graph signal processing relates to 3D data. Dynamic 3D scenes such as humans in motion can now be captured by arrays of color plus depth (or 'RGBD') video cameras [80], and such data is getting very popular in emerging applications such as animation, gaming, virtual reality, and immersive communications. The geometry captured by RGBD camera arrays, unlike computer-generated geometry, has little explicit spatio-temporal structure, and is often represented by sequences of colored point clouds. Frames, which are the point clouds captured at a given time instant as shown in Fig. 3.1, may have different numbers of points, and there is no explicit association between points over time. Performing motion estimation, motion compensation, and effective compression of such data is therefore a challenging task.

In this chapter, we focus on the compression of the 3D geometry and color attributes and propose a novel motion estimation and compensation scheme that exploits temporal correlation in sequences of point clouds[1]. To deal with the large size of these sequences, we consider that the point clouds are voxelized, that is, their 3D positions are quantized to a regular, axis-aligned, 3D grid having a given stepsize. This quantization of the space is commonly achieved by modeling the 3D point cloud sequences as a series of octree data structures [80], [81], [82]. In contrast to polygonal mesh representations, the octree structure exploits the spatial organization of the 3D points, which results in easy manipulations and permits real-time processing of the point cloud data. In more detail, an octree is a tree structure with a predefined depth, where every branch node represents a certain cube volume in the 3D space, which is called a voxel. A voxel containing at least one point is said to be occupied. Although the overall voxel set lies on a regular grid, the set of occupied voxels are non-uniformly distributed in space. To uncover the irregular structure of the occupied voxels inside each frame, we consider voxels as vertices in a graph $\mathcal{G}$, with edges between nearby vertices. Attributes of each voxel n, including 3D position $p(n) = [x, y, z](n)$ and color

---

[1] D. Thanou, P. A. Chou, and P. Frossard. Graph-based compression of dynamic 3D point cloud sequences, *Accepted for publication in IEEE Trans. on Image Proc.*, Dec. 2015.

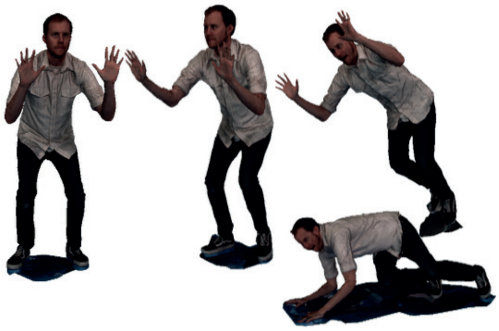
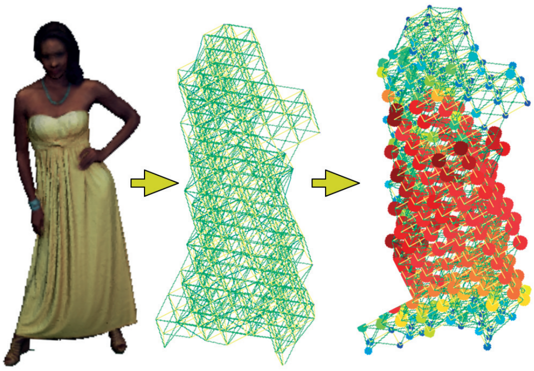


Figure 3.1: Sequence of point cloud frames captured at different time instances in the 'man' sequence.

components $c(n) = [r, g, b](n)$, are treated as signals residing on the vertices of the graph. Such an example is illustrated in Fig. 3.2. As frames in the 3D point cloud sequences are correlated, the graph signals at consecutive time instants are also correlated. Hence, removing temporal correlation implies comparing the signals residing on the vertices of consecutive graphs. The estimation of the correlation is however a challenging task as the graphs usually have different numbers of nodes and no explicit correspondence information between the nodes is available in the sequence.

First, we propose a novel algorithm for motion estimation and compensation in 3D point cloud sequences [83]. We cast motion estimation as a feature matching problem on dynamic graphs. In particular, we compute new local features at different scales with spectral graph wavelets (SGW) [3] for each node of the graph. Our feature descriptors, which consist of the wavelet coefficients of each of the signals placed in the corresponding vertex, are then used to compute point-to-point correspondences between graphs of different frames. We match our SGW features in different graphs with a criterion that is based on the Mahalanobis distance and trained from the data. To avoid inaccurate matches, we first compute the motion on a sparse set of matching nodes that satisfy the matching criterion. We then interpolate the motion of the other nodes of the graph by solving a new graph-based quadratic regularization problem, which promotes smoothness of the motion vectors on the graph in order to build a consistent motion field.

Then, we design a compression system for 3D point cloud sequences, where we exploit the estimated motion information in the predictive coding of the geometry and color information [84]. The basic blocks of our compression architecture are shown in Fig. 3.3. We code the motion field in the graph Fourier domain by exploiting its smoothness on the graph. Temporal redundancy in consecutive 3D positions is removed by coding the structural difference between the target frame and the motion compensated reference frame. The structural difference is efficiently described in a binary stream format as discussed in [1]. Finally, we predict the color of the target frame by interpolating it from the color of the motion compensated reference frame. Only the difference between the actual color information and the result of the motion compensation is actually coded with a state-of-the-art encoder for static octree data [2]. Experimental results illustrate that our motion estimation scheme effectively captures the correlation between consecutive frames. Moreover, introducing motion compensation in compression of 3D point cloud sequences results in significant improvement in terms of rate-distortion performance of the overall system, and in particular in



(a) (b) (c)

Figure 3.2: Example of a point cloud of the ‘yellow dress’ sequence (a). The geometry is captured by a graph (b) and the r component of the color is considered as a signal on the graph (c). The size and the color of each disc indicate the value of the signal at the corresponding vertex.

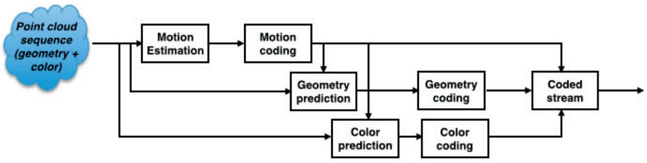
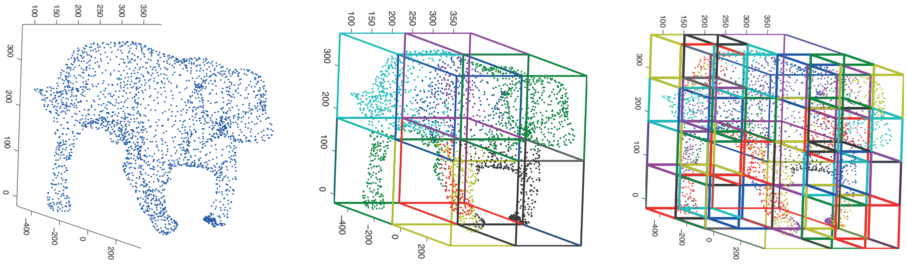


Figure 3.3: Schematic overview of the encoding architecture of a point cloud sequence. Motion estimation is used to reduce the temporal redundancy for efficient compression of the 3D geometry and the color attributes.

the compression of the color attributes where we achieve a gain of up to 10 dB in comparison to state-of-the-art encoders.

The contribution of the chapter is summarized as follows. The proposed encoder is the first one to exploit motion estimation in efficient coding of point cloud sequences, without going first through the expensive conversion of the data into a temporally consistent polygonal mesh. Second, we represent the point cloud sequences as a set of graphs and we solve the motion estimation problem as a new feature matching problem in dynamic graphs. Third, we propose a differential coding scheme for geometry and color compression that provides significant overall gain in terms of rate-distortion performance.

The rest of the chapter is organized as follows. First, in Section 3.2, we describe the representation of 3D point clouds by performing an octree decomposition of the 3D space and we introduce graphs to capture the irregular structure of this representation. The motion estimation scheme is presented in Section 3.3. The estimated motion is then applied to the predictive coding of the geometry and the color in Section 3.4 and experimental results are given in Section 3.5. Finally, in Section 3.6, we review the existing work in the literature that studies the problem of compression of 3D point clouds.



(a) Original point cloud (b) Depth 1 (c) Depth 2

Figure 3.4: Octree decomposition of a 3D model for two different depth levels. The points belonging to each voxel are represented by the same color.

3.2 Structural representation of 3D point clouds

3D point clouds usually have little explicit spatial structure. Someone can however organize the 3D space by converting the point cloud into an octree data structure [80], [81], [82]. In what follows, we recall the octree construction process, and introduce graphs as a tool for capturing the structure of the leaf nodes of the octree.

3.2.1 Octree representation of 3D point clouds

An octree is a tree structure with a predefined depth, where every branch node represents a certain cube volume in the 3D space, which is called a voxel. A voxel containing at least one sample from the 3D point cloud is said to be occupied. Initially, the 3D space is hierarchically partitioned into voxels whose total number depends on the number of 3D volume subdivisions, i.e., the depth of the resulting tree structure. For a given depth, an octree is constructed by traversing the tree structure in depth-first order. Starting from the root, each node can generate eight children voxels. At the maximum depth of the tree, all the points are mapped to leaf voxels. An example of the voxelization of a 3D model for different depth levels, or equivalently for different quantization stepsizes, is shown in Fig. 3.4.

In contrast to temporally consistent polygonal mesh representations, the octree structures are appropriate for modeling 3D point cloud sequences as they are easy to obtain. Thanks to the different depths of the tree, they permit a multiresolution representation of the data that leads to efficient data processing in many applications. In particular, this multiresolution representation permits a progressive compression of the 3D positions of the data, which is lossless within each representation level [1].

3.2.2 Graph-based representation of 3D point clouds

Although the overall voxel set lies on a regular grid, the set of occupied voxels is non-uniformly distributed in space, as most of the leaf voxels are unoccupied. In order to represent the irregular structure formed by the occupied voxels, we use a graph-based representation. Graph-based representations are flexible and well adapted to data that live on an irregular domain [85]. In particular, we represent the set of occupied voxels of the octree using a weighted and undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$, where $\mathcal{V}$ and $\mathcal{E}$ represent the vertex and edge sets of $\mathcal{G}$. Each of the N nodes in $\mathcal{V}$ corresponds to an occupied voxel, while each edge in $\mathcal{E}$ connects neighboring occupied voxels. We define the connectivity of the graph based on the K-nearest neighbors (K-NN) graph, which is widely used in the literature. We usually set K to 26 as it corresponds to the maximum number of neighbors for a node that has a maximum distance of one step along any axis of the 3D space. However, since in general not all 26 voxels are occupied, we extend our construction to the general K-NN graph. Two vertices are thus connected if they are among the 26 nearest neighbors in the voxel grid, which results in a connected graph. This property is useful in the interpolation of the motion vectors, as we see in the following section. The matrix W is a matrix of positive edge weights, with W_{ij} denoting the weight of an edge connecting vertices i and j. This weight captures the connectivity pattern of nearby occupied voxels and is chosen to be inversely proportional to the 3D distance between voxels.

After the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$ is constructed, we consider the attributes of the 3D point cloud — the 3D coordinates $p = [x, y, z]^T \in \mathbb{R}^{3\times N}$ and the color components $c = [r, g, b]^T \in \mathbb{R}^{3\times N}$ — as signals that reside on the vertices of the graph $\mathcal{G}$. A spectral representation of these signals can be obtained with the help of the Graph Fourier Transform (GFT), as defined in Chapter 2. The GFT will be used later to define spectral features and to code effectively data on the graph.

3.3 Motion estimation in 3D point cloud sequences

As the frames have irregular structures, we use a feature-based matching approach to find correspondences in temporally successive point clouds. We use the graph information and the signals residing on its vertices to define feature descriptors on each vertex. We first define simple octant indicator functions to capture the signal values in different orientations. We then characterize the local topological context of each of the point cloud signals in each of these orientations, by using spectral graph wavelets (SGW) computed on the color and geometry signals at different resolutions [3]. Our feature descriptors, which consist of the wavelet coefficients of these signals are then used to compute point-to-point correspondences between graphs of different frames. We select a subset of best matching nodes to define a sparse set of motion vectors that describe the temporal correlation in the sequence. A dense motion field is eventually interpolated from the sparse set of motion vectors to obtain a complete mapping between two frames. The overall procedure is detailed below.

3.3.1 Multi-resolution features on graphs

We define features in each node by computing the variation of the signal values, i.e., geometry and color components, in different parts of its neighborhood. For each node i belonging to the vertex set $\mathcal{V}$ of a graph $\mathcal{G}$, i.e., $i \in \mathcal{V}$, we first define the octant indicator function $o_{k,i} \in \mathbb{R}^N, \forall\ k = [1, 2, ..., 8]$, for the eight octants around the node i. For example, for the first octant it is given as follows

$$o_{1,i}(j) = \mathbf{1}_{\{x(j)\geq x(i), y(j)\geq y(i), z(j)\geq z(i)\}}(j),$$

where $\mathbf{1}_{\{\cdot\}}(j)$ is the indicator function on $j \in \mathcal{V}$, evaluated in a set $\{\cdot\}$ of voxels given by specific 3D coordinates. The first octant indicator function is thus nonzero only in the entries corresponding to the voxels whose 3D position coordinates are bigger than the ones of node i. We consider all possible combinations of coordinates, which results in a total of 2^3 indicator functions for the eight octants around i. These functions provide a notion of orientation of each node in the 3D space with respect to i, which is given by the octree decomposition.

We then compute graph spectral features based on both geometry and color information, by treating their values independently in each orientation. In particular, for each node $i \in \mathcal{V}$ and each geometry and color component $f \in \mathbb{R}^N$, where $f \in \{x, y, z, r, g, b\}$, we compute the spectral graph wavelet coefficients by considering independently the values of f in each orientation k with respect to node i such that

$$\phi_{i,s,o_{k,i},f} =< f \cdot o_{k,i}, \psi_{s,i} >, \tag{3.1}$$

where $k \in \{1, 2, ..., 8\}$, $s \in \mathcal{S} = \{s_1, ..., s_{max}\}$, is a set of discrete scales, and $\cdot$ denotes the pointwise product. The function $\psi_{s,i}$ represents the spectral graph wavelet of scale s placed at that particular node i. We recall that the spectral graph wavelets [3] are operator-valued functions of the graph Laplacian defined as

$$\psi_{s,i} = T_g^s \delta_i = \sum_{\ell=0}^{N-1} g(s\lambda_\ell)\chi_\ell^*(i)\chi_\ell,$$

where $\chi = [\chi_0, \chi_1, ..., \chi_{N-1}]$, and $0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_{(N-1)}$ are the eigenvectors and the corresponding eigevalues of the combinatorial graph Laplacian L. The graph wavelets are determined by the choice of a generating kernel g, which acts as a band-pass filter in the spectral domain, and a scaling kernel h that acts as a lowpass filter and captures the low frequency content. The scaling is defined in the spectral domain, i.e., the wavelet operator at scale s is given by $T_g^s = g(sL)$. Spectral graph wavelets are finally realized through localizing these operators via the impulse δ on a single vertex i. The application of these wavelets to signals living on the graph results in a multi-scale descriptor for each node. We finally define the feature vector $\boldsymbol{\phi}_i$ at node i as the concatenation of the coefficients computed in (3.1) with wavelets at different scales, including the features obtained from the wavelet scaling function, i.e., $\boldsymbol{\phi}_i = [\phi_{i,s,o_{k,i},f},\ \phi_{i,h,o_{k,i},f}] \in \mathbb{R}^{8\times 6\times(|\mathcal{S}|+1)}$, where

$$\phi_{i,h,o_{k,i},f} =< f \cdot o_{k,i}, h(L)\delta_i > .$$

To summarize, we note that our spectral features characterize each vertex by computing the local variation of these signals at different scales. Furthermore, this approach gives us the flexility to consider the signal values in different orientations as discussed above, and makes the descriptor of each node more informative.

3.3.2 Finding correspondences on dynamic graphs

We translate the problem of finding correspondences in two consecutive point clouds or frames of the sequence into finding correspondences between the vertices of their representative graphs. For the rest of this chapter, we denote the sequence of frames as $\mathcal{I} = \{\mathcal{I}_1,\ \mathcal{I}_2,\ ...,\ \mathcal{I}_{max}\}$ and the set of graphs corresponding to each frame as $\mathcal{G} = \{\mathcal{G}_1,\ \mathcal{G}_2,\ ...,\ \mathcal{G}_{max}\}$. For two consecutive frames of the sequence, $\mathcal{I}_t$, $\mathcal{I}_{t+1}$, called also reference and target frame respectively, our goal is to find correspondences between the vertices of their representative graphs $\mathcal{G}_t$ and $\mathcal{G}_{t+1}$. The number of vertices in the respective vertex sets $\mathcal{V}_t$, $\mathcal{V}_{t+1}$ can differ between the graphs and is denoted as N_t and N_{t+1} respectively.

We use the features defined in the previous subsection to measure the similarity between vertices. We compute the matching score between two nodes $m \in \mathcal{V}_t, n \in \mathcal{V}_{t+1}$ as the Mahalanobis distance between the corresponding feature vectors, i.e.,

$$\sigma(m,n) = (\boldsymbol{\phi}_m - \boldsymbol{\phi}_n)^T P (\boldsymbol{\phi}_m - \boldsymbol{\phi}_n), \quad \forall m \in \mathcal{V}_t,\ n \in \mathcal{V}_{t+1}, \tag{3.2}$$

where P is a matrix that characterizes the relationships between the geometry and the color feature components (measured in different units), as well as the contribution of each of the wavelet scales in the matching performance. As a result, if $m \in \mathcal{V}_t$ corresponds to $n \in \mathcal{V}_{t+1}$, $\boldsymbol{\phi}_m$ is a Gaussian random vector with mean $\boldsymbol{\phi}_n$ and covariance P^{-1}, while if m does not correspond to n, $\boldsymbol{\phi}_m$ comes from a very flat (essentially uniform) distribution. Hence the matching score $\sigma(m,n)$ can be considered a log likelihood ratio for testing the hypothesis that m corresponds to n. We learn the positive definite matrix P by estimating the sample inverse covariance matrix from a set of training features that are known to be in correspondence. More precisely, we consider two frames $\mathcal{I}_\alpha$, $\mathcal{I}_\beta$, for which the correspondences are known. For each $m \in \mathcal{V}_\alpha$ corresponding to $n \in \mathcal{V}_\beta$, we compute the error obtained from their feature difference, i.e., $\epsilon_{m,n} = \boldsymbol{\phi}_m - \boldsymbol{\phi}_n$. The error vectors defined by all the corresponding nodes are then used to estimate the sample covariance matrix of the feature differences and subsequently the inverse sample covariance matrix P.

For each node in $\mathcal{G}_{t+1}$, we then use the matching score of Eq. (3.2) to define the best matching node in $\mathcal{G}_t$. In particular, for each $n \in \mathcal{V}_{t+1}$, we define as its best match in $\mathcal{V}_t$, the node m_n with the minimum Mahalanobis distance, i.e.,

$$m_n = \arg\min_{m \in \mathcal{V}_t} \sigma(m,n). \tag{3.3}$$

From the global set of correspondences computed for all the nodes of $\mathcal{V}_{t+1}$, we select a sparse set of significant matches, namely correspondences with best scores. The objective of this selection is to take into consideration only accurate matches and ignore others since inaccurate correspondences are possible in the case of large displacements. We also want to avoid matching points in $\mathcal{I}_{t+1}$ that do not have any true correspondence in the preceding frame $\mathcal{I}_t$. In order to ensure that we keep correspondences in all areas of the 3D space, we cluster the vertices of $\mathcal{G}_{t+1}$ into different regions and we keep only one correspondence, i.e., one representative vertex, per region. Clustering is performed by applying K-means in the 3D coordinates of the nodes of the target frame, where K is usually set to be equal to the target number of significant matches. In order to avoid inaccurate matches, a representative vertex per cluster is included in the sparse set only if its best matching distance given by Eq. (3.3) is smaller than a predefined threshold. This procedure results in detecting a sparse set of vertices n in $\mathcal{V}_{t+1}$, denoted $\mathcal{V}_{t+1}^S \subset \mathcal{V}_{t+1}$, and the set of their correspondences m_n in $\mathcal{V}_t$, $\mathcal{V}_t^S \subset \mathcal{V}_t$. Moreover, our sparse set of matching points tend to represent accurate correspondences that are well distributed spatially.

3.3.3 Computation of the motion vectors

We now describe how we generate a dense motion field from the sparse set of matching nodes $(m_n, n) \in \mathcal{V}_t^S \times \mathcal{V}_{t+1}^S$. Our implicit assumption is that vertices that are close in terms of 3D positions, namely close neighbors in the underlying graph, undergo a similar motion. We thus use the structure of the graph in order to interpolate the motion field, which is assumed to be smooth on the graph.

In more detail, our goal is to estimate the dense motion field $v_t = [v_t(m)]$, for all $m \in \mathcal{G}_t$, using the correspondences $(m_n, n) \in \mathcal{V}_t^S \times \mathcal{V}_{t+1}^S$. To determine $v_t(m)$ for $m = m_n \in \mathcal{V}_t^S$, we use the vector between the pair of matching points (m_n, n),

$$v_t(m_n|n) \overset{\Delta}{=} p_{t+1}(n) - p_t(m_n). \tag{3.4}$$

Here we recall that p_t and p_{t+1} are the 3D positions of the vertices of $\mathcal{G}_t$ and $\mathcal{G}_{t+1}$, respectively. To determine $v_t(m)$ for $m \notin \mathcal{V}_t^S$, we consider the motion field v_t to be a vector-valued signal that lives on the vertices of $\mathcal{G}_t$. Then we smoothly interpolate the sparse set of motion vectors (3.4). The interpolation is performed by treating each component independently. Given the motion values on some of the vertices, we cast the motion interpolation as a regularization problem that estimates the motion values on the rest of the vertices by requiring the motion signal to vary smoothly across vertices that are connected by an edge in the graph. Moreover, we allow some smoothing on the known entries. The reason for that is that the proposed matching scheme does not necessarily guarantee that the sparse set of correspondences, and the estimated motion vectors associated with them, are correct. To limit the effect of motion estimation inaccuracies, for each matching pair $(m_n, n) \in \mathcal{V}_t^S \times \mathcal{V}_{t+1}^S$, we model the matching score in the local neighborhood of $m_n \in \mathcal{V}_t^S$ with a smooth signal approximation. Specifically, for each $n \in \mathcal{V}_{t+1}^S$, we extend the definition (3.4) to all $m \in \mathcal{V}_t$, i.e.,

$$v_t(m|n) = p_{t+1}(n) - p_t(m).$$

Then, for each node that belongs to the two-hop neighborhood of m_n i.e., $m \in \mathcal{N}_{m_n}^2$, we express $\sigma(m, n)$ as a function of the geometric distance of $p_t(m)$ from $p_t(m_n)$, using a second-order Taylor series expansion around $p_t(m)$. That is,

$$\begin{aligned}\sigma(m,n) &\approx \sigma(m_n,n) + (p_t(m) - p_t(m_n))^T M_n^{-1}(p_t(m) - p_t(m_n)) \\ &= \sigma(m_n,n) + (v_t(m|n) - v_t(m_n|n))^T M_n^{-1}(v_t(m|n) - v_t(m_n|n)).\end{aligned} \tag{3.5}$$

For each $n \in \mathcal{V}_{t+1}^S$, we take $\sigma(m, n)$ to be a discrete sampled version of a continuous function $\sigma(v, n)$ where the second order Taylor approximation is

$$\sigma(v,n) \approx \sigma(m_n,n) + (v - v_t(m_n|n))^T M_n^{-1}(v - v_t(m_n|n))).$$

Thus for each $n \in \mathcal{V}_{t+1}^S$, we assume that the matching score with respect to nodes that are in the neighborhood of its best match $m_n \in \mathcal{V}_t^S$ can be well modeled by a quadratic approximation function. We estimate M_n of this quadratic approximation as the normalized covariance matrix of the 3D offsets,

$$M_n = \frac{1}{|\mathcal{N}_{m_n}^2|} \sum_{m \in \mathcal{N}_{m_n}^2} \frac{(p_t(m) - p_t(m_n))(p_t(m) - p_t(m_n))^T}{\sigma(m,n) - \sigma(m_n,n)}.$$

This is motivated by the fact that if

$$\sigma(m,n) - \sigma(m_n,n) = (v_t(m) - v_t(m_n|n))^T M_n^{-1}(v_t(m) - v_t(m_n|n)),$$

then

$$u = \frac{v_t(m) - v_t(m_n|n)}{\sqrt{\sigma(m,n) - \sigma(m_n,n)}}$$

satisfies $1 = u^T M_n^{-1} u$. Hence, u lies in an ellipsoid whose second moment is proportional to M_n. Although there are other ways for computing M_n in (3.5), this moment-matching method is fast and guarantees that M_n is positive semi-definite. Next, we use the covariance matrices of the 3D offsets to define a diagonal matrix $Q \in \mathbb{R}^{3N_t \times 3N_t}$, such that

$$Q = \begin{bmatrix} M_1^{-1} & \cdots & \mathbf{0}_{3\times3} \\ \vdots & \ddots & \vdots \\ \mathbf{0}_{3\times3} & \cdots & M_{N_t}^{-1} \end{bmatrix},$$

where $M_m^{-1} = M_n^{-1}$ if $m = m_n$ for some $n \in \mathcal{V}_{t+1}^S$ and $M_m^{-1} = \mathbf{0}_{3\times3}$ otherwise. The matrix Q captures the second order Taylor approximation of the total matching score as a function of the motion vectors and the 3D geometry coordinates in the neighborhoods of the nodes in $\mathcal{V}_t^S$ and is used to regularize the motion vectors of the known entries in v_t as shown next.

Finally, we interpolate the dense set of motion vectors $v_t{}^*$ by taking into account the covariance of the motion vectors in the neighborhoods around the points that belong to the sparse set $\mathcal{V}_t^S$ and imposing smoothness of the dense motion vectors on the graph

$$v_t{}^* = \underset{v \in \mathbb{R}^{3N_t}}{\arg\min} (v - v_0)^T Q (v - v_0) + \mu \sum_{i=1}^{3} (S_i v)^T L_t (S_i v), \tag{3.6}$$

where $\{S_i\}_{i=1,2,3}$ is a selection matrix for each of the 3D components respectively, and L_t is the Laplacian matrix of the graph $\mathcal{G}_t$. The motion field $v_0 = [v_t(1), v_t(2), \cdots, v_t(N_t)]^T \in \mathbb{R}^{3N_t}$ is the concatenation of the initial motion vectors, with $v_t(m) = \mathbf{0}_{3\times1}$, if $m \notin \mathcal{V}_t^S$. We note that the optimization problem consists of a fitting term that penalizes the excess matching score on the sparse set of matching nodes, and of a regularization term that imposes smoothness of the motion vectors in each of the position components independently. The tradeoff between the two terms is defined by the constant μ. A small μ promotes a solution that is closed to v_0, while a big μ favors a solution that is very smooth. The corresponding optimization problem is convex and it has a closed form solution given by

$$v_t^* = \big(Q + \mu \sum_{i=1}^{3} S_i^T L_t S_i\big)^{-1} Q v_0, \tag{3.7}$$

which can be computed iteratively using MINRES-QLP [86] in large systems. With a slight abuse of notation, we will from now on denote as v_t^* the reshaped motion vectors of dimensionality $3 \times N_t$, where each row represents the motion in one of the three coordinates. Finally, $v_t^*(m) \in \mathbb{R}^3$ denotes the 3D motion vector of node $m \in \mathcal{V}_t$.

3.3.4 Implementation details

The proposed spectral features can be efficiently computed by approximating the spectral graph wavelets with Chebyshev polynomials of degree M, as described in [3]. Given this approximation, the wavelet coefficients at each scale can then be computed as a polynomial of L applied to a graph signal f. The latter can be performed in a way that accesses L only through iterative matrix-vector multiplications. The polynomial approximation can be particularly efficient when the graph is sparse, which is indeed the case of our K-NN graph. Using a sparse matrix representation, the computation cost of applying L to a vector is proportional to the number $|\mathcal{E}|$ of nonzero edges in the graph. The overall computational complexity is $\mathcal{O}(M|\mathcal{E}|+N(M+1)(|\mathcal{S}|+1))$ [3], where $|\mathcal{S}|$ are the number of scales. Moreover, this approximation avoids the need to compute the complete spectrum of the graph Laplacian matrix. Thus, the computational cost of the features can be substantially reduced.

Regarding the computation of correspondences, we note that the motion between consecutive frames is expected to be relatively smooth. We can avoid computing pairwise distances with all the vertices of the reference frame, by only comparing with vertices whose distance in geometry is smaller than a predefined threshold. Moreover, although in our experiments we have used K-means clustering, dividing the space into small blocks could be enough for our purposes. An example is the procedure followed in [7], where for efficiency the octree is divided into smaller blocks containing $k \times k \times k$ voxels, where k is relatively small. Thus, in the case when the number of vertices is big and K-means may not be appropriate for grouping them, the procedure that we describe above can be very efficient and possibly applied in real time.

3.4 Compression of 3D point cloud sequences

We describe now how the above motion estimation can be used to reduce temporal redundancy in the compression of 3D point cloud sequences, as shown in Fig. 3.3. The first frame of the sequence is always encoded using intra-frame coding. For the rest of the frames, we code the motion vectors by transforming them to the graph Fourier domain. We assume that the reference frame has already been sent and is known to the decoder. Coding of the 3D positions is then performed by comparing the structural difference between the target frame ($\mathcal{I}_{t+1}$) and the motion compensated reference frame ($\mathcal{I}_{t,mc}$). Temporal redundancy in color compression is finally exploited by encoding the difference between the target frame and the color prediction obtained with motion compensation.

3.4.1 Coding of motion vectors

We recall that, for each pair of two consecutive frames $\mathcal{I}_t$, $\mathcal{I}_{t+1}$, the sparse set of motion vectors is initially smoothed at the encoder side. The estimated dense motion field is then transmitted to the decoder. We exploit the fact that the graph Fourier transform is suitable for compressing smooth signals [87], [74], by coding the motion vectors in the graph Fourier domain. In particular, since the motion v_t^* is estimated in each of the nodes of the graph $\mathcal{G}_t$, we use the eigenvectors $\chi_t = [\chi_{t,0}, \chi_{t,1}, ..., \chi_{t,N_t-1}]$ of the graph Laplacian operator corresponding to the graph $\mathcal{G}_t$ of the

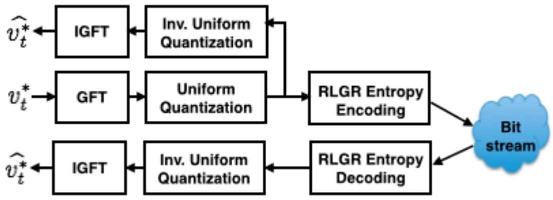


Figure 3.5: Schematic overview of the motion vector coding scheme. The motion vectors v_t^* between two consecutive frames of the sequence are transformed in the graph Fourier domain, quantized uniformly, and sent to entropy coded. The decoder performs the reverse procedure to obtain $\widehat{v_t^*}$.

reference frame, to transform the motion in each of the 3D directions separately such as

$$F_{v_t^*}(\lambda_\ell) =< v_t^*, \chi_{t,\ell} >, \quad \forall \ell = 0, 1, ..., N_t - 1.$$

The transformed coefficients are uniformly quantized as $round(\frac{F_{v_t^*}}{\Delta})$, where Δ is the quantization stepsize that is constant across all the coefficients, and *round* refers to the rounding operation. The quantized coefficients are then entropy coded independently with the adaptive run-length / golomb-rice (RLGR) entropy coder [88] and sent to the decoder. The decoder performs the reverse operations to obtain the decoded motion vectors $\widehat{v_t^*}$. Note that given that the decoder already knows the 3D positions of the reference frame, it can recover the K-NN graph. Thus, the connectivity of the graph does not have to be sent. A block diagram of the encoder and the decoder is shown in Fig. 3.5.

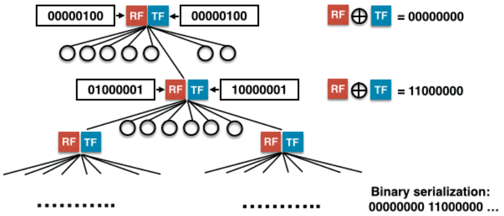
3.4.2 Motion compensated coding of 3D geometry

From the reference frame $\mathcal{I}_t$ and its quantized motion vectors $\widehat{v_t^*}$, both of which are signals on $\mathcal{G}_t$, it is possible to predict the 3D positions of the points in the target frame $\mathcal{I}_{t+1}$, which is a signal on $\mathcal{G}_{t+1}$. Since the two graphs are of different size, a vector space prediction of $\mathcal{I}_{t+1}$ from $\mathcal{I}_t$ is not possible. One can however warp $\mathcal{I}_t$ to $\mathcal{I}_{t+1}$ in order to obtain a warped frame $\mathcal{I}_{t,mc}$ that is close to $\mathcal{I}_{t+1}$. Given that the 3D positions p_t and the decoded motion vectors $\widehat{v_t^*}$ of $\mathcal{I}_t$ are known to both the encoder and the decoder, the position of node m in the warped frame $\mathcal{I}_{t,mc}$ can be estimated on both sides as

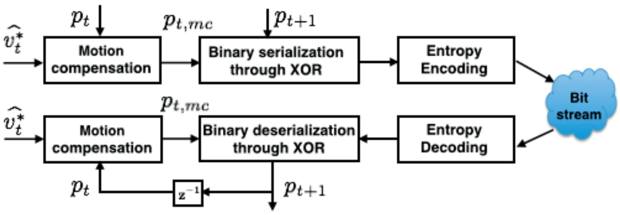
$$p_{t,mc}(m) = p_t(m) + \widehat{v_t^*}(m), \quad \forall m \in \mathcal{V}_t. \tag{3.8}$$

Note that the 3D coordinates of the warped frame $\mathcal{I}_{t,mc}$ remain signals on the graph $\mathcal{G}_t$.

Given the warped frame $\mathcal{I}_{t,mc}$, we use the real-time compression algorithm proposed in [1] to code the structural difference between the 3D positions of $\mathcal{I}_{t+1}$ and $\mathcal{I}_{t,mc}$. Specifically, we assume that the point clouds corresponding to $\mathcal{I}_{t,mc}$ and $\mathcal{I}_{t+1}$ have already been spatially decomposed into octree data structures at a predefined depth. By knowing the occupied voxels of the reference frame $\mathcal{I}_t$ and the motion vectors $\widehat{v_t^*}$, both the encoder and decoder are able to compute the occupied voxels of the motion compensated reference frame $\mathcal{I}_{t,mc}$ and the representative bit indicator function. The encoding of the occupied voxels of the target frame $\mathcal{I}_{t+1}$ is performed by computing the exclusive-OR (XOR) between the indicator functions for the occupied voxels in frames $\mathcal{I}_{t,mc}$ and $\mathcal{I}_{t+1}$. This



(a) Differential encoding of consecutive frames



(b) Schematic overview of the compression architecture

Figure 3.6: Illustration of the geometry compression of the target frame (TF) based on the motion compensated reference frame (RF). (a) Differential encoding of the consecutive frames where structural changes within octree occupied voxels are extracted during the binary serialization process and encoded using the XOR operator. The bit stream of the XOR operator is sent to the decoder. The figure is inspired by [1]. (b) Schematic overview of the overall 3D geometry coding scheme.

can be implemented by an octree decomposition of the set of voxels that are occupied in $\mathcal{I}_{t,mc}$ but not in $\mathcal{I}_{t+1}$, or vice versa, as illustrated in Fig. 3.6(a). Thus, motion compensation is expected to reduce the set difference and hence the number of bits used by the octree decomposition. The decoder can eventually use the motion compensated reference frame and the bits from the octree decomposition to recover exactly the set of occupied voxels (and hence the graph and 3D positions) of the target frame $\mathcal{I}_{t+1}$. We note that the first frame of the sequence is coded based on a static octree coding scheme. A schematic overview of the encoding and decoding architecture is shown in Fig. 3.6(b). A detailed description of the algorithm can be found in the original paper [1].

3.4.3 Motion compensated coding of color attributes

After coding the 3D positions and the motion vectors, motion compensation is used to predict the color of the target frame from the motion compensated reference frame. While the 3D positions $p_{t,mc}$ of the points in the warped frame $\mathcal{I}_{t,mc}$ are based on the 3D positions of the reference frame $\mathcal{I}_t$ and the motion field on the graph $\mathcal{G}_t$ according to (3.8), the colors $c_{t,mc}$ of the warped frame $\mathcal{I}_{t,mc}$ can be transferred directly from $\mathcal{I}_t$ according to

$$c_{t,mc}(m) = c_t(m), \quad \forall m \in \mathcal{V}_t.$$

Unfortunately, the graphs $\mathcal{G}_t$ and $\mathcal{G}_{t+1}$ have different sizes and there is no direct correspondence between their nodes. However, since $\mathcal{I}_{t,mc}$ is obtained by warping $\mathcal{I}_t$ to $\mathcal{I}_{t+1}$, we can use the colors

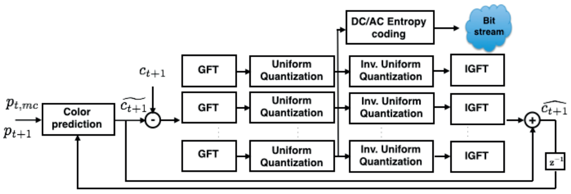


Figure 3.7: Schematic overview of the predictive color coding scheme. The color residual in each block of the octree is projected in the graph Fourier domain. The graph Fourier coefficients are uniformly quantized and entropy coded based on the scheme of [2]. The bit stream is sent to the decoder where the inverse operations are performed to decode the color of the target frame.

of the points in $\mathcal{I}_{t,mc}$ to predict the colors of nearby points in $\mathcal{I}_{t+1}$. To be specific, for each $n \in \mathcal{V}_{t+1}$, we compute a predicted color value $\widetilde{c_{t+1}}(n)$ by averaging the color values of the nearest neighbors NN_n in terms of the Euclidean distance of the 3D positions $p_{t+1}(n)$ and $p_{t,mc}$, i.e.,

$$\widetilde{c_{t+1}}(n) = \frac{1}{|\mathrm{NN}_n|} \sum_{m \in \mathrm{NN}_n} c_{t,mc}(m),$$

where the number of nearest neighbors $|\mathrm{NN}_n|$ is usually set to 3.

Overall, the color coding is implemented as follows. We code the first frame using the coding algorithm of [2]. For the rest of the frames, temporal redundancy in the color information is removed by coding with the graph-based compression algorithm in [2] only the residual of the target frame with respect to the color prediction obtained with the above method, i.e., $\Delta c_{t+1} = c_{t+1} - \widetilde{c_{t+1}}$. The algorithm in [2] is designed for compressing the 3D color attributes in static frames and it essentially removes the spatial correlation within each frame by coding each color component in the graph Fourier domain. The algorithm divides each octree in small blocks containing $k \times k \times k$ voxels. In each of these blocks, it constructs a graph and computes the graph Fourier transform. We adapt the algorithm to point cloud sequences by applying the graph Fourier transform to the color residual Δc_{t+1}. The residuals in each of the three color components are encoded separately. The graph Fourier coefficients are quantized uniformly with a stepsize Δ, as $round(\frac{\chi_{t+1}^T \Delta c_{t+1}}{\Delta})$, where $round$ denotes the rounding operator and χ_{t+1} are the eigenvectors of the graph Laplacian matrix of the corresponding block. The quantized coefficients are then entropy coded, where the structure of the graph is exploited for better efficiency. More details about the color coding scheme are given in [2] and a schematic overview is given in Fig. 3.7. Finally, we recall that, while the algorithm was originally used for coding static frames, in this chapter we use it for coding the residual of the target frame from the motion compensated reference frame. The algorithm however remains a valid choice as the statistical distributions are carefully adapted to the actual signal characteristics.

3.5 Experimental results

We illustrate in this section the matching performance of our motion estimation scheme and the performance of the proposed compression scheme. We use three different sequences that capture

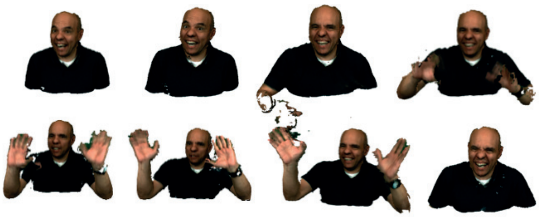


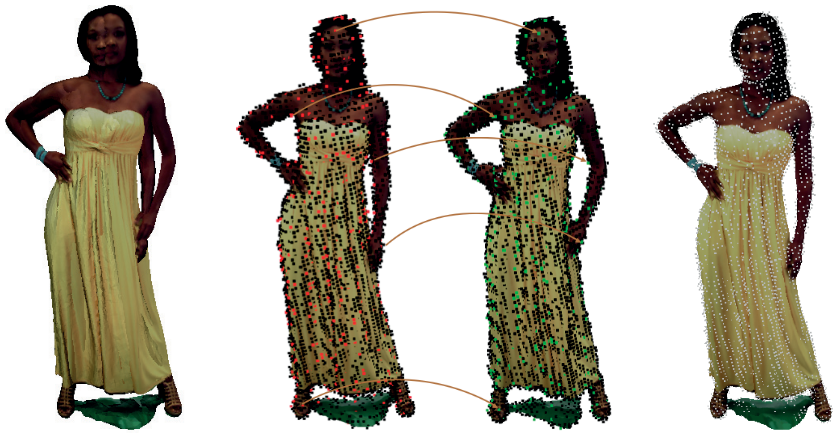
Figure 3.8: Illustrative frames of the upper body sequence.

human bodies in motion, i.e., the yellow dress (see Fig. 3.2) and the man (see Fig. 3.1) sequences, which have been captured according to [89] and voxelized to resemble data collected by the real-time high resolution sparse voxelization algorithm [80], and a human upper body sequence, which has been captured according to [80] (see Fig. 3.8). The first sequence consists of 64 frames, the second one of 30 frames, and the third one of 63. The latter sequence, illustrated in Fig. 3.8, is more noisy and incomplete. We voxelize the point cloud of each frame in these sequences to an octree with a depth of seven. The depth of the octree acts as a sort of quantization of the 3D space. However, our motion estimation and compression scheme can be applied to any other octree level, with similar performance.

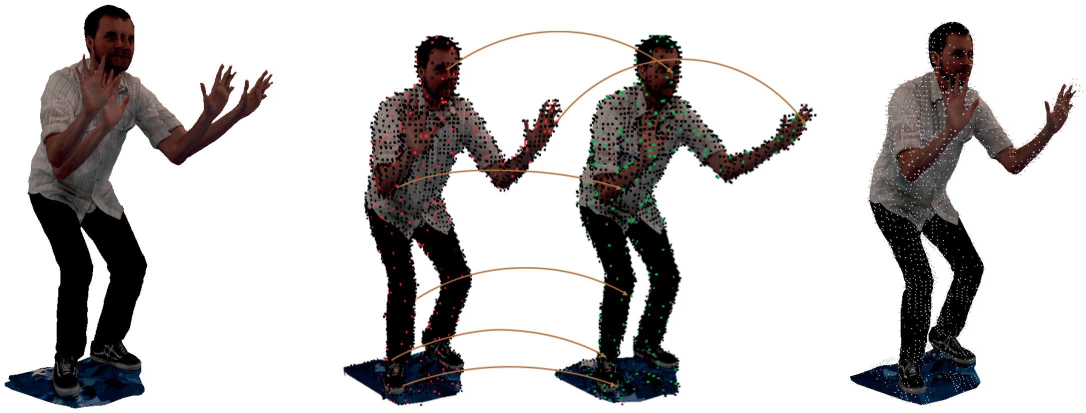
3.5.1 Motion estimation

We first illustrate the performance of our motion estimation algorithm by studying its effect in motion compensation experiments. We select two consecutive frames for each sequence, namely the reference ($\mathcal{I}_t$) and the target frame ($\mathcal{I}_{t+1}$). The graph for each frame is constructed as described in Section 3.2. We define spectral graph wavelets of 4 scales on these graphs, and for computational efficiency, we approximate them with Chebyshev polynomials of degree 30 [3]. We select the number of representative feature points to be around 500, which corresponds to fewer than 10% of the total occupied voxels, and we compute the sparse motion vectors on the corresponding nodes by spectral matching. We estimate the motion on the rest of the nodes by smoothing the motion vectors on the graph according to Eq. (3.6).

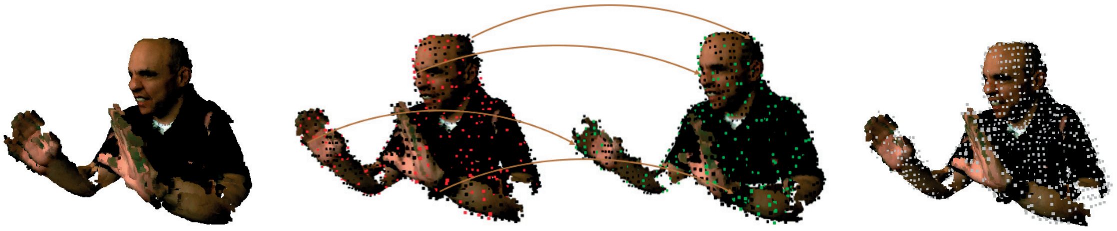
In Figs. 3.9(a), 3.9(d), 3.9(g) we superimpose the reference and the target frames for the yellow dress, the man, and the upper body sequences accordingly in order to illustrate the motion involved between two consecutive frames. The key points used for spectral matching in each of the two frames are shown in Figs. 3.9(b), 3.9(e), 3.9(h), and they are represented in red for the target and in green for the reference frame. For the sake of clarity, we highlight only some of the correspondences used for computing the motion vectors. We observe that the sparse set of matching vertices are accurate and well-distributed in space for both sequences. Finally, in Figs. 3.9(c), 3.9(f), 3.9(i) we superimpose the target frame and the voxel representation of the motion compensated reference frame. By comparing visually these three figures to 3.9(a), 3.9(d), 3.9(g) respectively, we observe that in all the cases the motion compensated reference frame is much closer to the target frame than the reference frame. The result is actually true also for the quite noisy frames of the upper body sequence. The obtained results confirm that our algorithm is able to estimate accurately the motion even in pretty adverse conditions.



(a) $\mathcal{I}_t + \mathcal{I}_{t+1}$ (b) Correspondence between $\mathcal{I}_t$ and $\mathcal{I}_{t+1}$ (c) $\mathcal{I}_{t,mc} + \mathcal{I}_{t+1}$



(d) $\mathcal{I}_t + \mathcal{I}_{t+1}$ (e) Correspondence between $\mathcal{I}_t$ and $\mathcal{I}_{t+1}$ (f) $\mathcal{I}_{t,mc} + \mathcal{I}_{t+1}$



(g) $\mathcal{I}_t + \mathcal{I}_{t+1}$ (h) Correspondence between $\mathcal{I}_t$ and $\mathcal{I}_{t+1}$ (i) $\mathcal{I}_{t,mc} + \mathcal{I}_{t+1}$

Figure 3.9: Example of motion estimation and compensation in the yellow dress, man and upper body motion sequences. The superimposition of the reference ($\mathcal{I}_t$) and target frame ($\mathcal{I}_{t+1}$) is shown in (a), (d), and (g) while in (b), (e), (h) we show the correspondences between the target (red) and the reference frame (green). The superposition of the motion compensated reference frame ($\mathcal{I}_{t,mc}$) and the target frame ($\mathcal{I}_t$) is shown in (c), (f), and (i). Each small cube corresponds to a voxel in the motion compensated frame.

3.5.2 3D geometry compression

We now study the benefits of motion estimation in the compression of geometry in 3D point cloud sequences. The compressed geometry information includes motion vectors and the geometry

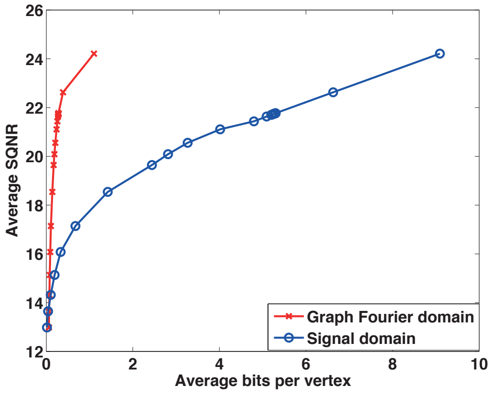
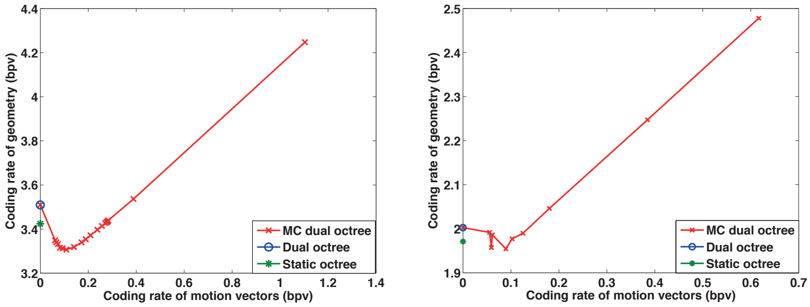


Figure 3.10: Performance comparison of the average signal-to-quantization noise ratio (SQNR) versus bits per vertex (bpv) for coding the motion vectors in the graph Fourier domain and in the signal domain.

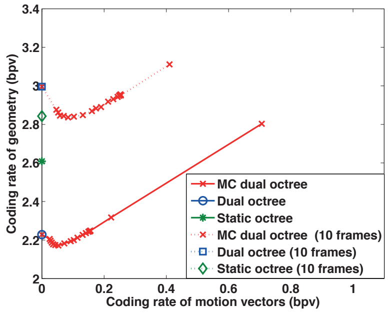
difference between the target frame and the motion compensated reference frame captured by the XOR encoded information. We note that the compression is performed on the whole sequence. The frames of the sequences are coded sequentially in the following way. Only the first frame is coded independently using a classical octree compression scheme based on children pattern sequence [90], while all the other frames are coded by using as a reference frame the previously coded frame. We first code the motion vectors with the proposed coding scheme of Sec. 3.4.1. The motion signal in each of the 3D directions is coded separately.

In Fig. 3.10 we first show the advantage of transforming the motion vectors in the graph Fourier domain, in comparison to coding directly in the signal domain, for the man sequence. Different stepsizes for uniform quantization are used to obtain different coding rates, hence different accuracies of the motion vectors. The performance is measured in terms of the signal-to-quantization noise ratio (SQNR) for a fixed number of bits per vertex. The SQNR is computed on pairs of frames. Each point in the rate distortion curve corresponds to the average over 64 frame pairs. The results confirm that coding the motion vectors in the graph Fourier domain results in an efficient spatial decorrelation of the motion signals, which brings significant gain in terms of coding rate. Similar results hold for the other two sequences.

We study next the effect of motion compensation in the coding rate of the 3D positions. We recall that for a particular depth of the tree, the coding of the geometry is lossless. There exists however a tradeoff between the overall coding rate of the geometry and the coding rate of the motion vectors as we illustrate next. In particular, we compare the motion compensated dual octree scheme as described in Sec. 3.4.2, to the dual octree scheme of [1], and the static octree compression algorithm [90]. In Fig. 3.11, we illustrate the coding rate of the geometry with respect to the coding rate of the motion vectors, measured in terms of the average number of bits per vertex (bpv) over all the frames, for each of the three competitive schemes. The coding rate of the geometry includes the coding rate of the motion vectors. In Fig. 3.11(a), the smallest coding rate of the geometry (3.3 bpv) for the man sequence is achieved for a coding rate of the motion vectors of only 0.1 bpv. The latter indicates that coarse quantization of the motion vectors is



(a) Man sequence (b) Upper body sequence



(c) Yellow dress sequence

Figure 3.11: Effect of the coding rate of the motion vectors on the overall coding rate of the geometry for the motion compensated dual octree algorithm. By sending the motion vectors at low bit rate (≈ 0.1 bpv), the motion compensated dual octree scheme performs slightly better than the static octree and the dual octree compression algorithm.

enough for an efficient geometry compression. A smaller number of bits per vertex however tends to penalize the effect of motion compensation, giving an overall coding rate that approaches the one of the dual octree compression scheme. Of course, a finer coding of the motion vectors increases the overhead in the total coding rate of the geometry. The corresponding numbers for the static octree and the dual octree compression scheme are approximately 3.42 and 3.5 respectively. These results indicate that the temporal structure captured by the dual octree compression scheme is not sufficient to improve the coding rate with respect to the static octree compression algorithm. Motion compensation is thus needed to remove the temporal correlation. However, the overall gain that we obtain is small and corresponds to 3.5% bpv and 5.7% bpv with respect to the static octree and the dual octree compression algorithm respectively. Moreover, motion compensation does not seem to bring a significant gain in the coding of the geometry of the upper body sequence in Fig. 3.11(b). As we already mentioned before this sequence contains frames that are quite noisy. As a result, the performance of motion compensation seems to deteriorate, especially in the case of consecutive frames with appearing or disappearing nodes.

In order to study the effect of the motion in the compression performance, we perform two different tests in the yellow dress sequence. In the first test, we compress the entire yellow dress sequence, which is a low motion sequence. In the second test, we sample the sequence by keeping only 10 frames that are characterized by higher motion between consecutive frames. We then compress the geometry for this new smaller sequence. In Fig. 3.11(c), we observe that when the motion is low, the motion compensated dual octree and the dual octree compression algorithms are much more efficient in coding the geometry in comparison to the static octree compression algorithm. Moreover, the motion compensated dual octree scheme requires a slightly smaller number of bits per vertex (2.2 bpv), for a coding rate of the motion vectors of 0.1 bpv. The coding rate for the dual octree and the static octree compression algorithm are respectively 2.24 and 2.6 bpv. On the other hand, the static octree compression scheme outperforms the dual octree compression algorithm in the higher motion sequence of 10 frames, with coding rates of 3 and 2.8 bpv respectively. The motion compensated dual octree compression algorithm can close the gap between these two methods by achieving a coding rate of 2.8 bpv. We note that this performance is achieved for an overhead of 0.15 bpv for coding the motion vectors. Due to this overhead, the performance of the static octree and the motion compensated dual octree compression algorithm are relatively close.

3.5.3 Color compression

In the next set of experiments, we use motion compensation for color prediction, as described in Section 3.4.3. That is, using the smoothed motion field, we warp the reference frame $\mathcal{I}_t$ to the target frame $\mathcal{I}_{t+1}$, and predict the color of each point in $\mathcal{I}_{t+1}$ as the average of the three nearest points in the warped frame $\mathcal{I}_{t,mc}$. We fix the coding rate of the motion vectors to 0.1 bpv and, for the sake of comparison, we compute the signal-to-noise ratio (SNR) after predicting the color in the following three different ways: (i) the colors of points in the target frame are predicted from their nearest neighbors in the warped frame $\mathcal{I}_{t,mc}$, (SNR_{mc}) (ii) the colors of points in the target frame are predicted from their nearest neighbors in $\mathcal{I}_t$ ($\mathrm{SNR}_{previous}$), and (iii) the colors of points in the target frame are predicted as the average color of all the points in $\mathcal{I}_t$ (SNR_{avg}). The SNR for frame $\mathcal{I}_{t+1}$ is defined as $\mathrm{SNR} = 20\log_{10}\frac{\|c_{t+1}\|}{\|c_{t+1}-\widetilde{c_{t+1}}\|}$, where we recall that c_{t+1} and $\widetilde{c_{t+1}}$ are the actual color and the color prediction respectively. The prediction error is measured by taking pairs of frames in the sequence and computing the average over all the pairs. The obtained values are shown in Table 4.1. We notice that for three sequences motion compensation can significantly reduce the prediction error, by obtaining an average gain in the color prediction of 2.5 dB and 8-10 dB with respect to simple prediction based on the color of the nearest neighbors in the reference frame, and the average color of the reference frame respectively.

Table 3.1: Color Prediction Error (SNR in dB)

Sequence	SNR_{mc}	$\mathrm{SNR}_{previous}$	SNR_{avg}
Yellow dress	17	15	6.5
Man	13	10.5	4
Upper body	9.8	7.5	1.2

We finally use the prediction obtained from our motion estimation and compensation scheme to build a full scheme for color compression, that is based on a prediction path of a series of frames. Compression of color attributes is obtained by coding the residual of the target frame with respect

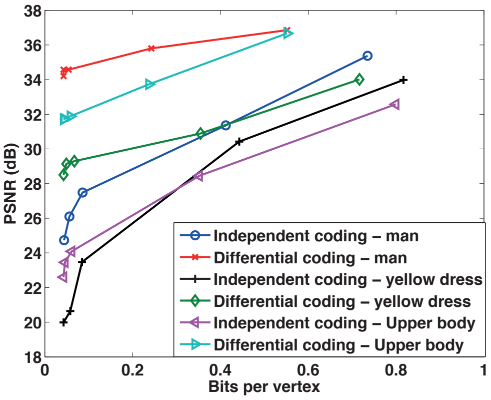
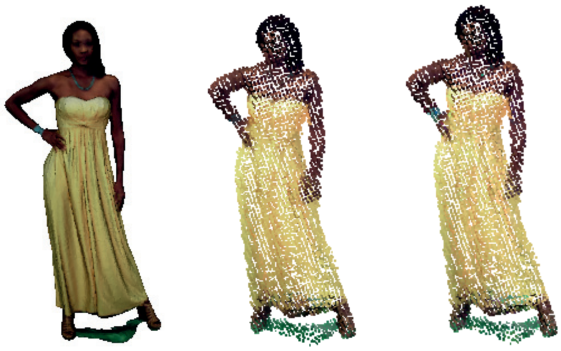


Figure 3.12: Color compression performance (dB) vr. bits per vertex for independent and differential coding on the three datasets for a quantization stepsize of $\Delta = [32,\ 64,\ 256,\ 512,\ 1024]$.

to the color prediction obtained with the scheme described in Section 3.4.3. In our experiments, we code the color in small blocks of $16 \times 16 \times 16$ voxels. We measure the PSNR obtained for different levels of the quantization stepsize in the coding of the color information, hence different coding rates, for both independent [2] and differential coding. The results for the three datasets are shown in Fig. 3.12. Each point on the curve corresponds to the average PSNR of the RGB components across the first ten frames of each sequence, obtained for a quantization stepsize of $\Delta = [32, 64, 256, 512, 1024]$ respectively. We observe that at low bit rate ($\Delta = 1024$), differential coding provides a gain with respect to independent coding of approximately 10 dB for all the three sequences. On the other hand, at high bit rate, the difference between independent and differential coding tends to become smaller, as both methods can code the color quite accurately. Examples of the decoded frames of the yellow dress sequence for $\Delta = 32, 1024$ are shown in Fig. 3.13. Finally, we note that the gain in the coding performance is highly dependent on the length of the prediction path. As the number of predicted frames increases, the accumulated quantization error from the previously coded frames is expected to lead to a gradual PSNR degradation that is more significant at low bit rate. This can be mitigated by periodic insertion of reference frames, and by optimizing the number of predicted frames between consecutive reference frames.

3.5.4 Discussion

Our experimental results have shown that motion compensation is beneficial overall in the compression of 3D point cloud sequences. The main benefit though is observed in the coding of the color attributes, providing a gain of up to 10 dB with respect to coding each frame independently. The gain in the compression of the 3D geometry is only marginal due to the overhead for coding the motion vectors. Moreover, from the experimental validation in our datasets, we observe that the proposed motion compensated geometry compression framework that is based on the differential coding of consecutive octree graph structures is the most expensive part of the overall compression system. Only a very coarse quantization of the motion vectors is sufficient to achieve an overall good compression rate. We expect however the bit rate to increase with the level of the motion.



(a) (b) (c)

Figure 3.13: Rendering results of a point cloud frame from the yellow dress sequence compressed at a quantization stepsize of $\Delta = 1024$, and $\Delta = 32$; (a) original point cloud, (b) voxalized and decoded frame for $\Delta = 1024$, and (c) voxalized and decoded frame for $\Delta = 32$.

Empirically, for each vertex in the man sequence, we need 0.1-0.2 bits to code the motion vectors, 0.1-0.3 bits for the color residual, and 3.3 bits for the geometry compression. Similar observations hold for the other datasets.

3.6 Related work

The direct compression of 3D point cloud sequences has been largely overlooked so far in the literature. A few works have been proposed to compress static 3D point clouds. Some examples include the 2D wavelet transform based scheme of [91], and the subdivision of the point cloud space in different resolution layers using a kd-tree structure [92]. An efficient binary description of the spatial point cloud distribution is performed through a decomposition of the 3D space using octree data structures. The octree decomposition, in contrast to the mesh construction, is quite simple to obtain. It is the basic idea behind the geometry compression algorithms of [82], [81]. The octree structure is also adopted in [2], to compress point cloud attributes. The authors construct a graph for each branch of leaves at certain levels of the octree. The graph transform, which is equivalent to the Karhunen-Loève transform, is then applied to decorrelate the color attributes that are treated as signals on the graph. The proposed algorithm has been shown to remove the spatial redundancy for compression of the 3D point cloud attributes, with significant improvement over traditional methods. Sparse representations in a trained dictionary have further been used in [93] to compress the geometry of 3D point clouds surfaces. Recently, the authors in [94], proposed a novel geometry compression algorithm for large-scale 3D point clouds obtained by terrestrial laser scanners. In their work, the point clouds are converted into a range image and the radial distance in the range image is encoded in an adaptive way. However, all the above methods are designed mainly for static point clouds. In order to apply them to point cloud sequences, we need to consider each frame of the sequence independently, which is clearly suboptimal.

Temporal and spatial redundancy of point cloud sequences has been recently exploited in [1]. The authors compress the geometry by comparing the octree data structure of consecutive point clouds and encoding their structural difference. The proposed compression framework can handle general point cloud streams of arbitrary and varying size, with unknown correspondences. It enables detection and differential encoding of spatial changes within temporarily adjacent octree structures by modifying the octree data structure without computing the exact motion of the voxels. Motion estimation in point cloud sequences can be quite challenging due to the fact that point-to-point correspondences between consecutive frames are not known. While there exists a huge amount of works in the literature that study the problem of motion estimation in video compression, these methods cannot be extended easily to graph settings. In classical video coding schemes, motion in 3-D space is mainly considered as a set of displacements in the regular image plane. Pixel-based methods [95], such as block matching algorithms, or optical and scene flow algorithms, are designed for regular grids. Their generalization to the irregular graph domain is however not straightforward. Feature-based methods [96], such as interest point detection, have also been widely used for motion estimation in video compression. These features usually correspond to key points of images such as corners or sharp edges [97, 98, 99]. With an appropriate definition of features on graphs, these methods can be extended to graphs. To the best of our knowledge though, they have not been adapted so far to estimate the motion on graphs, nor on point clouds. Someone could also apply classical 3D descriptors such as [100, 101, 102, 103, 104, 105] to define 3D features. However, these types of descriptors assume that the point cloud represents a surface, which is not well adapted to the case of graphs. An overview of classical 3D descriptors can be found in [106].

For the sake of completeness, we should mention that 3D point clouds are often converted into polygonal meshes, which can be compressed with a large body of existing methods. In particular, there exists literature for compressing dynamic 3D meshes with either fixed connectivity and known correspondences (e.g., [107, 108, 109, 110, 79]) or varying connectivity (e.g., [111, 112, 113]). A different type of approach consists of the video based methods. The irregular 3D structure of the meshes is parametrized into a rectangular 2D domain, obtaining the so called geometry images [114] in the case of a single mesh and geometry videos [115], [89] in the case of 3D mesh sequences. The mapping of the 3D mesh surface onto a 2D array, which can be done either by using only the 3D geometry information or both the geometry and the texture information [116], allows conventional video compression to be applied to the projected 2D videos. Within the same line of work, emphasis has been given to extending these types of algorithms to handling sequences of meshes with different numbers of vertices and exploiting temporal correlation between them. An example is the recent work in [117], which proposes a framework for compressing 3D human motion oriented geometry videos by constructing key frames that are able to reconstruct the whole motion sequence. Comparing to the mesh-based compression algorithms, the advantage is that the mesh connectivity information (i.e., vertices and faces) does not need to be sent to the decoder, and the complexity is reduced by performing the operations from the 3D to the 2D space. All the above mentioned works however require the conversion process of the point cloud into a mesh in the encoder and the inverse at rendering, which might be computationally expensive. Finally, marching cubes algorithm [118] can be used to extract a polygonal mesh in a fast way, but it requires a “filled" volume. Thus, we believe that octree representations are efficient for modeling temporally changing unorganized point clouds, where input 3D points correspond to sampling of surfaces. In this chapter, we filled a gap in the literature by proposing a new compression algorithm that is based on predictive coding of 3D point cloud sentences modeled as a sequence of octree structures.

3.7 Conclusions

In this chapter, we have proposed a novel graph-based compression framework for 3D point cloud sequences that is based on exploiting temporal correlation between consecutive point clouds. We have first proposed an algorithm for motion estimation and compensation. The algorithm is based on the assumption that 3D models are representable by a sequence of weighted and undirected graphs where the geometry and the color of each model can be considered as graph signals. Correspondence between a sparse set of nodes in each graph is first determined by matching descriptors based on spectral features that are localized on the graph. The motion on the rest of the nodes is interpolated by exploiting the smoothness of the motion vectors on the graph. Motion compensation is then used to perform geometry and color prediction. Finally, these predictions are used to differentially encode both the geometry and the color attributes. Experimental results have shown that the proposed method is efficient in estimating the motion and it eventually provides significant gain in the overall compression performance of the system.

There are a few directions that can be explored in the future. First, it has been shown in our experimental section that a significant part of the bit budget is spent for the compression of the 3D geometry, which given a particular depth of the octree, is lossless. A lossy compression scheme that permits some errors in the reconstruction of the geometry could bring non-negligible benefits in terms of the overall rate-distortion performance. Second, the optimal bit allocation between geometry, color and motion vector data stays an interesting and open research problem, due mainly to the lack of a suitable metric that balances geometry and color visual quality. Third, the estimation of the motion is done by computing features based on the spectral graph wavelet transform. Features based on data-driven dictionaries, such as the ones proposed in the next chapter, are expected to increase significantly the matching, and consequently the compression performance.

Chapter 4

Learning Parametric Dictionaries for Signals on Graphs

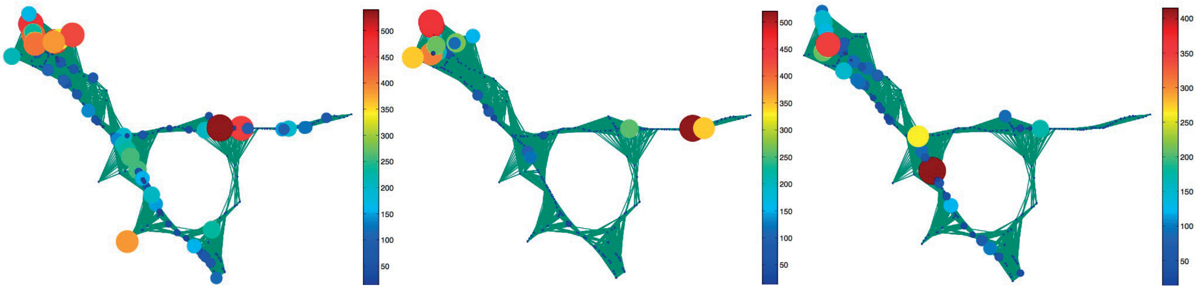
4.1 Introduction

In Chapter 3, we applied graph-based transform representations to extract core features of graph signals, and we used them to perform motion estimation and sequentially efficient compression of temporally correlated 3D signals. In this chapter, we move toward more flexible and data-adapted representations or dictionaries[1]. In particular, we are interested in finding meaningful graph signal representations that (i) capture the most important characteristics of the graph signals, and (ii) are sparse. That is, given a weighted graph and a class of signals on that graph, we want to construct an overcomplete dictionary of atoms that can sparsely represent the graph signals as linear combinations of only a few atoms in the dictionary. An important challenge in the design of dictionaries for graph signals is to account for the intrinsic geometric structure of the underlying weighted graph. This is because important signal characteristics such as smoothness depend on the actual topology of the graph on which the signal resides (see, e.g., [9, Example 1]).

For signals in Euclidean domains as well as signals on irregular data domains such as graphs, the choice of the dictionary often involves a tradeoff between two desirable properties – the ability to adapt to specific signal data and a fast implementation of the dictionary [119]. In the *dictionary learning* or *dictionary training* approach for signals in Euclidean domains, numerical algorithms such as K-SVD [4] and the Method of Optimal Directions (MOD) [120] (see [119, Section IV] and references therein) learn a dictionary from a set of realizations of the data (training signals). The learned dictionaries are highly adapted to a given class of signals and therefore usually exhibit good representation performance. Such approaches can certainly be applied to graph signals, which can be viewed as vectors in $\mathbb{R}^N$. However, the learned dictionaries will neither feature a fast implementation, nor explicitly incorporate the underlying graph structure. On the other hand,

---

[1] Parts of this chapter have been published in :
D. Thanou, D. I Shuman, and P. Frossard. Learning parametric dictionaries for signals on graphs, *IEEE Trans. on Signal Proc.*, vol. 62, no. 15, pp. 3849-3862, Aug. 2014.
D. Thanou and P. Frossard. Multi-graph learning of spectral graph dictionaries, in *Proc. of IEEE Inter. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*, Brisbane, Australia, Apr. 2015



(a) Day 1 (b) Day 2 (c) Day 3

Figure 4.1: Illustrative example: The three signals on the graph are the minutes of bottlenecks per day at different detector stations in Alameda County, California, on three different days. The detector stations are the nodes of the graph and the connectivity is defined based on the GPS coordinates of the stations. The size and the color of each ball indicate the value of the signal at each vertex of the graph. Note that all signals consist of a set of localized features positioned on different nodes of the graph.

analytic dictionaries based on signal transforms such as the graph Fourier transform, the windowed graph Fourier transform [16], or wavelet-like transforms such as the spectral graph wavelets [3] and the diffusion wavelets [23], have pre-defined structures that are derived from the graphs; some of them can even be efficiently implemented. However, they are generally not adapted to the signals at hand. Therefore, their ability to efficiently represent the data depends on the accuracy of the mathematical data model.

The work in [5] tries to bridge the gap between the graph-based transform methods and the purely numerical dictionary learning algorithms by proposing an algorithm to learn structured graph dictionaries. The learned dictionaries have a structure that is derived from the graph topology, while their parameters are learned from the data. However, it does not necessarily lead to efficient implementations as the obtained dictionary is not necessarily a smooth matrix function (see, e.g., [121] for more on matrix functions) of the graph Laplacian matrix.

In this chapter, we capitalize on the benefits of both numerical and analytical approaches by learning a dictionary that incorporates the graph structure and can be implemented efficiently [122], [123]. We model the graph signals as combinations of overlapping local patterns, describing localized events or causes on the graph, which can appear in different vertices. That could be the case in graph signals for traffic data (see Fig. 4.1), brain data, or other type of networks. We incorporate the underlying graph structure into the dictionary through the graph Laplacian operator, which encodes the connectivity. In order to ensure that atoms are localized in the graph vertex domain, we impose the constraint that our dictionary is a concatenation of subdictionaries that are polynomials of the graph Laplacian [3]. We then learn the coefficients of the polynomial kernels via numerical optimization. As such, our approach falls into the category of parametric dictionary learning [119, Section IV.E]. The learned dictionaries are adapted to the training data and have computationally efficient implementation. Experimental results demonstrate the effectiveness of our scheme in the approximation of both synthetic signals and graph signals collected from real world applications. The localization property of the atoms in the graph domain further leads to an easier interpretation of the data from their atomic representations.

Then, we extend our dictionary learning algorithm to graph signals that live on different weighted graphs [124]. The underlying assumption is that, although the signals lie on different topologies, they share some common characteristics, which are expected to be captured by the atoms of the dictionary which are polynomial functions of the graph Laplacian. Under this assumption, the polynomial coefficients define continuous kernels in the spectral domain that can generate signals on different graph topologies. We propose a new algorithm to learn these atoms from signals across all the topologies. We then perform experiments on graph signals that represent common processes on different graphs and show that our dictionary learning method is able to recover the core components of these signals. We finally confirm the performance of our algorithm on traffic data, where the learnt dictionary is shown to provide better sparse approximations than non-adaptive representations or graph-structured representations that are optimized on each graph independently.

The structure of the chapter is as follows. We describe the polynomial dictionary structure and the dictionary learning algorithm in Section 4.2. Next, in Section 4.3, we discuss the benefits of the polynomial structure. In Section 4.4, we evaluate the performance of our algorithm on the approximation of both synthetic and real world graph signals. Finally, the extension of the algorithm to multiple graphs and the corresponding experimental results are presented in Section 4.5.

4.2 Parametric dictionary learning on a single graph

Our objective is to learn a structured dictionary that can sparsely represent a set of training signals on a weighted graph. We consider a general class of graph signals that are linear combinations of (overlapping) graph patterns positioned at different vertices on the graph. The latter implies that the signal model is not necessarily the same across all the vertices but it can differ across the different neighborhoods. We aim to learn a dictionary that is capable of capturing all possible translations of a set of patterns. We use the definition of generalized translation on the graph, and we learn a set of polynomial generating kernels (i.e., patterns) that capture the main characteristics of the signals in the spectral domain. Learning directly in the spectral domain enables us to detect components such as atoms that are supported on specific frequency components. In this section, we first introduce translation of polynomial kernels on the graph. Then, we describe in detail the structure of our dictionary and the learning algorithm.

4.2.1 Translation of polynomial graph kernels

Before introducing the structure of our dictionary, we first need to recall the definition of translation on the graph. As discussed in Chapter 2, the generalized translation operator can be defined as a generalized convolution with a Kronecker δ function centered at vertex n [17, 16, 3]:

$$T_n g = \sqrt{N}(g * \delta_n) \overset{(a)}{=} \sqrt{N} \sum_{\ell=0}^{N-1} \hat{g}(\lambda_\ell) \chi_\ell^*(n) \chi_\ell, \tag{4.1}$$

where $\chi = [\chi_0, \chi_1, ..., \chi_{N-1}]$, and $0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_{N-1}$ are respectively the eigenvectors and the eigenvalues of the graph Laplacian. The right-hand side of (4.1) allows us to interpret the generalized translation as an operator acting on the kernel $\hat{g}(\cdot)$, which is defined directly in the graph spectral domain. The localization of $T_n g$ around the center vertex n is controlled by the smoothness of the kernel $\hat{g}(\cdot)$ [3, 16]. One can thus design atoms $T_n g$ that are localized around n in the vertex domain by taking the kernel $\hat{g}(\cdot)$ in (4.1) to be a smooth polynomial function of degree K:

$$\hat{g}(\lambda_\ell) = \sum_{k=0}^{K} \alpha_k \lambda_\ell^k, \quad \ell = 0, ..., N-1. \tag{4.2}$$

Combining (4.1) and (4.2), we can translate a polynomial kernel g to a vertex n in the graph as

$$\begin{aligned} T_n g = \sqrt{N}(g * \delta_n) &= \sqrt{N} \sum_{\ell=0}^{N-1} \sum_{k=0}^{K} \alpha_k \lambda_\ell^k \chi_\ell^*(n) \chi_\ell \\ &= \sqrt{N} \sum_{k=0}^{K} \alpha_k \sum_{\ell=0}^{N-1} \lambda_\ell^k \chi_\ell^*(n) \chi_\ell = \sqrt{N} \sum_{k=0}^{K} \alpha_k (\mathcal{L}^k)_n, \end{aligned} \tag{4.3}$$

where $(\mathcal{L}^k)_n$ denotes the n^{th} column of the matrix $\mathcal{L}^k$. The concatenation of N such columns allows us to generate a set of N localized atoms, which are the columns of

$$Tg = \sqrt{N}\hat{g}(\mathcal{L}) = \sqrt{N}\chi \hat{g}(\Lambda) \chi^T = \sqrt{N} \sum_{k=0}^{K} \alpha_k \mathcal{L}^k, \tag{4.4}$$

where Λ is the diagonal matrix of the eigenvalues. In short, if $\hat{g}(\cdot)$ is a polynomial of degree K, then $(T_n g)(i) = 0$ for all vertices i more than K hops away from the center vertex n; that is, in the vertex domain, the support of the kernel translated to the center vertex n is contained in a ball of K hops from vertex n [3, Lemma 5.2], [16, Lemma 2]. Furthermore, within this ball, the smoothness properties of the polynomial kernel can be used to estimate the decay of the magnitude $|(T_n g)(i)|$ as the distance from n to i increases [16, Section 4.4]. Note that throughout this chapter, we use the normalized graph Laplacian eigenvectors as the Fourier basis in order to avoid some numerical instabilities that arise when taking large powers of the combinatorial graph Laplacian.

4.2.2 Dictionary structure

We use the definition of translation to design a structured graph dictionary $\mathcal{D} = [\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_S]$ that is a concatenation of a set of S subdictionaries of the form

$$\mathcal{D}_s = \widehat{g_s}(\mathcal{L}) = \chi \left(\sum_{k=0}^{K} \alpha_{sk} \Lambda^k \right) \chi^T = \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k, \tag{4.5}$$

where $\widehat{g_s}(\cdot)$ is the generating kernel or pattern of the subdictionary $\mathcal{D}_s$. Note that the atom given by column n of subdictionary $\mathcal{D}_s$ is equal to $\frac{1}{\sqrt{N}} T_n g_s$. It is a polynomial $\widehat{g_s}(\cdot)$ of order K that is translated to the vertex n. The polynomial structure of $\widehat{g_s}(\cdot)$ ensures that the atom given by column n of subdictionary $\mathcal{D}_s$ has its support contained in a K-hop neighborhood of vertex n [3, Lemma 5.2].

The polynomial constraint guarantees the localization of the atoms in the vertex domain, but it does not provide any information about the spectral representation of the atoms. In order to control their frequency behavior, we impose two constraints on the spectral representation of the kernels $\{\widehat{g_s}(\cdot)\}_{s=1,2,...,S}$. First, we require that the kernels are nonnegative and uniformly bounded by a given constant c. In other words, we impose that $0 \leq \widehat{g_s}(\lambda) \leq c$ for all $\lambda \in [0, \lambda_{\max}]$, or, equivalently,

$$0I \preceq \mathcal{D}_s \preceq cI, \quad \forall s \in \{1, 2, ..., S\}, \tag{4.6}$$

where I is the $N \times N$ identity matrix, and the inequality $\preceq$ refers to the eigenvalues of the matrix. Each subdictionary $\mathcal{D}_s$ has to be a positive semi-definite matrix whose maximum eigenvalue is upper bounded by c, a pre-defined constant.

Second, since the classes of signals under consideration usually contain frequency components that are spread across the entire spectrum, the learned kernels $\{\widehat{g_s}(\cdot)\}_{s=1,2,...,S}$ should also cover the full spectrum. We thus impose the constraint $c - \epsilon_1 \leq \sum_{s=1}^{S} \widehat{g_s}(\lambda) \leq c + \epsilon_2$, for all $\lambda \in [0, \lambda_{\max}]$, or equivalently

$$(c - \epsilon_1)I \preceq \sum_{s=1}^{S} \mathcal{D}_s \preceq (c + \epsilon_2)I, \tag{4.7}$$

where ϵ_1, ϵ_2 are small positive constants. Note that both (4.6) and (4.7) are quite generic and do not assume any particular prior on the spectral behavior of the atoms. If we have additional prior information, we can incorporate that prior into our optimization problem by modifying these constraints. For example, if we know that our signals' frequency content is restricted to certain parts of the spectrum, by choosing ϵ_1 close to c, we relax the constraint on the coverage of the entire spectrum, and we give the flexibility to our learning algorithm to learn filters covering only part of it.

Finally, from the constants c, ϵ_1 and ϵ_2, we can derive frame bounds for $\mathcal{D}$, as shown in the following proposition.

Proposition 1. *Consider a dictionary* $\mathcal{D} = [\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_S]$, *where each* $\mathcal{D}_s$ *is of the form of* $\mathcal{D}_s = \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k$. *If the kernels* $\{\widehat{g_s}(\cdot)\}_{s=1,2,...,S}$ *satisfy the constraints* $0 \leq \widehat{g_s}(\lambda) \leq c$ *and* $c - \epsilon_1 \leq \sum_{s=1}^{S} \widehat{g_s}(\lambda) \leq c + \epsilon_2$, *for all* $\lambda \in [0, \lambda_{\max}]$ *then the set of atoms* $\{d_{s,n}\}_{s=1,2,...,S,n=1,2,...,N}$ *of* $\mathcal{D}$ *form a frame. Namely, for every signal* $y \in \mathbb{R}^N$,

$$\frac{(c - \epsilon_1)^2}{S} \|y\|_2^2 \leq \sum_{n=1}^{N} \sum_{s=1}^{S} |\langle y, d_{s,n} \rangle|^2 \leq (c + \epsilon_2)^2 \|y\|_2^2.$$

Proof: From [25, Lemma 1], which is a slight generalization of [3, Theorem 5.6], we have

$$\sum_{n=1}^{N} \sum_{s=1}^{S} |\langle y, d_{s,n} \rangle|^2 = \sum_{\ell=0}^{N-1} |\hat{y}(\lambda_\ell)|^2 \sum_{s=1}^{S} |\widehat{g_s}(\lambda_\ell)|^2, \quad \forall \lambda \in \sigma(\mathcal{L}). \tag{4.8}$$

From the constraints on the spectrum of kernels $\{\widehat{g}_s(\cdot)\}_{s=1,2,...,S}$ we have

$$\sum_{s=1}^{S} |\widehat{g}_s(\lambda_\ell)|^2 \leq \left(\sum_{s=1}^{S} \widehat{g}_s(\lambda_\ell)\right)^2 \leq (c+\epsilon_2)^2, \quad \forall \lambda \in \sigma(\mathcal{L}). \tag{4.9}$$

Moreover, from the left side of (4.7) and the Cauchy-Schwarz inequality, we have

$$\frac{(c-\epsilon_1)^2}{S} \leq \frac{\left(\sum_{s=1}^{S} \widehat{g}_s(\lambda_\ell)\right)^2}{S} \leq \sum_{s=1}^{S} |\widehat{g}_s(\lambda_\ell)|^2, \quad \forall \lambda \in \sigma(\mathcal{L}). \tag{4.10}$$

Combining (4.8), (4.9) and (4.10) yields the desired result. ☐

To summarize, the polynomial dictionary $\mathcal{D}$ is a parametric dictionary that depends on the parameters $\{\alpha_{sk}\}_{s=1,2,...,S;\ k=1,2,...,K}$, and the constraints (4.6) and (4.7) can be viewed as constraints on these parameters. They provide some control on the spectral representation of the atoms and the stability of signal reconstruction with the learned dictionary.

4.2.3 Dictionary learning algorithm

Given a set of training signals $Y = [y_1, y_2, ..., y_M] \in \mathbb{R}^{N \times M}$, all living on the weighted graph $\mathcal{G}$, our objective is to learn a graph dictionary $\mathcal{D} \in \mathbb{R}^{N \times NS}$ with the polynomial structure described in Section 4.2.2, which can efficiently represent all the signals in Y as linear combinations of only a few atoms. Since $\mathcal{D}$ has the form (4.5), this is equivalent to learning the parameters $\{\alpha_{sk}\}_{s=1,2,...,S;\ k=1,2,...,K}$ that characterize the set of generating kernels, $\{\widehat{g}_s(\cdot)\}_{s=1,2,...,S}$. We denote these parameters in vector form as $\alpha = [\alpha_1; ...; \alpha_S]$, where α_s is a column vector with $(K+1)$ entries.

Therefore, the dictionary learning problem can be cast as the following optimization problem:

$$\begin{aligned} \underset{\alpha \in \mathbb{R}^{(K+1)S},\ X \in \mathbb{R}^{SN \times M}}{\arg\min} \quad & \left\{ ||Y - \mathcal{D}X||_F^2 + \mu \|\alpha\|_2^2 \right\} \\ \text{subject to} \quad & \|X^m\|_0 \leq T_0, \quad \forall m \in \{1, ..., M\}, \\ & \mathcal{D}_s = \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k, \forall s \in \{1, 2, ..., S\} \\ & 0I \preceq \mathcal{D}_s \preceq cI, \quad \forall s \in \{1, 2, ..., S\} \\ & (c-\epsilon_1)I \preceq \sum_{s=1}^{S} \mathcal{D}_s \preceq (c+\epsilon_2)I, \end{aligned} \tag{4.11}$$

where X^m corresponds to column m of the coefficient matrix X, $\mathcal{D} = [\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_S]$, and T_0 is the sparsity level in the coding of each signal. Note that, in the objective of the optimization problem (4.11), we penalize the norm of the polynomial coefficients α in order to (i) promote smoothness in the learned polynomial kernels, and (ii) improve the numerical stability of the learning algorithm. In practice, a small value of the corresponding weight factor μ is enough to guarantee the stability of the solution while preserving large values in the polynomial coefficients. The value of the parameter c does not affect the frequency behavior nor the localization of the atoms. It simply scales the magnitude of the kernel coefficients. Finally, the values of ϵ_1, ϵ_2 are generally chosen to be arbitrarily small, unless prior information like frequency spread information, imposes other choices.

The optimization problem (4.11) is not convex, but it can be approximately solved in a computationally efficient manner by alternating between the sparse coding and dictionary update steps. We analyze next each of these two steps.

4.2.3.1 Sparse coding step

In the first step, we fix the parameters α (and accordingly fix the dictionary $\mathcal{D}$ via the structure of Eq. (4.5)) and solve

$$\underset{X}{\operatorname{argmin}} ||Y - \mathcal{D}X||_F^2 \text{ subject to } \|X^m\|_0 \leq T_0, \tag{4.12}$$

for all $m \in \{1, ..., M\}$, using orthogonal matching pursuit (OMP) [125], [126], which has been shown to perform well in the dictionary learning literature. Before applying OMP, we normalize the atoms of the dictionary so that they all have a unit norm. This step is essential for the OMP algorithm in order to treat all of the atoms equally. After computing the coefficients X, we renormalize the atoms of our dictionary to recover our initial polynomial structure [127, Chapter 3.1.4] and the sparse coding coefficients in such a way that the product $\mathcal{D}X$ remains constant.

The success of the OMP in the sparse coding step and its relation with the graph structure is analyzed in the Appendix A. However, the computational complexity of OMP increases with the size of the graph. In applications where the computational time is crucial and the graph is sparse, it would be interesting to employ an optimized OMP implementation, or rely on first order methods such as the iterative soft thresholding, by exploiting the polynomial structure of the dictionary. For the numerical examples in this chapter, we generally use OMP to solve the sparse coding step.

4.2.3.2 Dictionary update step

In the second step, we fix the coefficients X and update the dictionary by finding the vector of polynomial coefficients, α, that solves

$$\underset{\alpha \in \mathbb{R}^{(K+1)S}}{\arg\min} \left\{ ||Y - \mathcal{D}X||_F^2 + \mu \|\alpha\|_2^2 \right\} \tag{4.13}$$

$$\begin{aligned}
\text{subject to } \ & \mathcal{D}_s = \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k, \quad \forall s \in \{1, 2, ..., S\} \\
& 0I \preceq \mathcal{D}_s \preceq cI, \quad \forall s \in \{1, 2, ..., S\} \\
& (c - \epsilon_1) I \preceq \sum_{s=1}^{S} \mathcal{D}_s \preceq (c + \epsilon_2) I.
\end{aligned}$$

The optimization problem (4.13) is a quadratic program [128] as it consists of a quadratic objective function and a set of affine constraints. In particular, the objective function is written as

$$\begin{aligned}||Y-\mathcal{D}X||_F^2+\mu\|\alpha\|_2^2 &= \sum_{n=1}^{N}\sum_{m=1}^{M}(Y-\mathcal{D}X)_{nm}^2+\mu\alpha^T\alpha \\ &= \sum_{n=1}^{N}\sum_{m=1}^{M}\left(Y-\sum_{s=1}^{S}\sum_{k=0}^{K}\alpha_{sk}\mathcal{L}^k X_s\right)_{nm}^2+\mu\alpha^T\alpha,\end{aligned} \tag{4.14}$$

where $X_s \in \mathbb{R}^{N\times M}$ denotes the rows of the matrix X corresponding to the atoms in the subdictionary $\mathcal{D}_s$. Let us define the column vector $P_{nm}^s \in \mathbb{R}^{(K+1)}$ as

$$P_{nm}^s = [(\mathcal{L}^0)_{(n,:)}X_{s(:,m)}; (\mathcal{L}^1)_{(n,:)}X_{s(:,m)}; ...; (\mathcal{L}^K)_{(n,:)}X_{s(:,m)}],$$

where $(\mathcal{L}^k)_{(n,:)}$ is the n^{th} row of the k^{th} power of the Laplacian matrix and $X_{s(:,m)}$ is the m^{th} column of the matrix X_s. We then stack these column vectors into the column vector $P_{nm} \in \mathbb{R}^{S(K+1)}$, which is defined as $P_{nm} = [P_{nm}^1; P_{nm}^2; ...; P_{nm}^S]$. Using this definition of P_{nm}, (4.14) can be written as

$$\begin{aligned}||Y-\mathcal{D}X||_F^2+\mu\|\alpha\|_2^2 &= \sum_{n=1}^{N}\sum_{m=1}^{M}(Y_{nm}-P_{nm}^T\alpha)^2+\mu\alpha^T\alpha \\ &= \sum_{n=1}^{N}\sum_{m=1}^{M}Y_{nm}^2-2Y_{nm}P_{nm}^T\alpha+\alpha^T P_{nm}P_{nm}^T\alpha+\mu\alpha^T\alpha \\ &= \|Y\|_F^2-2\left(\sum_{n=1}^{N}\sum_{m=1}^{M}Y_{nm}P_{nm}^T\right)\alpha+\alpha^T\left(\sum_{n=1}^{N}\sum_{m=1}^{M}P_{nm}P_{nm}^T+\mu I_{S(K+1)}\right)\alpha,\end{aligned}$$

where $I_{S(K+1)}$ is the $S(K+1)\times S(K+1)$ identity matrix. The matrix $\sum_{n=1}^{N}\sum_{m=1}^{M}P_{nm}P_{nm}^T+\mu I$ is positive definite, which implies that our objective is quadratic.

Finally, the optimization constraints (4.6), (4.7) can be expressed as affine functions of α with

$$\begin{aligned}0 \le I_S \otimes B\alpha \le c\mathbf{1}, \\ (c-\epsilon_1)\mathbf{1} \le \mathbf{1}^T \otimes B\alpha \le (c+\epsilon_2)\mathbf{1}\ ,\end{aligned}$$

where the inequalities are component-wise inequalities, $\mathbf{1}$ is the vector of ones, $\otimes$ is the Kronecker product, I_S is the $S\times S$ identity matrix, and B is the Vandermonde matrix

$$B = \begin{bmatrix}1 & \lambda_0 & \lambda_0^2\dots\lambda_0^K \\ 1 & \lambda_1 & \lambda_1^2\dots\lambda_1^K \\ \vdots & \vdots & \vdots \\ 1 & \lambda_{N-1} & \lambda_{N-1}^2\dots\lambda_{N-1}^K\end{bmatrix}.$$

Thus, the coefficients of the polynomials can be found by solving the following quadratic optimization problem:

$$\begin{aligned} \underset{\alpha \in \mathbb{R}^{(K+1)S}}{\arg\min} \ & \alpha^T \left(\sum_{n=1}^{N} \sum_{m=1}^{M} P_{nm} P_{nm}^T + \mu I_{S(K+1)} \right) \alpha - 2 \left(\sum_{n=1}^{N} \sum_{m=1}^{M} Y_{nm} P_{nm}^T \right) \alpha \qquad (4.15) \\ \text{subject to} \ & 0 \leq I_S \otimes B\alpha \leq c\mathbf{1}, \quad \forall s \in \{1, 2, ..., S\} \\ & (c - \epsilon_1)\mathbf{1} \leq \mathbf{1}^T \otimes B\alpha \leq (c + \epsilon_2)\mathbf{1}. \end{aligned}$$

The quadratic problem can be efficiently solved in polynomial time using optimization techniques such as interior point methods [128] or operator splitting methods (e.g., Alternating Direction Method of Multipliers [129]). The former methods lead to more accurate solutions, while the latter are better suited to solve large scale problems.

Algorithm 1 contains a summary of the basic steps of our complete dictionary learning algorithm. We can initialize the dictionary either by generating a set of polynomial kernels that satisfy the constraints imposed in the learning or simply by generating for each kernel $\widehat{g_s}$, a set of discrete values $\widehat{g_s}(\lambda_0), \widehat{g_s}(\lambda_1), \ldots, \widehat{g_s}(\lambda_{N-1})$ uniformly distributed in the range between 0 and c.

Since the optimization problem (4.11) is solved by alternating between the two steps, the polynomial dictionary learning algorithm is not guaranteed to converge to the optimal solution; practically, we observed in most of our experiments that the total representation error $||Y - \mathcal{D}X||_F^2$ either reduced or remained constant over the iterations, which implies that the algorithm tends to converge to a local optimum.

Finally, we note here that for the design of the dictionary, we have used the normalized graph Laplacian eigenvectors as the Fourier basis. Given the polynomial structure of our dictionary, the upper bound of $\lambda_{N-1} \leq 2$ on the spectrum of the normalized Laplacian makes it more appropriate for our framework. The unnormalized Laplacian contains eigenvectors that have similar interpretation in terms of frequency. However, its eigenvalues can have a large magnitude, causing some numerical instabilities when taking large powers in the solution of the optimization problem. Regarding the spectral constraints, we remark that if we alternatively impose that $\sum_{s=1}^{S} |\widehat{g_s}(\lambda)|^2$ is constant for all $\lambda \in [0, \lambda_{\max}]$, the resulting dictionary $\mathcal{D}$ would be a tight frame. However, such a constraint leads to a dictionary update step that is non-convex and requires optimization techniques that are different from the one described in this section.

4.3 Computational efficiency of the learned polynomial dictionary

The structural properties of the proposed class of dictionaries lead to compact representations and computationally efficient implementations, which we elaborate on briefly in this section. First, the number of free parameters depends on the number S of subdictionaries and the degree K of the polynomials. The total number of parameters is $(K+1)S$, and since K and S are small in practice, the dictionary is compact and easy to store. Second, contrary to the unstructured dictionaries learned by algorithms such as K-SVD and MOD, the dictionary forward and adjoint operators can be efficiently applied when the graph is sparse, as is usually the case in practice. Recall from Eq. (4.5) that $\mathcal{D}_s^T y = \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k y$. The computational cost of the iterative sparse matrix-vector multiplication required to compute $\{\mathcal{L}^k y\}_{k=0,2,\ldots,K}$ is $O(K|\mathcal{E}|)$, where $|\mathcal{E}|$ is the cardinality of the

Algorithm 1 Parametric Dictionary Learning on Graphs

```
Input: Signal set $Y$, initial dictionary $\mathcal{D}_{init}$, target signal sparsity $T_0$, polynomial degree $K$, number of subdictionaries $S$, number of iterations $iter$
Output: Sparse signal representations $X$, polynomial coefficients $\alpha$
Initialization: $\mathcal{D} = \mathcal{D}_{init}$
for $i = 1, 2, ..., iter$ do:
    Sparse Approximation Step:
        (a) Scale each atom in $\mathcal{D}$ to a unit norm
        (b) Update $X$ using (4.12)
        (c) Rescale $X$, $\mathcal{D}$ to recover the polynomial structure
    Dictionary Update Step:
        Compute the polynomial coefficients $\alpha$ by solving (4.15), and update the dictionary according to (4.5)
end for
```

edge set of the graph. Therefore, the total computational cost to compute $\mathcal{D}^T y$ is $O(K|\mathcal{E}|+NSK)$. We further note that, by following a procedure similar to the one in [3, Section 6.1], the term $\mathcal{D}\mathcal{D}^T y$ can also be computed in a fast way by exploiting the fact that $\mathcal{D}\mathcal{D}^T y = \sum_{s=1}^{S} \widehat{g_s}^2(\mathcal{L})y$. This leads to a polynomial of degree $K^{'} = 2K$ that can be efficiently computed. Both operators $\mathcal{D}^T y$ and $\mathcal{D}\mathcal{D}^T y$ are important components of most sparse coding techniques. In turn, these efficient implementations are therefore useful in numerous signal processing tasks, and comprise one of the main advantages of learning structured parametric dictionaries. For example, to find sparse representations of different signals with the learned dictionary, rather than using OMP, we can use iterative soft thresholding [130] to solve the lasso regularization problem [131]. The two main operations required in iterative soft thresholding, $\mathcal{D}^T y$ and $\mathcal{D}^T\mathcal{D}x$, can both be approximated by the Chebyshev approximation method of [3], as explained in more detail in [55, Section IV.C]. The same procedure could be applied to compute efficiently the forward and adjoint operators of the dictionary learned in [5]. In that case however, we need to first approximate the discrete values of the kernel with a polynomial function, which as we will see in the experimental section, can deteriorate the approximation performance.

4.4 Experimental results

In the following experiments, we quantify the performance of the proposed dictionary learning method in the approximation of both synthetic and real data. First, we study the behavior of our algorithm in the synthetic scenario where the signals are linear combinations of a few localized atoms that are placed on different vertices of the graph. Then, we study the performance of our algorithm in the approximation of graph signals collected from real world applications. In all experiments, we compare the performance of our algorithm to the performance of (i) graph-based transform methods such as the spectral graph wavelet transform (SGWT)[3], (ii) purely numerical dictionary learning methods such as K-SVD [4] that treat the graph signals as vectors in $\mathbb{R}^N$ and ignore the graph structure, and (iii) the graph-based dictionary learning algorithm presented in [5]. The kernel bounds in (4.11), if not otherwise specified, are chosen as $c = 1$ and $\epsilon_1 = \epsilon_2 = 0.01$, and the number of iterations in the learning algorithm is fixed to 25. Moreover, we set $\mu = 10^{-4}$ and we initialize the dictionary by generating for each kernel $\widehat{g_s}$, a set of discrete values $\widehat{g_s}(\lambda_0), \widehat{g_s}(\lambda_1), \dots, \widehat{g_s}(\lambda_{N-1})$ uniformly distributed in the range between 0 and c. Each subdictionary is then set to $\mathcal{D}_s = \chi \widehat{g_s}(\Lambda)\chi^T$. The sparsity level in the learning phase is set to $T_0 = 4$ for all the synthetic experiments. We use the *sdpt3* solver [132] in the *yalmip* optimization toolbox [133] to solve the quadratic problem (4.13) in the learning algorithm (line 10 of Algorithm 1). In order to directly compare the above methods, we always use orthogonal matching pursuit (OMP) for the sparse coding step in the testing phase, where we first normalize the dictionary atoms to a unit norm. Finally, the average normalized approximation error is defined as $\frac{1}{|Y_{test}|}\sum_{m=1}^{|Y_{test}|} \|Y_m - \mathcal{D}X_m\|_2^2/\|Y_m\|^2$, where $|Y_{test}|$ is the cardinality of the testing set.

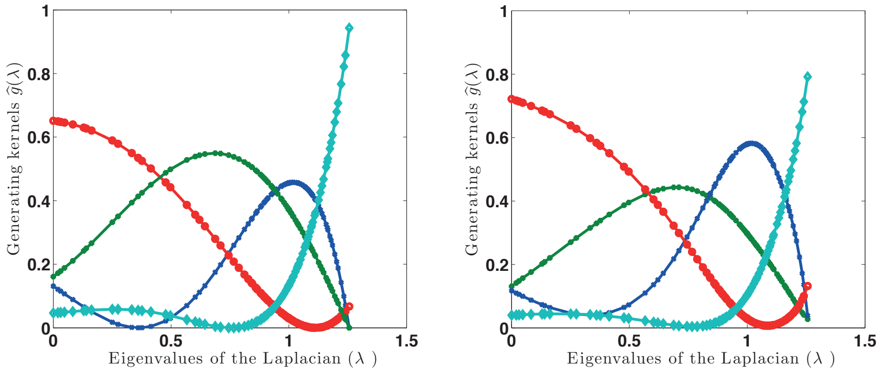
4.4.1 Synthetic signals

We first study the performance of our algorithm for the approximation of synthetic signals. We generate a graph by randomly placing $N = 100$ vertices in the unit square. We set the edge weights based on a thresholded Gaussian kernel function so that $W(i,j) = e^{-\frac{[dist(i,j)]^2}{2\theta^2}}$ if the physical distance between vertices i and j is less than or equal to κ, and zero otherwise. We fix $\theta = 0.9$ and $\kappa = 0.5$ in our experiments, and we keep only connected realizations of the graph.

4.4.1.1 Polynomial generating dictionary

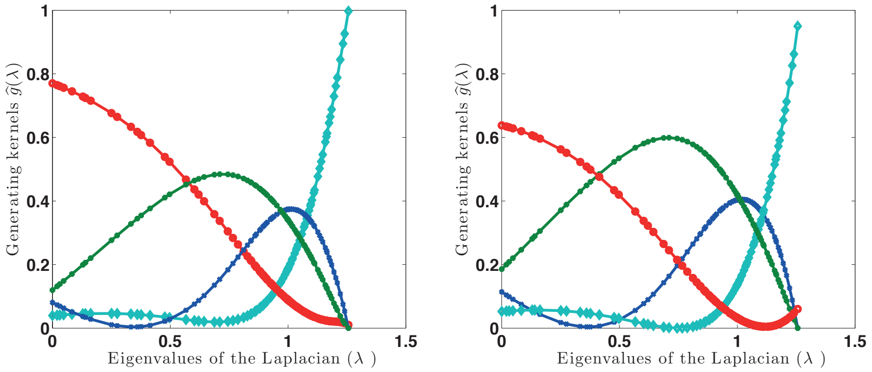
In our first set of experiments, we construct a set of synthetic training signals consisting of localized patterns on the graph, by using a generating dictionary that is a concatenation of $S = 4$ subdictionaries that comply with the constraints of our dictionary learning algorithm. Each subdictionary is a fifth order ($K = 5$) polynomial of the graph Laplacian according to (4.5) and captures one of the four constitutive components of our signal class. The generating kernels $\{\widehat{g_s}(\cdot)\}_{s=1,2,\dots,S}$ of the dictionary are shown in Fig. 4.2(a). We generate the graph signals by linearly combining $T_0 \leq 4$ random atoms from the dictionary with random coefficients. We then learn a dictionary from the training signals, and we expect this learned dictionary to be close to the actual generating dictionary.

We first study the influence of the size of the training set on the dictionary learning outcome. Collecting a large number of training signals can be infeasible in many applications. Moreover, training a dictionary with a large training set significantly increases the complexity of the learning phase, leading to intractable optimization problems. Using our polynomial dictionary learning algorithm with training sets of $M = \{400, 600, 2000\}$ signals, we learn a dictionary of $S = 4$ subdictionaries. To allow some flexibility into our learning algorithm, we fix the degree of the learned polynomials to $K = 20$. Comparing Fig. 4.2(a) to Figs. 4.2(b), 4.2(c), and 4.2(d), we observe that our algorithm is able to recover the shape of the kernels used for the generating dictionary, even with a very small number of training signals. However, the accuracy of the recovery improves as we increase the size of the training set. To quantify the improvement, we define the mean SNR of the learned kernels as $\frac{1}{S}\sum_{s=1}^{S} -20\log(\|\widehat{g_s}(\Lambda) - \widehat{g_s}'(\Lambda)\|_2)$, where $\widehat{g_s}(\Lambda)$ is the true pattern of Fig. 4.2(a) for the subdictionary $\mathcal{D}_s$ and $\widehat{g_s}'(\Lambda)$ is the corresponding pattern learned with



(a) Kernels of the generating dictionary

(b) Learned kernels with $M = 400$



(c) Learned kernels with $M = 600$

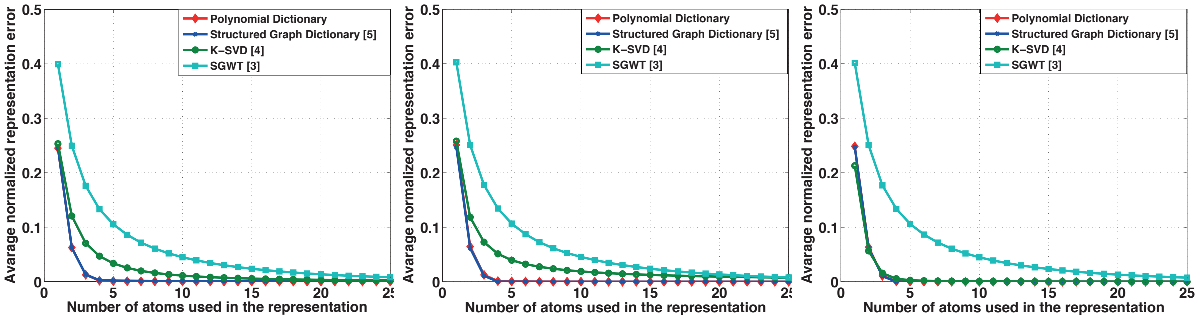
(d) Learned kernels with $M = 2000$

Figure 4.2: Comparison of the generating kernels $\{\widehat{g}_s(\cdot)\}_{s=1,2,\ldots,S}$ (shown in (a)) with the kernels learned by the polynomial dictionary learning algorithm for $M = 400$, $M = 600$ and $M = 2000$ training signals.

our learning algorithm. The SNR values that we obtain are $\{4.9, 5.3, 14.9\}$ for $M = \{400, 600, 2000\}$, respectively.

Next, we generate 2000 testing signals using the same method as for the construction of the training signals. We then study the effect of the size of the training set on the approximation of the testing signals with atoms from our learned dictionary. Fig. 4.3 illustrates the results for three different sizes of the training set and compares the approximation performance to that of other learning algorithms. Each point in the figure is the average of 20 random runs with different realizations of the training and testing sets. We first observe that the approximation performance of the polynomial dictionary is always better than that of SGWT, which demonstrates the benefits of the learning process. The improvement is attributed to the fact that the SGWT kernels are designed *a priori*, while our algorithm learns the shape of the kernels from the data.

We also see that the performance of K-SVD depends on the size of the training set. Recall that K-SVD is blind to the graph structure, and is therefore unable to capture translations of similar patterns. In particular, we observe that when the size of the training set is relatively

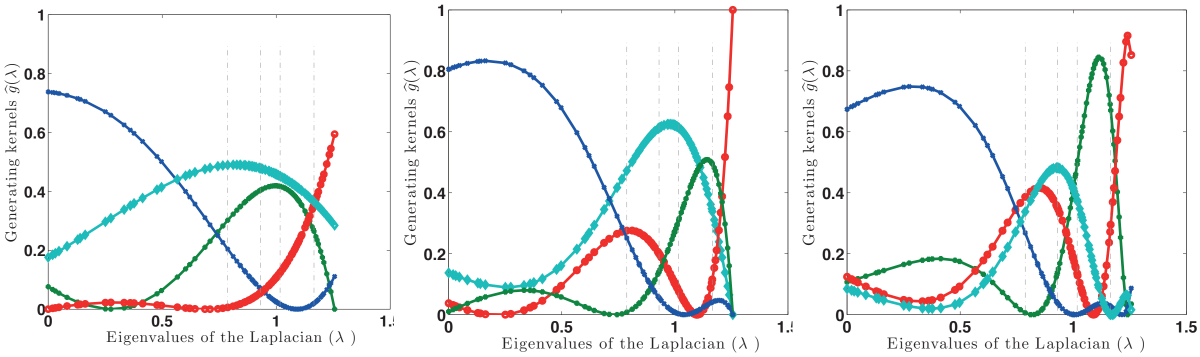


(a) M=400 (b) M=600 (c) M=2000

Figure 4.3: Comparison of the learned polynomial dictionary to the SGWT [3], K-SVD [4] and the graph structured dictionary [5] in terms of approximation performance on test data generated from a polynomial generating dictionary, for different sizes of the training set.

small, as in the case of $M = \{400, 600\}$, the approximation performance of K-SVD significantly deteriorates. It improves when the number of training signals increases (i.e., $M = 2000$). Our polynomial dictionary however shows much more stable performance with respect to the size of the training set. We note three reasons that may explain the better performance of our algorithm, as compared to K-SVD. First, we recall that the number of unknowns parameters for K-SVD is $N^2S = 40000$, while for the polynomial dictionary this number is reduced to $(K+1)S = 84$. Thus, due to the lack of structure, the number of training signals needed for K-SVD usually grows linearly with the size of the dictionary, and is greater than the number needed to effectively train the polynomial dictionary. This fact explains the improved performance of K-SVD with $M = 2000$ training signals. Second, due to the limited size of the training set, K-SVD tends to learn atoms that sparsely approximate the signal on the whole graph, rather than to extract common features that appear in different neighborhoods. As a result, the atoms learned by K-SVD tend to have a global support on the graph, and K-SVD shows poor performance in the datasets containing many localized signals. Third, even when K-SVD does learn a localized pattern appearing in the training data, it does not take into account that similar patterns may appear at other areas of the graph. Of course, as we increase the number of training signals, translated instances of the pattern are more likely to appear in other areas of the graph in the training data, and K-SVD is then more likely to learn atoms containing such patterns in different areas of the graph. On the other hand, our polynomial dictionary learning algorithm learns the patterns in the graph spectral domain, and then includes translated versions of the patterns to all locations in the graph in the learned dictionary, even if some specific instances of the translated patterns do not appear in the training set. Thus, for a smaller number of training examples, our polynomial dictionary shows significantly better performance with respect to K-SVD due to reduced overfitting.

The algorithm proposed in [5] represents some sort of intermediate solution between K-SVD and our algorithm. It learns a dictionary that consists of subdictionaries of the form $\chi\widehat{g_s}(\Lambda)\chi^T$, where the specific values $\widehat{g_s}(\lambda_0), \widehat{g_s}(\lambda_1), \dots, \widehat{g_s}(\lambda_{N-1})$ are learned, rather than learning the coefficients of a polynomial kernel $\widehat{g_s}(\cdot)$ and evaluating it at the N discrete eigenvalues as we do. Thus, the overall number of unknowns of this algorithm is NS, which is usually larger the number required



(a) $K = 5$ (b) $K = 10$ (c) $K = 20$

Figure 4.4: Kernels $\{\widehat{g_s}(\cdot)\}_{s=1,2,3,4}$ learned by the polynomial dictionary algorithm for (a) $K = 5$, (b) $K = 10$, and (c) $K = 20$.

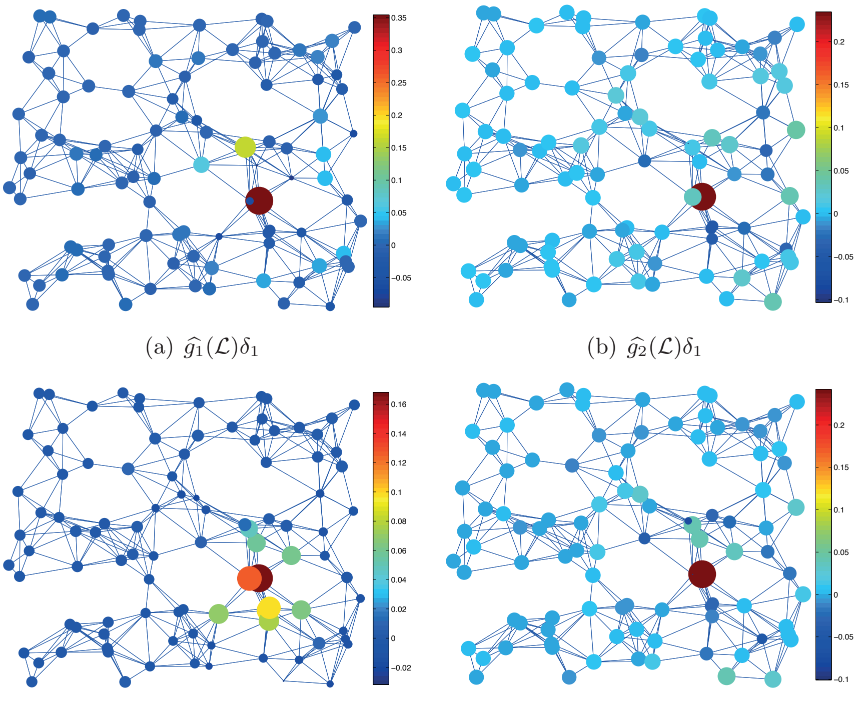
by the polynomial dictionary $((K + 1)S)$ and smaller than that of K-SVD (N^2S). The obtained dictionary is adapted to the graph structure and it contains atoms that are translated versions of the same pattern on the graph. However, the obtained atoms are not in general guaranteed to be well localized in the graph since the learned discrete values of $\widehat{g_s}$ are not necessarily derived from a smooth kernel. Moreover, the unstructured construction of the kernels in the method of [5] leads to more complex implementations, as discussed in Section 4.3.

4.4.1.2 Non-polynomial generating dictionary

In the next set of experiments, we depart from the idealistic scenario and study the performance of our polynomial dictionary learning algorithm in the more general case when the signal components are not exactly polynomials of the Laplacian matrix. In order to generate training and testing signals, we divide the spectrum of the graph into four frequency bands, defined by the eigenvalues of the graph: $[\lambda_0 : \lambda_{24}], [(\lambda_{25} : \lambda_{39}) \cup (\lambda_{90} : \lambda_{99})], [\lambda_{40} : \lambda_{64}]$, and $[\lambda_{65} : \lambda_{89}]$. We then construct a generating dictionary of $J = 400$ atoms, with each atom having a spectral representation that is concentrated exclusively in one of the four bands. In particular, the atom j is of the form

$$d_j = \widehat{h_j}(\mathcal{L})\delta_n = \chi\widehat{h_j}(\Lambda)\chi^T\delta_n. \tag{4.16}$$

Each atom is generated independently of the others as follows. We randomly pick one of the four bands, randomly generate 25 coefficients uniformly distributed in the range $[0, 1]$, and assign these random coefficients to be the diagonal entries of $\widehat{h_j}(\Lambda)$ corresponding to the indices of the chosen spectral band. The rest of the values in $\widehat{h_j}(\Lambda)$ are set to zero. The atom is then centered on a vertex n that is also chosen randomly. Note that the obtained atoms are not guaranteed to be well localized in the vertex domain since the discrete values of $\widehat{h_j}(\Lambda)$ are chosen randomly and are not derived from a smooth kernel. Therefore, the atoms of the generating dictionary do not exactly match the signal model assumed by our dictionary design algorithm, but rather are closer to the signal model assumed by [5]. Finally, we generate the training signals by linearly combining (with



(a) $\widehat{g}_1(\mathcal{L})\delta_1$ (b) $\widehat{g}_2(\mathcal{L})\delta_1$

(c) $\widehat{g}_3(\mathcal{L})\delta_1$ (d) $\widehat{g}_4(\mathcal{L})\delta_1$

Figure 4.5: Learned atoms centered on vertex $n = 1$, from each of the subdictionaries.

random coefficients) $T_0 \leq 4$ random atoms from the generating dictionary.

We first verify the ability of our dictionary learning algorithm to recover the spectral bands that are used in the synthetic generating dictionary. We fix the number of training signals to $M = 600$ and run our dictionary learning algorithm for three different degree values of the polynomial, i.e., $K = \{5, 10, 20\}$. The kernels $\{\widehat{g}_s(\cdot)\}_{s=1,2,3,4}$ obtained for the four subdictionaries are shown in Fig. 4.4 and the boundaries between the different frequency bands are indicated with the vertical dashed lines. We observe that for higher values of K, the learned kernels are more localized in the graph spectral domain and each kernel approximates one of the four bands defined in the generating dictionary, similarly to the behavior of classical frequency filters

In Fig. 4.5, we illustrate the four learned atoms centered at the vertex $n = 1$ (one atom for each subdictionary), with $K = 20$. We can see that the support of the atoms adapts to the graph topology. The atoms can be either smoother around a particular vertex, as for example in Fig. 4.5(c), or more localized, as in Fig. 4.5(a). Comparing Figs. 4.4, and 4.5, we observe that the localization of the atoms in the graph domain depends on the spectral behavior of the kernels. Note that the smoothest atom on the graph (Fig. 4.5(c)) corresponds to the subdictionary generated from the kernel that is concentrated on the low frequencies (i.e., $\widehat{g}_3(\cdot)$). This is because the graph Laplacian eigenvectors associated with the lower frequencies are smoother with respect to the underlying graph topology, while those associated with the larger eigenvalues oscillate more

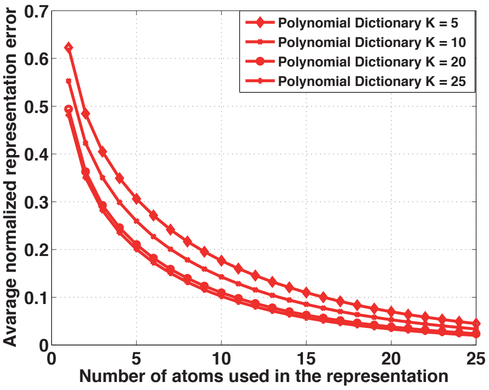
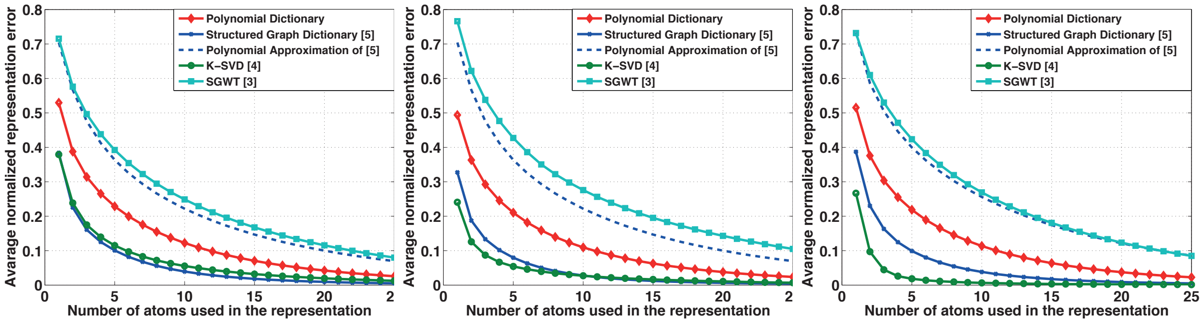


Figure 4.6: Comparison of the average approximation performance of our learned dictionary on test signals generated by the non-polynomial synthetic generating dictionary, for $K = \{5, 10, 20, 25\}$.

rapidly [9]. Apart from the polynomial degree, a second parameter that influences the support of the atoms on the graph is the sparsity level T_0 imposed in the leaning phase. A large T_0 implies that the learning algorithm has the flexibility to approximate the signals with many atoms. In the extreme case where T_0 is very large, the atoms of the dictionary tend to look like impulse functions. On the other hand, if T_0 is chosen to be small, the algorithm learns a dictionary that approximates the signals with only a few atoms. It implicitly guides the algorithm to learn atoms that are more spread on the graph, in order to cover it fully.

Next, we test the approximation performance of our learned dictionary on a set of 2000 testing signals generated in exactly the same way as the training signals, for four different degree values of the polynomial, i.e., $K = \{5, 10, 20\}$. Fig. 4.6 shows that the approximation performance obtained with our algorithm improves as we increase the polynomial degree. There are two main reasons for this improvement: (i) by increasing the polynomial degree, we allow more flexibility in the learning process; (ii) a small K implies that the atoms are localized in a small neighborhood and thus more atoms are needed to represent signals with support in different areas of the graph. However, we have empirically observed that, in practice, the improvement in the performance saturates as the value of K increases ($K = 20$ is usually enough to capture the frequency characteristics of the signals).

In Fig. 4.7, we fix $K = 20$, and compare the approximation performance of our learned dictionary to that of other dictionaries, with exactly the same setup as we used in Fig. 4.3. We again observe that K-SVD is the most sensitive to the size of the training data, and it clearly achieves the best performance when the size of the training set is large ($M = 2000$). Since the kernels used in the generating dictionary in this case do not match our polynomial model, the structured graph dictionary learning algorithm of [5] has more flexibility to learn non-smooth generating kernels and therefore generally achieves better approximation. For a fairer comparison of approximation performance, we fit an order $K = 20$ polynomial function to the discrete values $\widehat{g}_s$ learned with the algorithm of [5]. We observe that our polynomial dictionary outperforms the polynomial approximation of the dictionary learned by [5] in terms of approximation performance. An example of the atomic decomposition of a graph testing signal with respect to the K-SVD dictionary, the struc-



(a) $M = 400$ (b) $M = 600$ (c) $M = 2000$

Figure 4.7: Comparison of the learned polynomial dictionary to the SGWT [3], K-SVD [4] and the graph structured dictionary [5] in terms of approximation performance on test data generated from a non-polynomial generating dictionary, for different sizes of the training set.

tured graph dictionary of [5] and the polynomial graph dictionary is illustrated in Fig. 4.8. Note that the K-SVD atoms have a more global support in comparison to the other two graph dictionaries, while the polynomial dictionary atoms are the most localized in specific neighborhoods of the graph. Nonetheless, the approximation performance of our learned dictionary is competitive, especially for smaller training sets.

4.4.2 Approximation of real graph signals

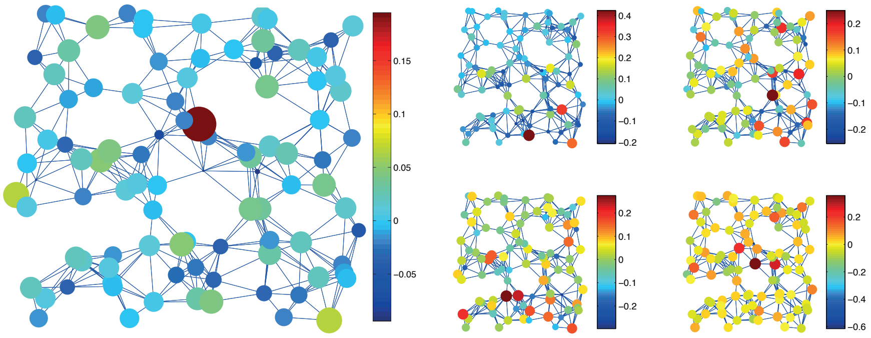
After examining the behavior of the polynomial dictionary learning algorithm for synthetic signals, we illustrate the performance of our algorithm in the approximation of localized graph signals from real world datasets. In particular, we examine the following three datasets.

Flickr Dataset: We consider the daily number of distinct Flickr users that took photos at different geographical locations around Trafalgar Square in London, between January 2010 and June 2012 [46]. Each vertex of the graph represents a geographical area of 10 × 10 meters and it corresponds to the centroid of the area. We measure the pairwise distance between the nodes and we set the cutoff distance of the graph to 30 meters. We assign an edge between two locations when the distance between them is smaller than the cutoff distance, and we set the edge weight to be inversely proportional to the distance. By following this procedure, we obtain a sparse graph. The number of vertices of the graph is $N = 245$. The signal on the graph is the total number of distinct Flickr users that have taken photos at each location during a specific day. We have a total of 913 signals, and we use 700 of them for training and the rest for testing. We set $S = 2$ and $K = 10$ in our learning algorithm.

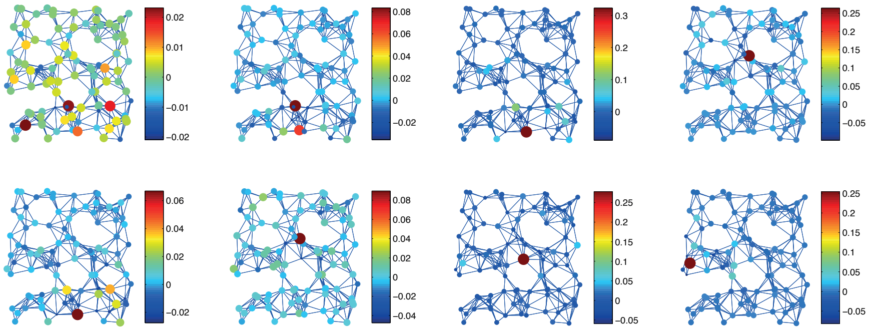
Traffic Dataset: We consider the daily bottlenecks in Alameda County in California between January 2007 and May 2013. The data are part of the Caltrans Performance Measurement System (PeMS) dataset that provides traffic information throughout all major metropolitan areas of California [134].[2] In particular, the nodes of the graph consist of $N = 439$ detector stations where

---

[2] The data are publicly available at http://pems.dot.ca.gov.



(a) Graph signal (b) K-SVD



(c) Structured graph dictionary (d) Polynomial dictionary

Figure 4.8: (a) An example of a graph signal from the testing set and its atomic decomposition with respect to (b) the K-SVD dictionary, (c) the dictionary learned by [5] and (d) the learned polynomial graph dictionary.

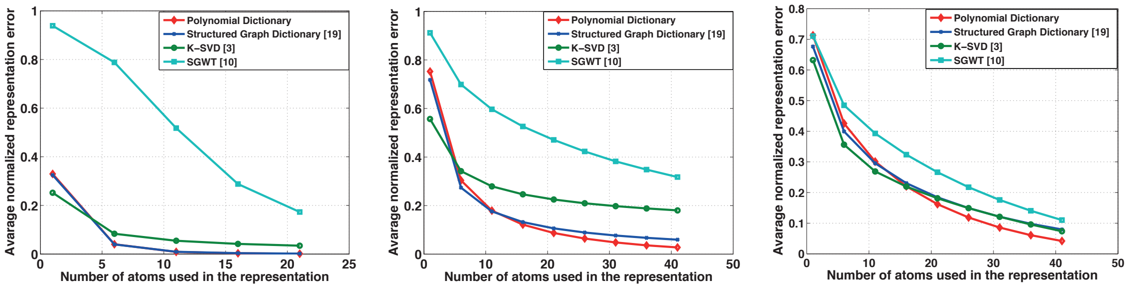
Table 4.1: Parameters of Real Graph Signals

Dataset	N	S	K	Training set	Testing set
Flickr	245	2	10	700	213
Traffic	439	2	10	1459	882
Brain	88	2	15	2580	3870

bottlenecks were identified over the period under consideration. The graph is designed by connecting stations when the distance between them is smaller than a threshold of $\theta = 0.08$, which corresponds to approximately 13 kilometers. The distance is set to be the Euclidean distance of the GPS coordinates of the stations and the edge weights are set to be inversely proportional to the distance. A bottleneck could be any location where there is a persistent drop in speed, such as merges, large on-ramps, and incidents. The signal on the graph is the average length of the time in minutes where a bottleneck is active for each specific day. In our experiments, we fix the maximum degree of the polynomial to $K = 10$ and we learn a dictionary consisting of $S = 2$ subdictionaries. We use the signals in the period between January 2007 and December 2010 for training and the rest for testing. For computational issues, we normalize all the signals with respect to the norm of the signal with maximum energy.

Brain Dataset: We consider a set of fMRI signals acquired on five different subjects [135], [136]. For each subject, the signals have been preprocessed into timecourses of $N = 88$ brain regions of contiguous voxels, which are determined from a fixed anatomical atlas, as described in [136]. The timecourses for each subject correspond to 1290 different graph signals that are measured while the subject is in two different states, either completely relaxing, in the absence of any stimulation, or passively watching small movie excerpts. For the purpose of this chapter, we treat the measurements at each time as an independent signal on the 88 vertices of the brain graph. The anatomical distances between regions of the brain are approximated by the Euclidean distance between the coordinates of the centroids of each region, the connectivity of the graph is determined by assigning an edge between two regions when the anatomical distance between them is shorter than 40 millimetres, and the edge weight is set to be inversely proportional to the distance. We then apply our polynomial dictionary learning algorithm in order to learn a dictionary of atoms representing brain activity across the network at a fixed point in time. We use the graph signals from the timecourses of two subjects as our training signals and we learn a dictionary of $S = 2$ subdictionaries and a maximum polynomial degree of $K = 15$. We use the graph signals from the remaining three timecourses to validate the performance of the learned dictionary. As in the previous dataset, we normalize all of the graph signals with respect to the norm of the signal with maximum energy. A summary of the main parameters of the three datasets is shown in Table 4.1.

Fig. 4.9 shows the approximation performance of the learned polynomial dictionaries for the three different datasets, for a sparsity constraints in the learning phase of $T_0 = 6$. The behavior is similar in all three datasets, and also similar to the results on the synthetic datasets in the previous section. In particular, the data-adapted dictionaries clearly outperform the SGWT dictionary in terms of approximation error on test signals, and the localized atoms of the learned polynomial



(a) (b) (c)

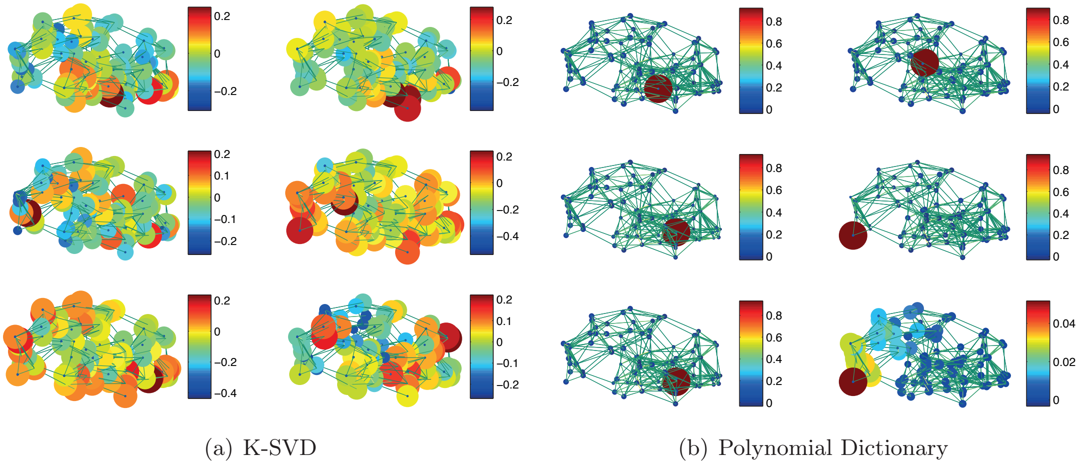
Figure 4.9: Comparison of the learned polynomial dictionaries to the SGWT, K-SVD, and graph structured dictionaries [5] in terms of approximation performance on testing data generated from the (a) Flickr, (b) traffic, and (c) brain datasets, for $T_0 = 6$.

dictionary effectively represent the real graph signals. It can even achieve better performance than K-SVD when sparsity increases. In particular, we observe that K-SVD outperforms both graph structured algorithms for a small sparsity level as it learns atoms that can smoothly approximate the whole signal. Comparing our algorithm with the one of [5], we observe that the performance of the latter is comparable. Apart from the differences between the two algorithms that we have already discussed in the previous subsections, one drawback of [5] is the way the dictionary is updated. Specifically, the update of the dictionary is performed block by block, which leads to a local optimum in the dictionary update step. This can lead to worse performance when compared to our algorithm, where all subdictionaries are updated simultaneously.

In Fig. 4.10, we illustrate the six most used atoms after applying OMP for the sparse decomposition of the testing signals from the brain dataset in the learned K-SVD dictionary and our learned polynomial dictionary. Note that in Fig. 4.10(b), the polynomial dictionary consists of localized atoms with support concentrated on small neighborhoods of vertices. These atoms capture the activation of particular regions of the brain. Interestingly, we observe that one of the most frequently chosen atoms is the one capturing the visual cortex, which is found in the back of the brain (second figure in the third row). The result of the sparse coding in this case is consistent with the pattern that we expect to appear in the brain, as the visual cortex is activated during visual stimuli. The price to pay for the interpretability and the localization of the atoms, is the poor approximation performance at low sparsity levels. However, as the sparsity tolerance increases, the localization property clearly becomes beneficial. Detecting the activated patterns in the brain using our polynomial dictionary is a very promising research direction.

4.5 Parametric dictionary learning from multiple graphs

We now extend our algorithm to graph signals that reside on the vertices of different topologies. Indeed, in many applications, one has to deal with signals that live on different graphs but share some common spectral characteristics. In such cases, a good representation dictionary should contain graph atoms that are adapted to the representations of the signals on each graph independently, and simultaneously allow to capture the spectral similarities across the signals on the different topolo-



(a) K-SVD (b) Polynomial Dictionary

Figure 4.10: Examples of atoms learned from training data from the brain dataset with (a) K-SVD and (b) the polynomial dictionary. The six atoms illustrated here are the ones that were most commonly included by OMP in the sparse decomposition of the testing signals.

gies. The signal model and in particular the polynomial dictionary structure defined in Section 4.2.2 captures a wide class of signals that are defined as processes on the graph. In particular, one can think of the polynomial dictionary $\mathcal{D}$ as a concatenation of S graph filters $\{\widehat{g}_s(\cdot)\}_{s=1}^{S}$ defined in the spectral domain of the graph, which act on some initial graph signals $\{x_s\}_{s=1}^{S}$ defined by the sparse codes, through the successive application of local graph operators i.e.,

$$\begin{aligned} y(n) &= \sum_{s=1}^{S}\sum_{\ell=0}^{N-1} \widehat{x}_s(\lambda_\ell)\widehat{g}_s(\lambda_\ell)\chi_\ell(n) \\ &= \sum_{s=1}^{S}\sum_{m=1}^{N} x_s(m)\sum_{k=0}^{K}\alpha_{sk}\sum_{\ell=0}^{N-1}\lambda_\ell^k\chi_\ell(m)\chi_\ell(n) \\ &= \sum_{s=1}^{S}\sum_{m=1}^{N} x_s(m)\sum_{k=0}^{K}\alpha_{sk}(\mathcal{L}^k)_{n,m}. \end{aligned} \tag{4.17}$$

Thus, the graph signal can be considered as the combination of the output of a set of diffusion processes, which start from different nodes and evolve according to the connectivity pattern defined by the graph topology. The polynomial coefficients become then parameters that control the rate of the diffusion. When a diffusion process evolves in different topologies, the connectivity pattern and the initial signals vary according to the topology while the rate of the diffusion i.e., the polynomial coefficients, are constant over the topologies.

In what follows, we exploit the polynomial structure of the graph dictionaries, and we capture the spectral components of the signals through the computation of kernels that are similar across all the graph topologies. Given a set of training signals $Y_t = [y_{t1}, y_{t2}, ..., y_{tM_t}] \in \mathbb{R}^{N\times M_t}$, $t = \{1, 2, .., T\}$, living on the weighted graphs $\mathcal{G}_t, t = \{1, 2, ..., T\}$, our objective is to learn a common structure for the different graph dictionaries $\mathcal{D}^t \in \mathbb{R}^{N \times NS}$ that can efficiently represent all the signals in $\mathcal{G}_t$ as linear combinations of only a few atoms. For each graph $\mathcal{G}_t$ and the corresponding training signals Y_t, we want to design a structured graph dictionary $\mathcal{D}^t = [\mathcal{D}^t_1, \mathcal{D}^t_2, ..., \mathcal{D}^t_S]$ that is a concatenation of a set of S subdictionaries of the form

$$\mathcal{D}^t_s = \widehat{g}_s(\mathcal{L}_t) = \chi_t \left(\sum_{k=0}^{K} \alpha_{sk} \Lambda_t^k \right) \chi_t^T = \sum_{k=0}^{K} \alpha_{sk} {\mathcal{L}_t}^k, \tag{4.18}$$

where $\mathcal{L}_t$ denotes the Laplacian of $\mathcal{G}_t$, and χ_t, Λ_t, are the corresponding eigenvectors and eigenvalues respectively. We thus impose a dictionary structure that consists of polynomial functions of the Laplacian and the coefficients of the polynomials, i.e., the generating kernels, capture the common information across the graphs in the spectral domain. The polynomial coefficients are learned jointly from the set of training signals on all graphs. Therefore, the dictionary learning problem can be cast as the following optimization problem:

$$\begin{aligned} \underset{\alpha \in \mathbb{R}^{(K+1)S},\ X_t \in \mathbb{R}^{SN \times M_t}}{\arg\min} \quad & \left\{ \sum_{t=1}^{T} \frac{1}{M_t} \|Y_t - \mathcal{D}^t X_t\|_F^2 + \mu \|\alpha\|_2^2 \right\} \\ \text{subject to} \quad & \|X_t^m\|_0 \leq T_{0t}, \quad \forall m \in \{1, ..., M_t\}, \\ & \mathcal{D}^t_s = \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}_t^k, \quad \forall s \in \{1, 2, ..., S\}, \quad \forall 1 \in \{1, 2, ..., T\} \\ & 0I \preceq \mathcal{D}^t_s \preceq cI, \quad \forall s \in \{1, 2, ..., S\}, \quad \forall 1 \in \{1, 2, ..., T\} \\ & (c - \epsilon_1) I \preceq \sum_{s=1}^{S} \mathcal{D}^t_s \preceq (c + \epsilon_2) I, \quad \forall 1 \in \{1, 2, ..., T\} \end{aligned} \tag{4.19}$$

where X_t^m corresponds to column m of the coefficient matrix X_t, and T_{0t} is the maximum sparsity level of the coefficients of each training signal in graph $\mathcal{G}_t$. Since $\mathcal{D}^t$ has the form (4.18), the optimization problem is equivalent to learning the parameters $\{\alpha_{sk}\}_{s=1,2,...,S;\ k=1,2,...,K}$ that characterize the set of generating kernels, $\{\widehat{g}_s(\cdot)\}_{s=1,2,...,S}$, and are the same across all the graphs. We denote these parameters in a vector form in Eq. (4.19) with $\alpha = [\alpha_1; ...; \alpha_S]$, where α_s is a column vector with $(K+1)$ entries. The constraints on the spectrum of the kernels are similar to the one in the optimization problem (4.11). One, however, can omit the last constraint that defines the frame bounds, and achieve a better fitting of the data at the cost of obtaining a less structured dictionary that is more sensitive to the set of training signals. The optimization problem (4.19) is not jointly convex, but it can be approximately solved by alternating between the sparse coding and dictionary update steps, as done in the optimization problem of (4.11). Similarly, the sparse coding step is solved by applying OMP to the training signals of each graph independently as follows

$$\begin{aligned} \underset{X_t}{\text{argmin}} \quad & ||Y_t - \mathcal{D}^t X_t||_F^2 \\ \text{subject to } & \|X_t^m\|_0 \leq T_{0t}, \end{aligned} \tag{4.20}$$

where X_t^m is the m^{th} column of the matrix X_t. In the second step, the polynomial coefficients are

Algorithm 2 Parametric Dictionary Learning on Multiple Graphs

```
Input: Signal sets $\{Y_1, Y_2, ..., Y_T\}$, initial dictionaries $\{\mathcal{D}^1_{init}, \mathcal{D}^2_{init}, ..., \mathcal{D}^T_{init}\}$, target signal sparsity $\{T_{01}, T_{02}, ..., T_{0T}\}$, polynomial degree $K$, number of subdictionaries $S$, number of iterations $iter$
Output: Sparse signal representations $\{X_1, X_2, ..., X_T\}$, polynomial coefficients $\alpha$
Initialization: $\mathcal{D}^1 = \mathcal{D}^1_{init}, \mathcal{D}^2 = \mathcal{D}^2_{init}, ..., \mathcal{D}^T = \mathcal{D}^T_{init}$
for $i = 1, 2, ..., iter$ do:
    for $t = 1, 2, ..., T$ do:
        Sparse Approximation Step:
        (a) Scale each atom in $\mathcal{D}^t$ to a unit norm
        (b) Update $X_t$ using (4.20)
        (c) Rescale $X_t$, $\mathcal{D}^t$ to recover the polynomial structure
    end for
    Dictionary Update Step:
        Compute the polynomial coefficients $\alpha$ by solving (4.21), and update the dictionary according to (4.18)
end for
```

updated by solving the following quadratic program

$$\begin{aligned}
&\underset{\alpha \in \mathbb{R}^{(K+1)S}}{\arg\min} \left\{ \sum_{t=1}^{T} \frac{1}{M_t} \|Y_t - \mathcal{D}^t X_t\|_F^2 + \mu \|\alpha\|_2^2 \right\} \\
&\text{subject to } \ \mathcal{D}^t_s = \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k_t, \quad \forall s \in \{1, 2, ..., S\}, \quad \forall 1 \in \{1, 2, ..., T\} \\
&\qquad\qquad 0 \preceq \mathcal{D}^t_s \preceq c, \quad \forall s \in \{1, 2, ..., S\}, \quad \forall 1 \in \{1, 2, ..., T\} \\
&\qquad\qquad (c - \epsilon_1) I \preceq \sum_{s=1}^{S} \mathcal{D}^t_s \preceq (c + \epsilon_2) I, \quad \forall 1 \in \{1, 2, ..., T\}
\end{aligned} \tag{4.21}$$

The main steps of the optimization algorithm are shown in Algorithm 2.

Finally, we note that above formulation provides a way to learn continuous graph kernels in the spectral domain. The accuracy of the approximation of the continuous kernels, definitely depends on the spectrum of the graph topologies. In particular, if the graph topologies from which the training signals are generated contain eigenvalues that are well-distributed samples of the continuous spectral domain, the obtained kernels are good approximations of the continuous ones. Thus, they could potentially be applied to generate dictionaries from other graph topologies that are not included in the training set. The discussion in the Appendix B gives us some intuition about the relationship between the accuracy of the approximation of the continuous kernels and the sampling of the spectrum. Of course, the exact relationship between the sampling distribution and the approximation accuracy depends on the sparse coding step and the optimality of the solution. Since the optimization problem is non-convex, such analysis however requires more involved optimization techniques, which would be interesting to explore in the future.

4.5.1 Experimental results for multi-graph dictionary learning

4.5.1.1 Learning of common graph processes

In this section, we quantify the performance of the proposed algorithm on learning dictionaries for synthetic data models that represent well-known processes. We build synthetic graph signals by choosing different graphs and computing the results of common diffusion processes on each of these graphs. We then learn dictionaries for these synthetic graph signals and show that our algorithm is able to recover the core components of the signals, which can be sparsely represented as combinations of few graph atoms.

We first construct three different types of graphs with 500 vertices, namely, a graph whose edges are determined based on Euclidean distances between vertices, and two graphs that follow the forest fire model [137] and the Barabási-Albert model [138], respectively. For the first graph, we generate the coordinates of the vertices uniformly at random in the unit square, and compute the edge weights between every pair of vertices using the Euclidean distances between them and a Gaussian radial basis function (RBF) $\exp\left(-d(i,j)^2/2\sigma^2\right)$, with the width parameter $\sigma = 0.04$. We then remove all the edges whose weights are smaller than 0.09. Next, we use the forest fire (FF) model with forward burning probability 0.1 to generate a random graph, and backward burning ratio 0.005. Finally, we use the Barabási-Albert (BA) model to generate a scale-free random graph. Then, we consider three data models that have been extensively used in the literature for applications such as classification [139], 3D shape analysis [70], and graph matching [69], for example. These models are the following:

1. *Heat diffusion kernel*: It is defined by choosing the kernel to be an exponential function of the eigenvalues of the Laplacian [139]:

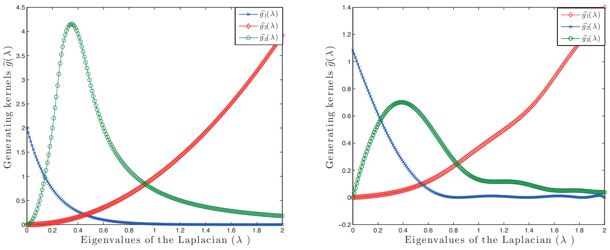
$$\widehat{g}_\tau(\lambda_\ell) = e^{-\tau\lambda_\ell}. \tag{4.22}$$

   The application of different powers τ of the heat diffusion operator to an initial signal describes the flow of the heat over the graph with the rates of the flows being proportional to the edge weights of the graph. Due to the exponential function of the kernel, this process mainly acts as a low frequency filtering, revealing information about the smooth parts of the signal.

2. *Wave kernel*: By selecting the kernel

$$\widehat{g}_\tau(\lambda_\ell) = e^{-\frac{(\tau - \log \lambda_\ell)^2}{2\sigma^2}}, \tag{4.23}$$

   the process is equivalent to a physical model that evaluates the average probability of a quantum particle with a certain energy distribution to be located at a particular vertex [67]. This model, in contrast to the heat diffusion process, has a natural notion of scale defined by τ, since it is a function of the energy levels that are directly related to the scales. Thus, it is more appropriate for capturing localized information.



(a) Original kernels (b) Learned kernels

Figure 4.11: Generating kernels $\{\widehat{g}_s(\cdot)\}_{s=1,2,\ldots,S}$ as a function of the eigenvalues.

3. *Spectral graph wavelet kernel*: By selecting

$$\widehat{g}_\tau(\lambda_\ell) = \widehat{g}(\tau\lambda_\ell), \tag{4.24}$$

with $\widehat{g}(\cdot)$ properly chosen band pass filter defined in the spectral domain, the polynomial dictionary becomes the spectral graph wavelet frame [3]. The different values of τ represent the different scales of the transform.

With the above models, we then construct graph signals as follows. For each graph, we generate a synthetic dictionary consisting of $S = 3$ subdictionaries, i.e., 1500 atoms, of the form (4.18), generated by the following kernels: (i) a heat kernel, with $\tau = 5$, (ii), a wavelet kernel for a fixed scale of $\tau = 4.1$, which is set to be a cubic spline as defined in [3, Section 8.1, Eq. (65)], with $\alpha = \beta = 2$, $x_1 = 1$, $x_2 = 2$, and (iii) a wave kernel, with $\tau = 0.01$ and $\sigma = 1/\sqrt{2}$. The training signals are generated by linearly combining $T_0 = 4$ atoms, chosen randomly from the corresponding dictionary, with randomly generated coefficients. We use these training signals to learn graph dictionaries built on polynomial functions of degree $K = 15$ of the three graph Laplacians using our dictionary learning algorithm. In all our experiments, we use the *sdpt3* solver [132] in the *yalmip* optimization toolbox [133] to update the polynomial coefficients for fixed sparse codes in the learning algorithm. For the sparse coding step in the testing phase, we use OMP, where we first normalize the dictionary atoms to a unit norm. In all the experiments, we set the values of the parameters of the optimization problem (4.19) to $c = 10$, $\mu = 10^{-4}$, $\epsilon_1 = c$, $\epsilon_2 = 2c$, and the maximum number of iterations of the alternating optimization algorithm to 50.

The original and the learned kernels are illustrated in Fig. 4.11 as a function of the eigenvalues of the Laplacian, for a training size of $M = 400$. We observe that the learned kernels capture the spectral characteristics of the original three data models used to construct the training signals. It confirms that the polynomial atoms are able to correctly approximate the processes represented by the graph signals, despite the fact that these signals live on different graphs.

Then, we test the approximation performance of the learned dictionary on a set of 2000 testing signals from each graph, generated in the same way as the training signals. We further consider two different sizes for the training data: the training set in the first one consists of $M = 400$ signals per graph, while in the second one, it consists of $M = 1000$ signals per graph. We compare the approximation performance of our algorithm to that obtained with (i) the spectral graph wavelet

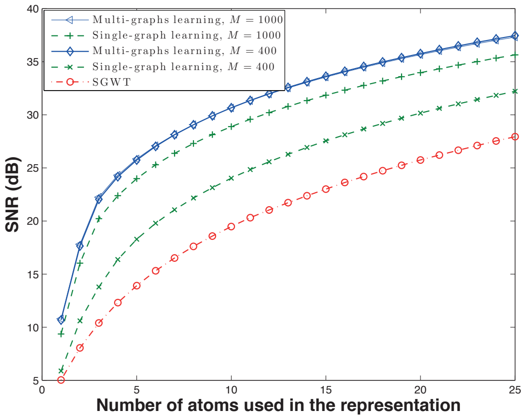
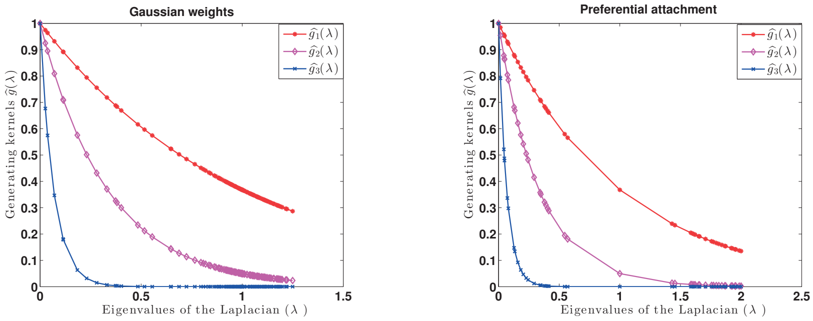


Figure 4.12: Comparison of multi-graphs learning, with SGWT and single-graph learning in terms of SNR in the synthetic data.

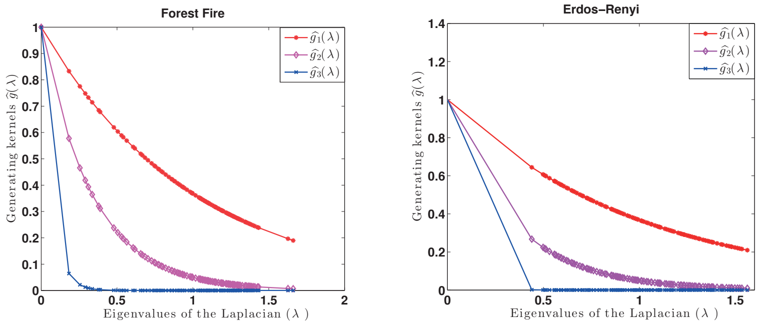
transform (SGWT) [3] and (ii) a polynomial graph dictionary, learned in each graph separately. In Fig. 4.12, we plot the signal-to-approximation noise ratio (SNR) in dB, for different sparsity levels. The SNR for each sparsity level is the average over the three graph topologies. We observe that learning on multiple graphs can significantly improve the SNR with respect to both SGWT, and dictionaries learnt on different graphs independently. In particular, the results indicate that the dictionaries resulting from joint learning are more stable with respect to the size of the training set. Learning on multiple graphs can thus be more efficient than learning dictionaries independently in each graph, especially when the training set is quite small ($M = 400$). In this case, the information obtained from the different graphs compensates for the lack of training signals in each graph separately and can be combined in order to learn the true dictionary.

4.5.1.2 Learning on different graph models

In the next set of experiments, we illustrate the effect of the distribution of the eigenvalues in the learning phase and in particular in the recovery of the continous spectral kernels. We generate random graphs using the following four different methods, i.e., (i) Gaussian kernel, (ii) preferential attachment, (iii) Erdős Renyi, and (iv) forest fire model. The distribution of the spectrum is different in each case. We generate $T = 4$ graphs, one from each model, of $N = 100$ vertices each. For each graph, we use a generating dictionary $\mathcal{D}^t, t = \{1, 2, 3, 4\}$ that is a concatenation of $S = 3$ subdictionaries, each representing a heat diffusion kernel of the form of (4.22) i.e., $\mathcal{D}^t = [e^{-\tau_1 \mathcal{L}_t}, e^{-\tau_2 \mathcal{L}_t}, e^{-\tau_3 \mathcal{L}_t}]$. We choose $\tau = \{1, 3, 15\}$ and we plot in Fig. 4.13 the samples of the generating kernels in each graph. We generate 400 graph signals per topology by linearly combining $T_0 \leq 4$ random atoms from the dictionary with random coefficients. The random coefficients represent the initial values of the diffusion process. During the learning phase, we train the dictionaries with signals from two topologies and we apply the polynomial coefficients to approximate the kernels from the spectrum of the remaining two topologies. We then choose $c = 4$, $\epsilon_1 = c$, $\epsilon_2 = 2c$, and $\mu = 10^{-6}$ and we learn jointly the parameters of the dictionary by solving the optimization



(a) Random graph with Gaussian weights (b) Preferential attachment model



(c) Forest Fire model (d) Erdős Renyi

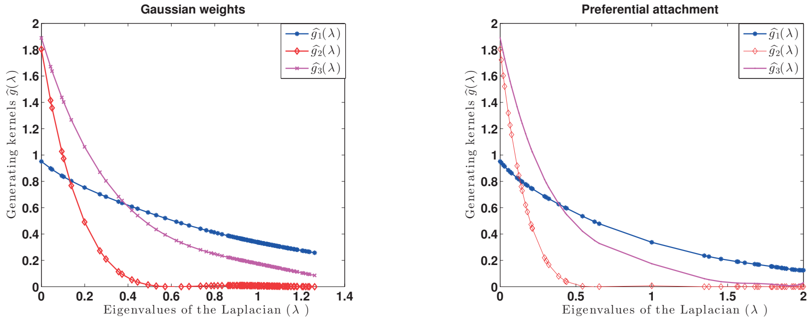
Figure 4.13: Generating kernels of the four graph models

problem (4.19).

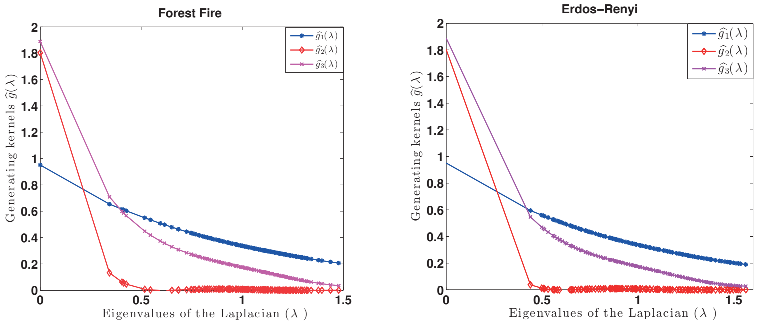
In particular, we consider the following two scenarios:

- First, we learn a dictionary from training signals living on a random graph with Gaussian weights and a randomly generated preferential attachment graph. We then apply the learned polynomial coefficients on an Erdős Renyi and a forest fire graph. The corresponding kernels are shown is Fig. 4.14. We observe that the sampling of the spectrum in the topologies used for training is good enough to provide a good approximation of the continuous kernels in all the four topologies.

- Second, we learn a dictionary from training signals living on a random graph with Gaussian weights and a Erdős Renyi graph. We then apply the learned polynomial coefficients on an preferential attachment and a forest fire graph. In Fig. 4.15, we observe that the learned kernels approximate the continuous ones only for the discrete values of the spectrum that are contained in the training graphs. We notice that if we apply the polynomial coefficients learned in the training phase to the preferential attachment graph, we are not able to approximate the continuous function. This is due to the fact that this graph has eigenvalues in higher frequencies (λ_{max} close to 2) that do not appear in the graphs that we used for learning.



(a) Random graph with Gaussian weights (b) Preferential attachment model



(c) Forest Fire model (d) Erdős Renyi

Figure 4.14: Learned kernels of the four graph models. Learning is based on (a) and (b).

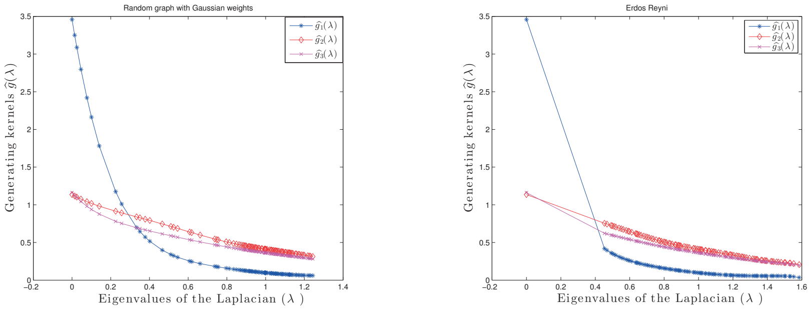
These results confirm the intuitive conclusion that the approximation of the continuous kernel depends on the sampling of the spectrum of the topologies included in the training phase.

4.5.1.3 Sparse representation of traffic data

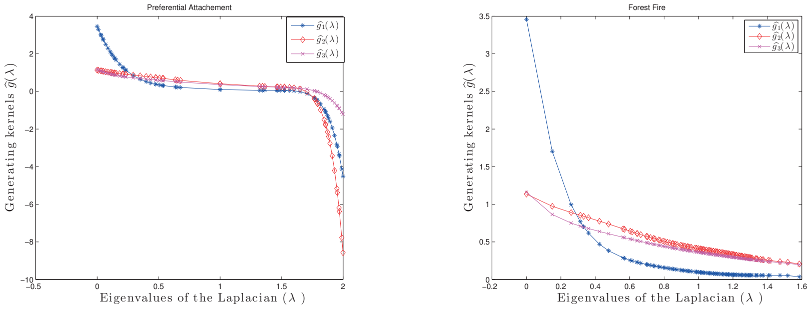
Finally, we illustrate the performance of our algorithm in the representation of localized graph signals related to the traffic information in different counties in the state of California. In particular, we consider the daily bottlenecks in San Francisco, Alameda, and Santa Barbara counties between January 2007 and August 2014. A bottleneck could be any location where there is a persistent drop in speed, such as merges, large on-ramps, and incidents. The data are part of the Caltrans Performance Measurement System (PeMS) dataset that provides traffic information throughout all major metropolitan areas of California [134].[3] For each of the counties we design a graph whose nodes consist of $N = 75$, $N = 559$, and $N = 62$ detector stations respectively where bottlenecks were identified over the period under consideration. We have thus in total three different graphs, and each of them is designed by connecting stations separated by a distance smaller than a threshold of $\theta = 0.04$. For two stations A, B, the distance d_{AB} is set to be the Euclidean distance of the

---

[3] The data are publicly available at http://pems.dot.ca.gov.



(a) Random graph with Gaussian weights (b) Erdős Renyi



(c) Forest Fire model (d) Preferential attachment model

Figure 4.15: Learned kernels of the four graph models. Learning is based on (a) and (b).

GPS coordinates of the stations and the edge weights are computed using the exponential kernels such that $W_{AB} = e^{-d_{AB}}$. The signal on the graph is the duration in minutes of bottlenecks for each specific day. We remove the signal instances where no daily bottleneck is identified; for computational reasons, we further normalize each signal to a unit norm. The final number of signals is 2766, 2772, and 894 for San Francisco ($\mathcal{G}_1$), Alameda ($\mathcal{G}_2$), and Santa Barbara ($\mathcal{G}_3$) respectively. For each graph, we use half of the signals for jointly learning a polynomial graph dictionary and the rest for testing the performance of the learned dictionary. In our experiments, we fix the maximum degree of the polynomial to $K = 15$ and we learn a dictionary with $S = 3$.

In Fig. 4.16, we illustrate the sparse approximation performance of our dictionary representation by studying the reconstruction performance in SNR on the set of testing signals for different sparsity levels. The performance is compared to that obtained by learning separately a dictionary on each graph [123], and the one obtained by the sparse decomposition in the graph wavelet dictionary [3]. We observe that multi-graph learning improves significantly the performance in comparison to SGWT. Moreover, it outperforms the single-graph learning algorithm in all the graphs. In the latter case however, the gain is more evident when representing signals from the Santa Barbara graph $\mathcal{G}_3$. In this particular graph, jointly learning a dictionary compensates for the relatively small number of available training signals for this graph in comparison to the other two graphs.

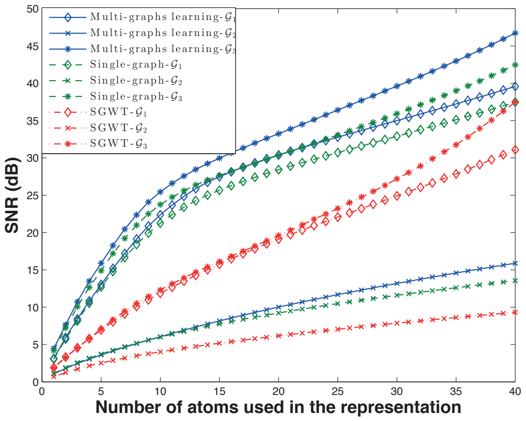


Figure 4.16: Comparison of multi-graphs learning, with SGWT and single-graph learning in terms of SNR in the traffic delay dataset for three different counties.

4.6 Conclusion

We have introduced a parametric family of structured dictionaries – namely, unions of polynomial matrix functions of the graph Laplacian – to sparsely represent signals on a given weighted graph, and an algorithm to learn the parameters of a dictionary belonging to this family from a set of training signals on the graph. When translated to a specific vertex, the learned polynomial kernels in the graph spectral domain correspond to localized patterns on the graph. Translating each of these patterns to different areas of the graph leads to sparse approximation performance that is clearly better than that of non-adapted graph wavelet dictionaries such as the SGWT, and comparable to or better than that of dictionaries learned by state-of-the art numerical algorithms such as K-SVD. The performance of our learned dictionaries is also more stable with respect to the size of training data. Furthermore, the polynomial structure permits to extend the proposed framework to signals that live on different graph topologies but share some common spectral characteristics. These characteristics are reflected in a common dictionary structure that is constructed to be a polynomial function of the graph Laplacian matrix, and the coefficients of the polynomials are jointly learned from all graph signals on all graph instances. Experimental results showed the effectiveness of our joint dictionary learning method in sparse representation of graph signals and significant benefit with respect to independent learning in the case when the size of the training set is small.

Although we have provided some preliminary results in signal approximation, the potential of the proposed dictionary structure in both the single and multi-graph case is yet to be explored in other applications (see Appendix C for an illustrative example). We believe that the simple, efficient, and at the same time informative polynomial structure can be applied in other graph signal processing and data analysis tasks such as classification, clustering, community detection, or source localization where we expect that the localization properties of our polynomial dictionary can be beneficial. In particular, in the multi-graph learning case, as our algorithm permits to learn an effective graph signal representation from training signals living on different graphs, it could surely provide important benefits in data mining tasks, where the goal is the detection of commonalities across different datasets. In practice, it is common to have graph signals that live on different topologies or on graphs that do no have exactly the same number of nodes. Then, it might be necessary to segment large graphs into smaller graphs for complexity reasons, or for augmenting the number of training data. Another extension of our multi-graph framework could be in settings where the number of nodes is the same across different graphs but each connectivity captures a different source of information i.e., in multilayer graphs. In that case, the graph structure of each layer could allow us to learn atoms with different orientations similar to edgelets in classical settings. In all these cases, our algorithm is expected to be helpful as it goes beyond the limitations of traditional learning solutions that require training signals to live on a fixed topology. Finally, because our learned dictionaries are unions of polynomial matrix functions of the graph Laplacian, they can be efficiently stored and implemented in distributed signal processing tasks. This is exactly the focus of our next chapter.

Chapter 5

Distributed Graph Signal Processing with Quantization

5.1 Introduction

In the previous chapter, we proposed a method for constructing polynomial graph dictionaries that sparsely represent graph signals. In this chapter, we study the application of these dictionaries in distributed settings in the framework of wireless sensor networks. Such networks are widely deployed, with applications such as surveillance, weather monitoring, or automatic control that are often supported by distributed signal processing methods. In such settings, the sensors are generally represented by the vertices of a graph, whose edge weights capture the pairwise relationships between the vertices. A graph signal is defined as a function that assigns a real value to each vertex, which corresponds to the quantity measured by the sensor, such as the current temperature or the road traffic level at a particular time instance.

Graph representations are certainly powerful and promising tools for representing signals in the irregular structured domain defined by sensor networks [9]. As studied in the previous chapter, graph signals can be modeled as the linear combinations of a small number of constitutive components in a polynomial graph dictionary [123]. Such dictionaries, which can also be seen as a set of spectral filters on graphs, incorporate the intrinsic geometric structure of the irregular graph domain into the atoms and are able to capture different processes evolving on the graph. Due to their polynomial structure, they can be implemented distributively and therefore represent ideal operators for graph signal processing in sensor networks.

In wireless sensor networks, each sensor node typically communicates only with a small number of neighbor nodes due to energy constraints and limited communication range. In addition, the information exchanged by the network nodes is quantized prior to transmission because of limitations in communication bandwidth and computational power. The quantization process induces some noise that impacts the performance of sensor network applications and requires careful consideration to ensure the proper convergence of the distributed signal processing algorithms.

In this chapter, we study the effect of quantization in distributed graph signal representations[1].

---

[1] Part of this chapter has been published in: D. Thanou and P. Frossard. Distributed signal processing with graph spectral dictionaries, in *Proc. of Annual Allerton Conf. on Comm., Control, and Computing*, IL, USA, Oct. 2015.

In particular, we first derive the quantization error that appears in the distributed computation of different operators defined on graph spectral dictionaries with polynomial structures. Our analysis is quite generic and can be applied to every graph spectral dictionary that can be approximated by polynomials of the graph Laplacian (e.g., spectral graph wavelets approximated with Chebyshev polynomials) [55], [56]. We then consider the problem of sparse representation of graph signals that is implemented in a distributed way with an iterative soft thresholding algorithm. We analyze the convergence of the algorithm and show how it depends on the quantization noise, whose influence is itself governed by the characteristics of the dictionary. We then propose an algorithm for learning polynomial graph dictionaries that permits to control the robustness of distributed algorithms to quantization noise. Experimental results illustrate the dictionary design tradeoffs between accurate signal representation and robustness to quantization errors. They show in particular that it is necessary to sacrifice on signal approximation performance for ensuring proper convergence of distributed algorithms in low bit rate settings.

The chapter is organized as follows. In Section 5.2, we model the sensor network with a graph, and we recall the use of polynomial graph dictionaries for distributed processing of graph signals. We study the quantization error that appears in the distributed computations with polynomial graph dictionaries in Section 5.3, and in Section 5.4 we analyze the more specific case of the sparse approximation of graph signals. In Section 5.5, we propose an algorithm for learning polynomial graph dictionaries that are robust to the quantization noise. Finally, in Section 5.6 we evaluate the performance of our algorithm in both synthetic and real world signals.

5.2 Polynomial graph dictionaries for distributed signal representation

For the sake of completeness, we recall some of the basic concepts and we introduce notations that are needed for the rest of this chapter. First, we model the sensor network topology as a weighted graph, whose connectivity indicates the communication pattern. Moreover, we recall briefly the sparse signal model of the previous chapter that is based on polynomial dictionaries of the graph Laplacian. Finally, we show the benefits of such dictionaries in distributed processing. As an illustrative example, we focus on the basis pursuit denoising algorithm.

5.2.1 Distributed sensor network topology

We consider a sensor network topology that is modeled as a weighted, undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$, where $\mathcal{V} \in \{1, \ldots, N\}$ represents the set of sensor nodes and $N = |\mathcal{V}|$ denotes the number of nodes. An edge denoted by an unordered pair $\{i, j\} \in \mathcal{E}$, represents a communication link between two sensor nodes i and j. Moreover, a positive weight $W_{ij} > 0$ is assigned to each edge if $\{i, j\} \in \mathcal{E}$, whose value is dependent on the distance between the nodes i and j. D is a diagonal degree matrix that contains as elements the sum of each row of the matrix W. The set of neighbors for node i is finally denoted as $\mathcal{N}_i = \{j | \{i, j\} \in \mathcal{E}\}$. The normalized graph Laplacian operator is defined as $\mathcal{L} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$. We denote its eigenvectors by $\chi = [\chi_1, \chi_2, ..., \chi_N]$, and the spectrum of the eigenvalues by $\Lambda := \Big\{0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_{(N-1)} \leq 2\Big\}$.

We borrow the signal model from the previous chapter and we model the sensor signals as sparse linear combinations of (overlapping) graph patterns $\widehat{g}(\cdot)$, defined in the spectral domain of the graph, and positioned at different vertices. We recall that each pattern captures the local form of the signal in the neighborhood of a vertex, and it can be considered as a function whose values depend on the local connectivity around that vertex. A graph dictionary is then defined as a concatenation of subdictionaries in the form $\mathcal{D} = [\widehat{g_1}(\mathcal{L}), \widehat{g_2}(\mathcal{L}), ..., \widehat{g_S}(\mathcal{L})]$, where each subdictionary is defined as

$$\widehat{g}(\mathcal{L}) = \chi \left(\sum_{k=0}^{K} \alpha_k \Lambda^k \right) \chi^T = \sum_{k=0}^{K} \alpha_k \mathcal{L}^k. \tag{5.1}$$

Finally, a graph signal y can be expressed as a linear combination of a set of atoms generated from different graph kernels $\{\widehat{g_s}(\cdot)\}_{s=1,2,...,S}$,

$$y = \sum_{s=1}^{S} \widehat{g_s}(\mathcal{L}) x_s = \sum_{s=1}^{S} \mathcal{D}_s x_s,$$

where we have set $\mathcal{D}_s = \widehat{g_s}(\mathcal{L})$, and x_s are the coefficients in the linear combination. In applications where sparsity is required, one can typically learn the polynomial coefficients numerically from a set of training signals that live on the graph as shown in Chapter 4. Another example of the dictionary $\mathcal{D}$ is the spectral graph wavelet dictionary [3], or more generally the union of graph Fourier multipliers that can be efficiently approximated with Chebyshev polynomials [56].

5.2.2 Distributed computation of the graph operators

An important benefit of polynomial graph dictionaries is the fact that they can be efficiently stored and implemented in distributed signal processing tasks. Each polynomial dictionary can be constructed locally, i.e., by exchanging only information between nodes that are connected by an edge on the graph. For the sake of completeness, we recall here the distributed computation of some of these operators. More details can be found in [55].

The distributed computation of $\mathcal{D}^T y$ requires first the computation of the different powers of the Laplacian matrix, i.e., $\{\mathcal{L}^0 y, \mathcal{L}^1 y, \mathcal{L}^2 y, ..., \mathcal{L}^K y\}$, in a distributed way. The latter can be done efficiently by successive multiplications of the matrix $\mathcal{L}$ with the signal y over K iterations. We introduce a new variable $z_k \in \mathbb{R}^N$ in the computation of $\mathcal{D}^T y$, which represents the value transmitted during the k^{th} iteration, with $z_0 = y$. Initially, each node transmits its component of z_0 only to its one-hop neighbors on the graph. After receiving the values from its neighbors, it updates its component as a linear combination of its own value in z_0 and the values received from its neighbors as follows

$$z_1 = \mathcal{L} z_0. \tag{5.2}$$

At the next iteration, the values of z_1 are exchanged locally in the network. The procedure is repeated over K iterations, and the exchanged messages are computed based on the previous recursive update relationship. After knowing $\{z_0(n), z_1(n), ..., z_K(n)\}$, each node n can compute the n^{th} component of $\mathcal{D}_s y$ by a simple linear combination with the polynomial coefficients i.e., $(\mathcal{D}_s y)(n) = \sum_{k=1}^{K} \alpha_{sk} z_k(n)$. The same can be done for the different subdictionaries and their

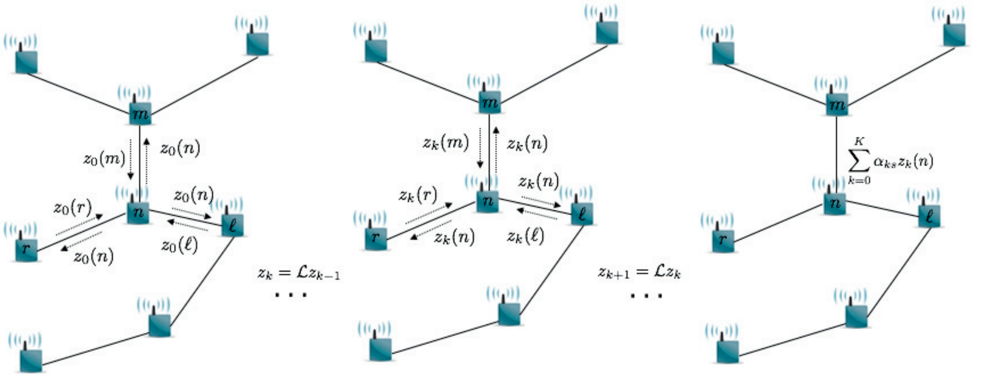


Figure 5.1: Distributed computation of $\mathcal{D}_s y$ in a sensor network. Sensor n exchanges messages with its one-hop neighbors for K iterations. After each iteration, the received messages are filtered with weights defined by the graph Laplacian i.e., $z_k(n) = (\mathcal{L}z_{k-1})(n)$, and are transmitted locally in the network. $(\mathcal{D}_s y)(n)$ is computed as a linear combination of the messages exchanged during the K iterations.

polynomial coefficients. An illustration of the process is shown in Fig. 5.1. The main steps are given in Algorithm 3.

Following the same reasoning, the forward operator $\mathcal{D}x$ can be computed in a distributed way. We recall that $\mathcal{D}x = \sum_{s=1}^{S} \mathcal{D}_s x_s$, where $x = (x_1,\ x_2,\ \ldots\ , x_S)$ is a vector containing as entries the sparse codes x_s corresponding to subdictionary $\mathcal{D}_s$. Each of the components in the summation is computed by sending iteratively the powers of the Laplacian as follows. For each of the subdictionaries, we define a new variable $z_{s,0} = x_s$. The transmitted value of this variable by sensor n at iteration k is

$$z_{s,k}(n) = (\mathcal{L}z_{s,k-1})(n).$$

Each sensor can then compute its component in $\mathcal{D}x$, which can be expressed as follows in a vector form

$$\mathcal{D}x = \sum_{s=1}^{S} \mathcal{D}_s x_s = \sum_{s=1}^{S} \sum_{k=0}^{K} \alpha_{sk} z_{s,k}.$$

The main steps of the distributed algorithm for the computation of $\mathcal{D}x$ are shown in Algorithm 4. Finally, the operator $\mathcal{D}^T\mathcal{D}x$ can be implemented in distributed settings by first computing $\mathcal{D}x$ and sequentially $\mathcal{D}^T\mathcal{D}x$ by following Algorithms 4 and 3 respectively.

Such operators are particularly useful in the distributed implementation of signal processing tasks related to learning or regularizing on the graph, such as denoising [55], semi-supervised learning [40], signal reconstruction [123], interpolation and reconstruction of band limited graph signals [39]. These types of applications typically require the computation of quantities such as the forward application of the dictionary and its adjoint to be computed only by local exchange of information. In the next section, we give an illustrative example of the use of such operators in the distributed sparse representation of graph signals.

Algorithm 3 Distributed computation of $\mathcal{D}^T y$

```
Input at node $n$: $y(n), \mathcal{L}_{n,:}$, $\alpha = [\alpha_1; ...; \alpha_S]$
Output at node $n$: $(\mathcal{D}^T y)\big((s-1)N + n\big)$ for all $s = \{1, ..., S\}$
Transmit $z_0(n) = y(n)$ to all neighbors in $\mathcal{N}_n$
Receive $z_0(m) = y(m)$ from all neighbors in $\mathcal{N}_n$
for $k = 2, ..., K$ do:
    Transmit $z_{k-1}(n) = (\mathcal{L}^T z_{k-2})(n)$ to all the neighbors
    Receive $z_{k-1}(m) = (\mathcal{L}^T z_{k-2})(m)$ from all the neighbors $m \in \mathcal{N}_n$.
end for
Compute $z_K(n) = (\mathcal{L} z_{K-1})(n)$
for $s = \{1, ..., S\}$ do
    Compute $(\mathcal{D}^T y)\big((s-1)N + n\big) = \sum_{k=0}^{K} \alpha_{ks} z_k(n)$
end for
```

5.2.3 Illustrative application of distributed graph signal processing

We consider the distributed processing scenario where each node n of the graph computes the sparse decomposition in a polynomial dictionary by solving a sparse regularization problem of the form

$$x^* = \underset{x}{\operatorname{argmin}} ||y - \mathcal{D}x||_2^2 + \kappa \|x\|_1, \tag{5.3}$$

where κ is a parameter that controls the sparsity level. The above problem is also called LASSO [131] or basis pursuit denoising [140] and is used to perform denoising with sparse prior. We thus start with the underlying assumption that the signal y is sparse in a polynomial graph dictionary, whose coefficients are known to all the sensors. Moreover, node n knows its own component of a signal $y \in \mathbb{R}^N$ (i.e., $y(n)$) and the n^{th} row of the corresponding Laplacian matrix $\mathcal{L}_{n,:}$. The above problem can be solved by an iterative soft thresholding algorithm (ISTA) [141], in which the update of the estimated coefficients is given by

$$x^{(t)} = \mathcal{S}_{\kappa\tau}\big(x^{(t-1)} + 2\tau\mathcal{D}^T\big(y - \mathcal{D}x^{(t-1)}\big)\big), \quad t = 1, 2, ... \tag{5.4}$$

where τ is the gradient stepsize and $\mathcal{S}_{\kappa\tau}$ is the soft thresholding operator

$$\mathcal{S}_{\kappa\tau}(z) = \begin{cases} 0, & \text{if } |z| \leq \kappa\tau \\ z - \operatorname{sgn}(z)\kappa\tau, & \text{otherwise,} \end{cases}$$

which corresponds to the proximal operator of the $\kappa\|x\|_1$ function. Thus, the whole algorithm is a particular instance of the general family of proximal gradient methods [142]. By combining the distributed computation of the operations $\mathcal{D}^T y$, $\mathcal{D}^T \mathcal{D}x$, $\mathcal{D}x$, as described in the previous subsection, each iteration of ISTA can be solved in a distributed way [55]. In particular, in the first iteration, each node n must compute $(\mathcal{D}_s y)(n)$ for all the subdictionaries, via Algorithm 3. In each iteration $(t+1)$, it must compute first $(\mathcal{D}x^{(t)})(n)$, and sequentially apply $\mathcal{D}^T\mathcal{D}x^{(t)}$ via Algorithms 4 and 3, respectively. The solution of (5.3) is found after a stopping criteria is satisfied (e.g., a fixed number of iterations is executed). The estimate of the signal at each node is then given by computing

Algorithm 4 Distributed computation of $\mathcal{D}x$

```
1: Input at node $n$: $x\big((s-1)N+n\big)$, for all $s=\{1,...,S\}, \mathcal{L}_{n,:}$, $\alpha=[\alpha_1;...;\alpha_S]$, quantization stepsize $\Delta$
2: Output at node $n$: $(\mathcal{D}x)(n)$
3: Set $z_{s,0}(n)=x\big((s-1)N+m\big)$ for all $s=\{1,..,S\}$.
4: for $k=2,...,K$ do:
5:     Transmit $z_{s,k-1}(n)=(\mathcal{L}z_{s,k-2})(n)$ to all the neighbors, for all $s=\{1,2,...,S\}$.
6:     Receive $z_{s,k-1}(m)$ from all $m\in\mathcal{N}_n$, for all $s=\{1,2,...,S\}$.
7: end for
8: Compute $z_{s,K}(n)=(\mathcal{L}z_{s,K-1})(n)$, for all $s=\{1,...,S\}$.
9: Compute and output $(\mathcal{D}x)(n)=\sum_{s=1}^{S}\sum_{k=0}^{K}\alpha_{ks}z_{s,k}(n)$.
```

$\hat{y}=\mathcal{D}x^*$ via Algorithm 4.

5.3 Distributed processing with quantization

We now study the effect of quantization in distributed signal processing with polynomial dictionaries by modeling the propagation of the quantization error in the different dictionary-based operators. Given a graph signal y, and the representation of the signal in a polynomial graph dictionary $\mathcal{D}$, i.e., $y=\mathcal{D}x$, we study the computation of three basic operators i.e., the forward operator $\mathcal{D}x$, the adjoint operator $\mathcal{D}^Ty$, and the operator $\mathcal{D}^T\mathcal{D}x$ in distributed settings when sensors exchange quantized messages. Although our main focus is on graph dictionaries that are given directly in a polynomial form, we note that the following results hold for every graph spectral dictionary that can be approximated by a polynomial of the graph Laplacian matrix.

5.3.1 Distributed computation of $\mathcal{D}^Ty$ with quantization

As we already saw in Section 5.2.2, the distributed computation of $\mathcal{D}^Ty$ requires first the computation of the different powers of the Laplacian matrix, i.e., $\{\mathcal{L}^0y,\mathcal{L}^1y,\mathcal{L}^2y,...,\mathcal{L}^Ky\}$, in a distributed way. The latter can be done efficiently by successive multiplications of the matrix $\mathcal{L}$ with the signal y over K iterations, which as we will see next contains some noise that is accumulated over the iterations. We introduce a new variable z_k in the computation of $\mathcal{D}^Ty$, which captures the sensors' values at the k^{th} iteration, with $z_0=y$. Before the sensors exchange information, the value of this variable at sensor n and iteration k, i.e., $z_k(n)$, is quantized such that

$$\tilde{z}_k(n)=z_k(n)+\epsilon_k(n), \tag{5.5}$$

where $\epsilon_k(n)$ is the quantization error in k, $z_k(n)$ is the value of the sensor before quantization and $\tilde{z}_k(n)$ is the quantized value that the sensor n sends to its neighbors. Then, each node updates the local value of the variable z_k as a linear combination of its own quantized value and the quantized values received from its neighbors $\tilde{z}_k(i)$ with $i\in\mathcal{N}_n$, based on the recursive update relationship in

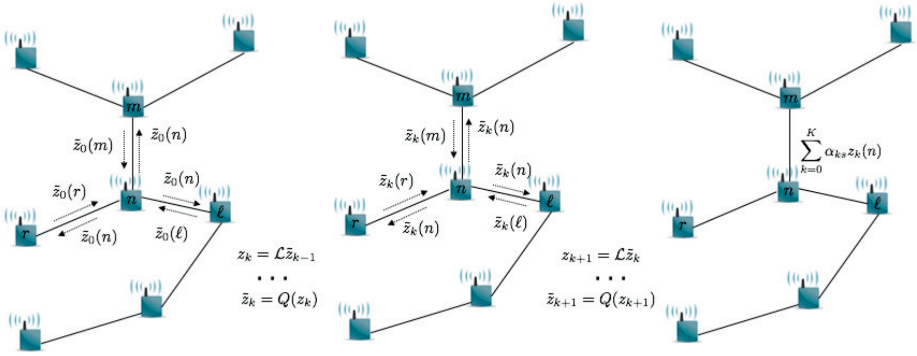


Figure 5.2: Distributed computation of $\mathcal{D}_s y$ with quantization in a sensor network. Sensor n exchanges quantized messages with its one-hop neighbors for K iterations. After each iteration $k-1$, the received messages $\tilde{z}_{k-1}$ are filtered with weights defined by the graph Laplacian (i.e., $z_k(n) = (\mathcal{L}\tilde{z}_{k-1})(n)$), quantized (i.e., $\tilde{z}_k(n) = Q(z_k(n))$), and finally transmitted locally in the network. $(\mathcal{D}_s y)(n)$ is computed as a linear combination of the messages exchanged during the K iterations.

a vectorized form

$$z_{k+1} = \mathcal{L}(z_k + \epsilon_k). \tag{5.6}$$

By taking into consideration the quantization error from the previous iterations, Eq. (5.6) can be re-written as

$$z_{k+1} = \mathcal{L}^{k+1} z_0 + \sum_{l=0}^{k} \mathcal{L}^{k+1-l} \epsilon_l. \tag{5.7}$$

We observe that the quantization process involved in the transmission of the different powers of the Laplacian, induces some quantization noise that is accumulated over the K iterations and is represented by the second term of Eq. (5.7).

We now compute $\widetilde{\mathcal{D}_s^T y}$, the quantized vector corresponding to $\mathcal{D}_s^T y$, by applying the polynomial coefficients on the values generated by the sequence $\{z_0, z_1, ..., z_K\}$ given by Eq. (5.7). It reads

$$\begin{aligned}\widetilde{\mathcal{D}_s^T y} &= \sum_{k=0}^{K} \alpha_{sk} z_k = \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k y + \sum_{l=1}^{K} \Big[\sum_{j=1}^{l-1} \alpha_{sl} \mathcal{L}^{l-j} \epsilon_j \Big] \\ &= \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k y + \sum_{l=0}^{K-1} \Big[\sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \Big] \epsilon_l \\ &= \mathcal{D}_s^T y + E(\mathcal{D}_s^T y),\end{aligned} \tag{5.8}$$

where Eq. (5.8) is obtained after some simple matrix manipulations. Note that

$$E(\mathcal{D}_s^T y) = \sum_{l=0}^{K-1} \Big[\sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \Big] \epsilon_l,$$

Algorithm 5 Distributed computation of $\mathcal{D}^T y$ with quantization

```
Input at node $n$: $y(n), \mathcal{L}_{n,:}$, $\alpha = [\alpha_1; ...; \alpha_S]$, quantization stepsize $\Delta$
Output at node $n$: $(\widetilde{\mathcal{D}^T y})\big((s-1)N + n\big)$ for all $s = \{1, ..., S\}$
Quantize and transmit $\tilde{y}(n) = y(n) + \epsilon_0(n)$ to all $m \in \mathcal{N}_n$.
Receive $\tilde{y}(m)$ from neighbors in $\mathcal{N}_n$.
Set $z_0(n) = y(n), \quad \tilde{z}_0(n) = \tilde{y}(n)$.
for $k = 2, ..., K$ do:
    Compute $z_{k-1}(n) = (\mathcal{L}^T \tilde{z}_{k-2})(n)$.
    Quantize and transmit $\tilde{z}_{k-1}(n) = z_{k-1}(n) + \epsilon_{k-1}(n)$ to all the neighbors.
    Receive $\tilde{z}_{k-1}(m)$ from all the neighbors $m \in \mathcal{N}_n$.
end for
Compute $z_K(n) = (\mathcal{L}\tilde{z}_{k-1})(n)$.
for $s = \{1, ..., S\}$ do
    Compute $(\widetilde{\mathcal{D}^T y})\big((s-1)N + n\big) = \sum_{k=0}^{K} \alpha_{ks} z_k(n)$.
end for
```

is the overall accumulated quantization noise that occurs in the distributed computation of $\mathcal{D}_s^T y$. Finally, the operation $\widetilde{\mathcal{D}^T y} = \{\widetilde{\mathcal{D}_s^T y}\}_{s=1}^S$ can be written as

$$\widetilde{\mathcal{D}^T y} = \mathcal{D}^T y + E(\mathcal{D}^T y),$$

where $E(\mathcal{D}^T y) = \{E(\mathcal{D}_s^T y)\}_{s=1}^S$ is an error vector in $\mathbb{R}^{SN}$ that contains as entries the error obtained by applying the S different sets of polynomial coefficients to the accumulated quantization noise. An illustration of the overall procedure is given in Fig. 5.2. The distributed algorithm for computing $\widetilde{\mathcal{D}^T y}$ is summarized in Algorithm 5.

5.3.2 Distributed computation of $\mathcal{D}x$ with quantization

We recall that $\mathcal{D}x = \sum_{s=1}^S \mathcal{D}_s x_s$, where $x = (x_1, \ x_2, \ ... \ , x_S)$ is a vector containing as entries the sparse codes x_s corresponding to subdictionary $\mathcal{D}_s$. Each of the components in the summation is computed by sending iteratively the powers of the Laplacian as described in Section 5.2.2. The quantization effect though in the iterative process is significant. To elaborate on that, for each of the subdictionaries, we define a new variable $z_{s,0} = x_s$. The transmitted value of this variable at sensor n and iteration k is

$$\tilde{z}_{s,k}(n) = z_{s,k}(n) + \zeta_{s,k}(n), \tag{5.9}$$

where $\zeta_{s,k}(n)$ is the quantization error, $z_{s,k}(n)$ is the value of the sensor before quantization that is computed as

$$z_{s,k}(n) = (\mathcal{L}\tilde{z}_{s,k-1})(n),$$

and $\tilde{z}_{s,k}(n)$ is the quantized value that sensor n sends to its neighbors. By taking into consideration the quantization error components from the previous iterations, the values of the sensors at iteration

Algorithm 6 Distributed computation of $\mathcal{D}x$ with quantization

```
Input at node n: $x\big((s-1)N+n\big)$ for all $s=\{1,...,S\}, \mathcal{L}_{n,:}$, $\alpha=[\alpha_1;...;\alpha_S]$, quantization stepsize $\Delta$
Output at node n: $(\widetilde{\mathcal{D}x})(n)$
Quantize and transmit $\tilde{x}\big((s-1)N+n\big)=x\big((s-1)N+n\big)+\zeta_{s,l}(n)$, for $s=\{1,...,S\}$, to all $m\in\mathcal{N}_n$.
Receive $\tilde{x}\big((s-1)N+n\big)$, for all $s=\{1,...,S\}$, from $m\in\mathcal{N}_n$.
Set $z_{s,0}(n)=\tilde{x}\big((s-1)N+n\big)$, for all $s=\{1,...,S\}$.
for $k=2,...,K$ do:
    Compute $z_{s,k-1}(n)=(\mathcal{L}^T\tilde{z}_{s,k-2})(n)$, for all $s=\{1,2,...,S\}$.
    Quantize and transmit $\tilde{z}_{s,k-1}(n)=z_{s,k-1}(n)+\zeta_{s,k-1}(n)$ to all the neighbors, for all $s=\{1,2,...,S\}$.
    Receive $\tilde{z}_{s,k-1}(m)$ from all $m\in\mathcal{N}_n$, for all $s=\{1,2,...,S\}$.
end for
Compute $z_{s,K}(n)=(\mathcal{L}\tilde{z}_{s,K-1})(n)$, for all $s=\{1,...,S\}$.
Compute and output $(\widetilde{\mathcal{D}x})(n)=\sum_{s=1}^{S}\sum_{k=0}^{K}\alpha_{ks}z_{s,k}(n)$.
```

$k+1$ are defined as

$$z_{s,k+1}=\mathcal{L}^{k+1}z_{s,0}+\sum_{l=0}^{k}\mathcal{L}^{k+1-l}\zeta_{s,l}. \tag{5.10}$$

Using Eq. (5.10), we obtain the operator $\mathcal{D}x$ with quantization

$$\begin{aligned}\widetilde{\mathcal{D}x}&=\sum_{s=1}^{S}\mathcal{D}_s x_s=\sum_{s=1}^{S}\sum_{k=0}^{K}\alpha_{sk}z_{s,k}\\&=\sum_{s=1}^{S}\Big\{\sum_{k=0}^{K}\alpha_{sk}\mathcal{L}^k x_s+\sum_{l=0}^{K-1}\Big[\sum_{j=1}^{K-l}\alpha_{s(l+j)}\mathcal{L}^j\Big]\zeta_{s,l}\Big\}\\&=\sum_{s=1}^{S}[\mathcal{D}_s x_s+E(\mathcal{D}_s x_s)],\end{aligned} \tag{5.11}$$

where the accumulated quantization noise corresponding to subdictionary $\mathcal{D}_s$ is defined as

$$E(\mathcal{D}_s x_s)=\sum_{l=0}^{K-1}\Big[\sum_{j=1}^{K-l}\alpha_{s(l+j)}\mathcal{L}^j\Big]\zeta_{s,l}.$$

The main steps of the distributed algorithm for the computation of $\mathcal{D}x$ with quantized messages are shown in Algorithm 6.

5.3.3 Distributed computation of $\mathcal{D}^T\mathcal{D}x$ with quantization

Finally, we illustrate the distributed computation of $\mathcal{D}^T\mathcal{D}x$ with quantization, which follows the same reasoning as the previous two operators. In particular, we assume that the operator $\mathcal{D}x$ has already been sent and the corresponding entries of $\widetilde{\mathcal{D}x}$ are known to the sensors. Thus, the new variable in this case is defined as $z_0 = \widetilde{\mathcal{D}x}$. The sensors' values at iteration $k+1$ are

$$z_{k+1} = \mathcal{L}^{k+1} z_0 + \sum_{l=0}^{k} \mathcal{L}^{k+1-l} \xi_l, \tag{5.12}$$

where ξ_l is the quantization error vector $\xi_l = (\xi_l(1), \xi_l(2), ..., \xi_l(N))$ that occurs after transmitting $\tilde{z}_l$. By combining Eqs. (5.8), (5.11), (5.12), for each subdictionary $\mathcal{D}_s$, we can compute $\mathcal{D}_s^T\mathcal{D}x$ with quantization as follows,

$$\begin{aligned}
\widetilde{\mathcal{D}_s^T\mathcal{D}x} &= \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k z_k \\
&= \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k z_0 + \sum_{l=1}^{K} \Big[\sum_{j=1}^{l-1} \alpha_{sl} \mathcal{L}^{l-j} \xi_j \Big] \\
&= \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k \widetilde{\mathcal{D}x} + \sum_{l=0}^{K-1} \Big[\sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \Big] \xi_l \\
&= \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k \sum_{s'=1}^{S} \Big\{ \sum_{k'=0}^{K} \alpha_{s'k'} \mathcal{L}^{k'} x_{s'} + \sum_{l'=0}^{K-1} \Big[\sum_{j'=1}^{K-l'} \alpha_{s'(l'+j')} \mathcal{L}^{j'} \Big] \zeta_{s',l'} \Big\} + \sum_{l=0}^{K-1} \Big[\sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \Big] \xi_l \\
&= \mathcal{D}_s^T\mathcal{D}x + \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k E(\mathcal{D}x) + E(\mathcal{D}_s^T\mathcal{D}x),
\end{aligned} \tag{5.13}$$

where we have set

$$E(\mathcal{D}x) = \sum_{s'=1}^{S} \sum_{l'=0}^{K-1} \Big[\sum_{j'=1}^{K-l'} \alpha_{s'(l'+j')} \mathcal{L}^{j'} \Big] \zeta_{s',l'},$$

$$E(\mathcal{D}_s^T\mathcal{D}x) = \sum_{l=0}^{K-1} \Big[\sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \Big] \xi_l.$$

Finally, the operation $\widetilde{\mathcal{D}^T\mathcal{D}x} = \{\widetilde{\mathcal{D}_1^T\mathcal{D}x}\}_{s=1}^{S}$ can be written as

$$\widetilde{\mathcal{D}^T\mathcal{D}x} = \mathcal{D}^T\mathcal{D}x + \mathcal{D}^T E(\mathcal{D}x) + E(\mathcal{D}^T\mathcal{D}x),$$

where $E(\mathcal{D}^T\mathcal{D}x) = \{E(\mathcal{D}_s^T\mathcal{D}x)\}_{s=1}^{S}$. Again, we observe that there is an error accumulated from the computation of both steps $\mathcal{D}x$ and $\mathcal{D}^T\mathcal{D}x$ that depends on the quantization noise and the structure of the dictionary through the coefficients $\{\alpha_{sk}\}_{s=1,k=0}^{S,K}$. The main steps of the algorithm are shown

Algorithm 7 Distributed computation of $\mathcal{D}^T\mathcal{D}x$ with quantization

```
Input at node n: $\widetilde{(\mathcal{D}x)(n)}, \mathcal{L}_{n,:}$, $\alpha = [\alpha_1; ...; \alpha_S]$, quantization stepsize $\Delta$
Output at node n: $\widetilde{(\mathcal{D}^T\mathcal{D}x)}\big((s-1)N+n\big)$ for all $s = \{1, .., S\}$
Set $z_0(n) = \widetilde{\mathcal{D}x}(n)$.
Quantize and transmit $\tilde{z}_0(n) = z_0(n) + \xi_0(n)$ to all $m \in \mathcal{N}_n$.
Receive $\tilde{z}_0(m)$ from neighbors in $\mathcal{N}_n$.
for $k = 2, ..., K$ do:
    Compute $z_{k-1}(n) = (\mathcal{L}^T \tilde{z}_{k-2})(n)$.
    Quantize and transmit $\tilde{z}_{k-1}(n) = z_{k-1}(n) + \xi_{k-1}(n)$ to all the neighbors.
    Receive $\tilde{z}_{k-1}(m)$ from all the neighbors $m \in \mathcal{N}_n$.
end for
Compute $z_K(n) = (\mathcal{L}\tilde{z}_{k-1})(n)$.
for $s = \{1, .., S\}$ do
    Compute $\widetilde{(\mathcal{D}^T\mathcal{D}x)}\big((s-1)N+n\big) = \sum_{k=0}^{K} \alpha_{ks} z_k(n)$.
end for
```

in Algorithm 7.

5.4 Distributed sparse graph signal regularization with quantization

In the following, we use the above operators for the sparse distributed representation of a signal y with respect to a dictionary $\mathcal{D}$, under communication constraints. The sparse representation in a dictionary $\mathcal{D}$ can be found by solving a lasso minimization problem [143] of the form of Eq. (5.3). The first step of ISTA requires the computation of the gradient of the fitting term of Eq. (5.3) i.e.,$\|y - \mathcal{D}x\|^2$, which implies the computation of the operations $\mathcal{D}^T y$, $\mathcal{D}^T\mathcal{D}x$ in each iteration of the algorithm. When the messages exchanged by the sensors are quantized, the quantization noise induced by each of these operations introduces an error in the gradient, such that

$$\tilde{x}^{(t)} = \mathcal{S}_{\kappa\tau}\big(x^{(t-1)} + 2\tau\big(\mathcal{D}^T y - \mathcal{D}^T\mathcal{D}x^{(t-1)} + e^{(t-1)}\big)\big), \tag{5.14}$$

where $e^{(t-1)}$ is the total gradient error, which according to Eqs. (5.8), (5.11), (5.13) can be expressed as

$$e^{(t-1)} = E^{(t-1)}(\mathcal{D}^T y) - \mathcal{D}^T E^{(t-1)}(\mathcal{D}x) - E^{(t-1)}(\mathcal{D}^T\mathcal{D}x).$$

The convergence of the sparse graph signal representation by the ISTA algorithm then depends on the sequence of errors over the iterations. It can be characterized by the following result from [144] that applies to the general family of proximal gradient methods such as ISTA.

Theorem 1 ([144]). *Let f a differentiable with Lipschitz continuous gradient function on some compact set with Lipschitz constant L, g a lower semi-continuous and convex function, and $\{\tau_t\}$ a* *sequence of gradient stepsizes that satisfy the conditions:*

$$0 < \beta \leq \tau_t \leq \min(1, 2/L - \beta), \text{ with } 0 < \beta < \frac{1}{L}.$$

Then, the sequence generated by the iterates

$$x^{(t)} = prox_{\tau_{t-1}g}(x^{(t-1)} - \tau_{t-1}\nabla f(x^{(t-1)}) + \tau_{t-1}e^{(t-1)}) \tag{5.15}$$

converges to a stationary point x^* if, given a fixed $\bar{\tau}$ the following condition on the gradient error holds

$$\forall \tau \in (0, \bar{\tau}], \quad \tau\|e\| \leq \bar{\epsilon}, \quad \text{for some } \bar{\epsilon} \geq 0.$$

The above result indicates that the proximal gradient method converges to an approximate stationary point if the norm of the gradient error is uniformly bounded. Furthermore, if the number of perturbed gradient computations is finite, or if the gradient error norm converges towards 0, then the sequence limit point is the exact solution of the initial problem. Therefore, we have to make sure that the error in the gradient is bounded so that the distributed sparse graph signal representation algorithm converges.

Assume that we use a uniform quantizer in our distributed signal processing algorithm which is widely used and simple to implement. Let us further assume that our quantizer has a quantization stepsize Δ for the magnitude and one bit for the sign. With such a quantizer, an upper-bound on the norm of the error is given by the following lemma.

Lemma 1. *Let $e^{(t-1)}$ be the error due to quantization in the computation of the gradient at iteration $t-1$ as defined in Eq. (5.14) and Δ the quantization stepsize of a uniform quantizer. Then, the quantization error is bounded as*

$$\|e^{(t-1)}\| \leq \sqrt{N}\frac{\Delta}{2}\sum_{s=1}^{S}\Big\{2\sum_{l=0}^{K-1}\|\sum_{j=1}^{K-l}\alpha_{s(l+j)}\mathcal{L}^j\| + c\sum_{s'=1}^{S}\sum_{l'=0}^{K-1}\|\sum_{j'=1}^{K-l'}\alpha_{s'(l'+j')}\mathcal{L}^{j'}\|\Big\}. \tag{5.16}$$

Proof: For ease of notation, we ignore the iteration index and we bound the error norm as follows

$$\begin{aligned}\|e\| &= \|E(\mathcal{D}^T y) - \mathcal{D}^T E(\mathcal{D}x) - E(\mathcal{D}^T\mathcal{D}x)\| \\ &\leq \|E(\mathcal{D}^T y)\| + \|\mathcal{D}^T E(\mathcal{D}x)\| + \|E(\mathcal{D}^T\mathcal{D}x)\| \\ &\leq \sum_{s=1}^{S}\Big\{\|E(\mathcal{D}_s^T y)\| + \|\mathcal{D}_s^T E(\mathcal{D}x)\| + \|E(\mathcal{D}_s^T\mathcal{D}x)\|\Big\},\end{aligned} \tag{5.17}$$

where we have used the standard Cauchy-Schwarz inequality. For the sake of simplicity, we work with each term of the summation separately. After some basic operations with matrix norms, we can write the first term as follows

$$\begin{aligned}
\|E(\mathcal{D}_s^T y)\| &= \| \sum_{l=0}^{K-1} \Big[\sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \Big] \epsilon_l \| \\
&\leq \sum_{l=0}^{K-1} \| \Big[\sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \Big] \epsilon_l \| \\
&\leq \sum_{l=0}^{K-1} \| \sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \| \|\epsilon_l\|. \qquad (5.18)
\end{aligned}$$

Similarly, the second and the third term can be written as

$$\begin{aligned}
\|\mathcal{D}_s^T E(\mathcal{D}x)\| &= \| \sum_{l=0}^{K-1} \Big[\sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \Big] \xi_l \| \\
&\leq \sum_{l=0}^{K-1} \| \sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \| \|\xi_l\|, \qquad (5.19)
\end{aligned}$$

and

$$\begin{aligned}
\|E(\mathcal{D}_s^T \mathcal{D}x)\| &= \| \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k \sum_{s'=1}^{S} \sum_{l'=0}^{K-1} \Big[\sum_{j'=1}^{K-l'} \alpha_{s'(l'+j')} \mathcal{L}^{j'} \Big] \zeta_{s',l'} \| \\
&\leq \| \sum_{k=0}^{K} \alpha_{sk} \mathcal{L}^k \| \sum_{s'=1}^{S} \sum_{l'=0}^{K-1} \| \sum_{j'=1}^{K-l'} \alpha_{s'(l'+j')} \mathcal{L}^{j'} \| \|\zeta_{s',l'}\| \\
&\leq c \sum_{s'=1}^{S} \sum_{l'=0}^{K-1} \| \sum_{j'=1}^{K-l'} \alpha_{s'(l'+j')} \mathcal{L}^{j'} \| \|\zeta_{s',l'}\|, \qquad (5.20)
\end{aligned}$$

where the last inequality comes from the assumption that $0I \preceq \mathcal{D}_s \preceq cI$, which is generally true for the class of spectral graph dictionaries that we are considering. Combining Eqs. (5.17)-(5.20), and using the assumption that the quantization noise is uniformly distributed with magnitude smaller than $\Delta/2$, we obtain the upper bound of (5.16). □

The above inequality shows that the error at each iteration of the gradient is upper-bounded by the quantization error norms $\|\epsilon_l\|, \|\xi_l\|, \|\zeta_{s',l'}\|$, multiplied by a matrix polynomial of the graph Laplacian $\mathcal{L}$. The quantization errors depend on the number of bits, i.e., the rate constraints on data exchanged in the sensor network. For a uniform quantizer, the magnitude of the quantization error is upper-bounded by the quantization stepsize. In particular, if the norm $\| \sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \|$ is bounded by a constant $\eta > 0$ i.e., $\| \sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \| \leq \eta$ for $l \in \{1, ..., K-1\}$, $s \in \{1, ..., S\}$, Eq. (5.16) becomes:

$$\|e^{(t-1)}\| \leq \sqrt{N} \frac{\Delta}{2} SK(2+c)\eta. \qquad (5.21)$$

Thus, the error of the gradient at each iteration is bounded, which implies that ISTA converges to a stationary point of the iteration (5.15) according to Theorem 1. When $\Delta \rightarrow 0$, i.e., the bit rate tends to infinity, and the quantization noise tends to zero, $\|e^{(t-1)}\| \rightarrow 0$, independently of η. However, when the number of bits is limited and fixed, the quantization noise depends on the characteristics of the dictionary. In particular, as $\| \sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \|$ tends to 0 (i.e., $\eta \rightarrow 0$), the error also tends to 0 (i.e., $\|e^{(t-1)}\| \rightarrow 0$). Finally, the upper-bound on the error due to quantization in Eq. (5.21) indicates that the higher the degree K of the polynomial, the more the error is accumulated over the iterations. This is quite intuitive as higher polynomial degree requires more information to be exchanged between the sensors, at the cost of more propagation of the quantization noise. A large value of K at the same time guarantees that the polynomial functions can better approximate the underlying spectral kernels and thus the graph signals. It indicates that there is a tradeoff in the design of the dictionary, between the representation performance of the polynomial dictionary and the propagation of the quantization noise.

5.5 Polynomial dictionary learning with quantization

We use the study of the previous section to include the quantization parameter in the dictionary design and we introduce an algorithm to learn polynomial dictionaries that are robust to quantization noise. Our approach consists in controlling the norm of the total error in each step of the gradient computation when solving ISTA-based algorithms in a distributed way. When the quantization step size and the graph are given, the total error due to quantized communication can be controlled by choosing the proper values for the polynomial coefficients $\{\alpha_{sk}\}_{s=1,k=0}^{S,K}$ such that the gradient error stays bounded. From Eq. (5.16), the polynomial coefficients need to be computed in such a way that the spectral norm $\| \sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \|$, for $l \in \{1, ..., K-1\}$, is bounded for a fixed Δ. We recall that the spectral norm is defined as $\| \sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \| = \lambda_{max}\Big(\sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j \Big)$. Since the matrix $\sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j$ is symmetric, the spectral norm is simply its largest eigenvalue. Therefore, constraining the spectral norm becomes equivalent to constraining the eigenvalues of the corresponding matrix.

Based on the above analysis, we propose here to control the maximum eigenvalue of the matrix $\sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j$ for constructing dictionaries that are robust to the quantization noise. Namely, we choose the polynomial coefficients such that the spectral norm of $\sum_{j=1}^{K-l} \alpha_{s(l+j)} \mathcal{L}^j$ is bounded and small. We modify the learning algorithm of the previous chapter to have explicit control on the propagation of the quantization error. In details, given a set of training signals $Y = [y_1, y_2, ..., y_M] \in \mathbb{R}^{N \times M}$, all living on the weighted graph $\mathcal{G}$, our objective is to learn a polynomial graph dictionary $\mathcal{D} \in \mathbb{R}^{N \times NS}$, which can efficiently represent all of the signals in Y as linear combinations of only a few of its atoms and at the same time be robust to the quantization error, when applied to distributed setting with rate constraints. Since $\mathcal{D}$ has the form (5.1), our problem is equivalent to learning the parameters $\{\alpha_{sk}\}_{s=1,2,...,S;\ k=1,2,...,K}$ that characterize the set of generating kernels, $\{\widehat{g_s}(\cdot)\}_{s=1,2,...,S}$. We denote these parameters in vector form as $\alpha = [\alpha_1; ...; \alpha_S]$, where α_s is a column vector with $(K+1)$ entries. The optimization problem is formulated similarly to the one in Eq. (4.11). However, in order to take into account the effect of the quantization noise, we impose an additional constraint on the original problem of Eq. (4.11), which bounds the eigenvalues of the matrix $\sum_{j=1}^{K-l} \alpha_{s(l+j)}\mathcal{L}^j$ for $l \in \{1, ..., K-1\}$ and $s \in \{1, ..., S\}$.

Formally, the dictionary learning problem can be cast as the following optimization problem:

$$\begin{aligned}
\underset{\alpha\in\mathbb{R}^{(K+1)S},\ X\in\mathbb{R}^{SN\times M}}{\arg\min} \quad & \left\{||Y-\mathcal{D}X||_F^2 + \mu\|\alpha\|_2^2\right\} \\
\text{subject to} \quad & \|x_m\|_0 \leq T_0, \quad \forall m \in \{1, ..., M\}, \\
& \mathcal{D}_s = \sum_{k=0}^{K} \alpha_{sk}\mathcal{L}^k, \forall s \in \{1, 2, ..., S\} \\
& 0I \preceq \mathcal{D}_s \preceq cI, \quad \forall s \in \{1, 2, ..., S\} \\
& (c-\epsilon_1)I \preceq \sum_{s=1}^{S} \mathcal{D}_s \preceq (c+\epsilon_2)I, \\
& -\eta I \preceq \sum_{j=1}^{K-l} \alpha_{s(l+j)}\mathcal{L}^j \preceq \eta I, \quad \forall l \in \{1, ..., K-1\},\ \forall s \in \{1, ..., S\},
\end{aligned} \tag{5.22}$$

where $\mathcal{D} = [\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_S]$, x_m corresponds to column m of the coefficient matrix X, T_0 is the sparsity level of the coefficients of each signal, and I is the identity matrix. The additional constraints provide some control over the spectral representation of the atoms and the stability of signal reconstruction with the learned dictionary as discussed in Chapter 4. The optimization problem (5.22) is not convex, but it can be approximately solved in a computationally efficient manner by alternating between the sparse coding and dictionary update steps, as described in the previous chapter. In the first step, we fix the parameters α (and accordingly fix the dictionary $\mathcal{D}$) and solve the sparse coding step using orthogonal matching pursuit (OMP) [125]. In the second step, we fix the coefficients X and update the dictionary by finding the vector of parameters, α, that solves the polynomial coefficient update step using interior point methods [128].

We notice that the parameter η is a design parameter that intuitively should be chosen inversely proportional to the quantization stepsize according to Eq. (5.21). In particular, a small value of η penalizes the propagation of the quantization noise when the dictionary is used in distributed settings, at the cost however of a reduced flexibility in the search space of the polynomial coefficients. The latter implies a loss in the accurate recovery of the underlying spectral kernels that generate the true dictionary atoms. A large value of η gives more flexibility for the algorithm of (5.22) to learn a set of polynomial coefficients that are good for approximating the kernels in ideal communication settings, without though restricting the accumulated quantization noise. The effect of this tradeoff is studied in the experimental section.

5.6 Experimental results

We first study the performance of our dictionary learning algorithm for the distributed approximation of synthetic signals. We generate a graph by randomly placing $N = 500$ vertices in the unit square. We set the edge weights based on a thresholded Gaussian kernel function so that $W_{ij} = e^{-\frac{[dist(i,j)]^2}{2\theta^2}}$ if the physical distance $dist(i,j)$ between vertices i and j is less than or equal

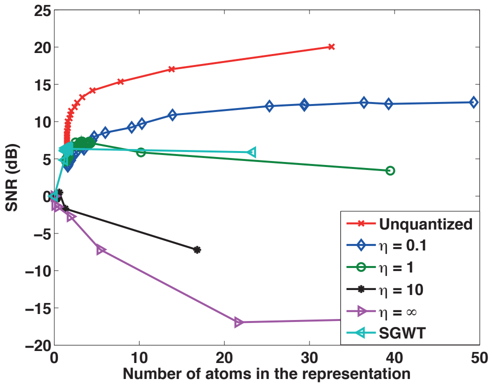
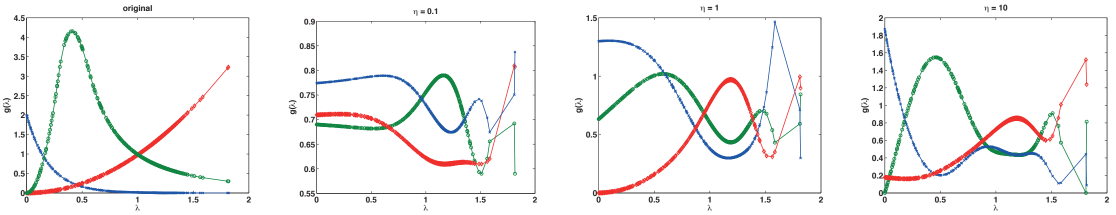


Figure 5.3: Distributed approximation performance (SNR) versus sparsity level, achieved with different polynomial graph dictionary learned with different values of η in distributed settings, with a bit rate of 6 bits per message.

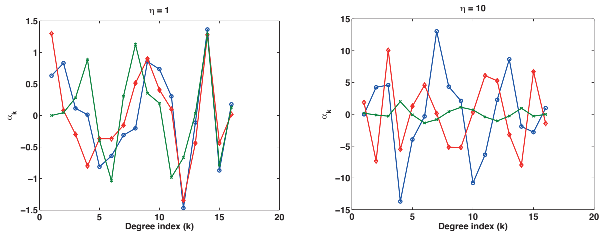
to κ, and zero otherwise. We fix $\theta = 0.04$ and $\kappa = 0.09$ in our experiments, and ensure that the graph is connected. In our first set of experiments, we construct a set of synthetic training signals consisting of localized patterns on the graph, drawn from a dictionary that is a concatenation of $S = 3$ subdictionaries. Each subdictionary is a polynomial of the graph Laplacian of degree $K = 15$ and captures one of the three constitutive components of our synthetic signal class. We generate the graph signals by linearly combining $T_0 \leq 10$ random atoms from the dictionary with random coefficients. We then learn a dictionary from a set of 1000 training signals for different values of the parameter η that controls the robustness to quantization error in distributed signal processing.

First, we study the distributed approximation of testing signals using the iterative soft thresholding algorithm with iterations defined by Eq. (5.14). The testing signals are generated in the same way as the training ones. We assume that the messages exchanged by the sensors are uniformly quantized before transmission. In particular, for each message we send one bit for the sign and quantize the magnitude of the data to be transmitted to neighbor sensors. For each signal y, the quantization range of the transmitted messages is defined to be $[0, \ \|y\|_\infty]$, and it is known by all the sensors. We fix the bit rate to 6 bits per message and we run ISTA for 300 iterations, and different values of the sparsity parameter κ in Eq. (5.14). We learn different polynomial dictionaries by solving the optimization problem (5.22) for different values of η. For the sake of comparison, we show also the approximation performance obtained with the spectral graph wavelet dictionary approximated by a Chebyshev polynomial of order $K = 30$. In Fig. 5.3, we illustrate the approximation performance in terms of SNR obtained for different numbers of atoms in the representation. The number of atoms is measured by counting the number of non-zero elements in the sparse codes for a particular value of κ. Interestingly, we observe that the best representation performance is obtained when η is very small. As we increase η, the effect of the quantization noise becomes significantly high, which leads to a dramatically low SNR in the distributed approximation algorithm. The worst performance is obtained when $\eta = \infty$, which is equivalent to ignoring the robustness constraint in the dictionary learning algorithm. This confirms that the robustness constraint can



(a) Original kernels (b) $\eta = 0.1$ (c) $\eta = 1$ (d) $\eta = 10$

Figure 5.4: Illustration of the kernels recovered for different values of η in the dictionary learning problem.



(a) $\eta = 1$ (b) $\eta = 10$

Figure 5.5: Polynomial coefficient values, for each of the dictionary kernels, for different values of η in solving the optimization problem (5.22).

indeed reduce the effect of the quantization noise. However, comparing to the performance obtained in the case of infinite bit rate (red curve in Fig. 5.3), we observe a saturation in the maximum SNR that is significantly lower than the one in ideal communication conditions. This is the price to pay for introducing the quantization constraint and reducing the search space in the dictionary learning problem of (5.22).

We look in more details on the effect of η on the dictionary learning outcome and study the effect of this parameter in the learned kernels. In Fig. 5.4(a), we illustrate the original kernels of the underlying dictionary, and in Figs. 5.4(b)-5.4(d), we plot the ones recovered by solving the dictionary learning algorithm of Eq. (5.22) for different values of η. As expected, we observe that, as we increase the value of η, the recovered kernels become closer to the original ones. The effect of the parameter η is more obvious in Fig. 5.5, where we plot the values of the polynomial coefficients. We observe that, when η is small, the polynomial coefficients become small in magnitude. As a result, the dictionary learning algorithm is not able to capture relatively complicated kernels, which seems to be the cost for improved robustness to quantization as shown in Section 5.5.

In the next set of experiments, we quantify the loss in the approximation performance by focusing on centralized settings without rate constraints. We approximate 1000 testing signals, generated in the same way as the training signals, by computing the sparse approximation in the learned dictionaries with OMP, for different sparsity levels. For the sake of comparison, we also compute the approximation performance achieved by applying OMP on the spectral graph wavelet dictionary [3]. The obtained results are illustrated in Fig. 5.6. Each point in the curve corresponds to the signal to approximation noise ratio (SNR in dB) for different sparsity levels,

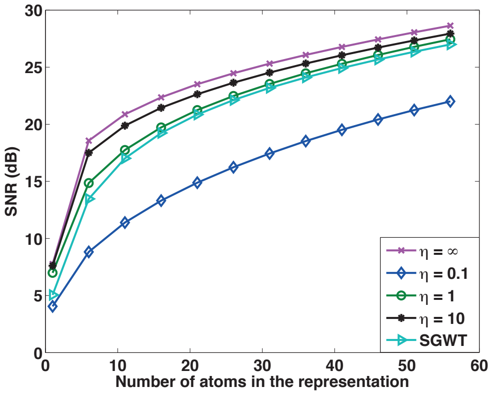


Figure 5.6: Approximation performance (SNR) versus sparsity level, achieved with the polynomial graph dictionary for different values of η in centralized settings.

and it is computed as the average value over all the testing signals. As we reduce the values of the parameter η in the dictionary learning algorithm, the approximation performance in the ideal scenario of infinite bit rate deteriorates significantly. It can become even worse than the one achieved with the spectral graph wavelet dictionary, which is not learned to efficiently represent the training signals. This behavior is consistent with the conclusion drawn from Figs. 5.4, 5.5. The more we reduce the search space (i.e., the smaller the value of η), the worse is the approximation performance of the graph signals from the learned dictionary.

In another set of experiments, we apply our dictionaries in distributed denoising applications. We add some Gaussian noise to the testing signals such that their initial SNR is 10 dB. We then use the dictionaries learned with the different parameters η to denoise the signals by imposing a sparse prior. Denoising is performed by applying the iterative soft thresholding algorithm of Eq. (5.14) for different values of the parameter κ. In Fig. 5.7, we illustrate the SNR obtained after distributed denoising with different numbers of bits per messages, for each of the learned dictionaries. For each bit rate, we keep the value of κ that corresponds to the highest SNR. The obtained results indicate that a small η at low bit rate can bring significant gain in terms of denoising performance. When the bit rate is high, the denoising performance obtained with the dictionary corresponding to a small η ($\eta = 0.1$) saturates to a low SNR value. Due to the quantization constraints, the solution of the optimization problem (5.22) is not necessarily the optimal, and the representation performance of the dictionary is reduced. These results are consistent with the one obtained in the previous experiments.

Finally, we study the application of the proposed framework to the denoising of real world signals. We consider the daily traffic bottlenecks in San Francisco that are parts of the Caltrans Performance Measurement System (PeMS) dataset [134].[2] It contains measurements of 75 detector stations between January 2007 and August 2014. The graph is designed by connecting stations when the distance between them is smaller than a threshold of $\theta = 0.04$. For two stations A, B, the

---

[2] The data are publicly available at http://pems.dot.ca.gov.

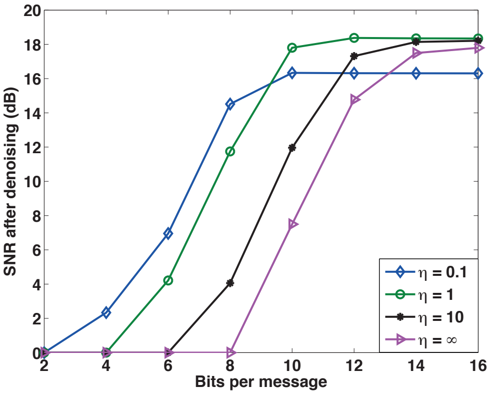


Figure 5.7: Denoising performance (in dB) versus bits per message using dictionaries learned with different values of η. The SNR of the input signals is 10 dB.

distance d_{AB} is set to be the Euclidean distance of the GPS coordinates of the stations and the edge weights are computed using the exponential kernels such that $W_{AB} = e^{-d_{AB}}$. The signal on the graph is the duration in minutes of bottlenecks for each specific day. These signals are normalized for computational issues with the maximum signal norm.

We use half of the signals to learn a polynomial dictionary with $S = 3$ subdictionaries, and a maximum polynomial degree of $K = 15$. The sparsity level in the learning phase is set to $T_0 = 5$. We run the dictionary learning algorithm for different values of the parameter $\eta = [0.1,\ 1,\ 10,\ \infty]$. We add random Gaussian noise to the rest of the signals to obtain a set of noisy signals, with SNR equal to 10 dB. The obtained dictionaries are then used for denoising the noisy signals in distributed settings with the iterative soft thresholding algorithm, at different bit rates. The results are illustrated in Fig. 5.8. As in the case of the synthetic data, we observe at low bit rate a significant performance gain when the dictionary has been learned with a small value of the robustness parameter η. When η becomes large, denoising of the signals by 2 dB requires approximately 8 bits per message, which is much higher than the number of bits required by a dictionary learned with a small η, for the same denoising performance.

5.7 Conclusions

In this chapter, we have studied the effect of quantization in distributed graph signal processing with polynomial dictionary operators. We have shown analytically that the overall quantization error depends on the graph geometry and the dictionary structure. Following this observation, we have then proposed an algorithm that learns graph dictionaries to sparsely approximate graph signals while staying robust to quantization noise. Experimental results have illustrated the trade-offs between effective distributed signal representation in low bit rate communication settings and accuracy of the signal approximation in ideal settings. An interesting extension of this work would be the analysis of the distributed processing performance with loss of information as it happens in practical networks. The study of the design of adaptive quantization algorithms that would yet

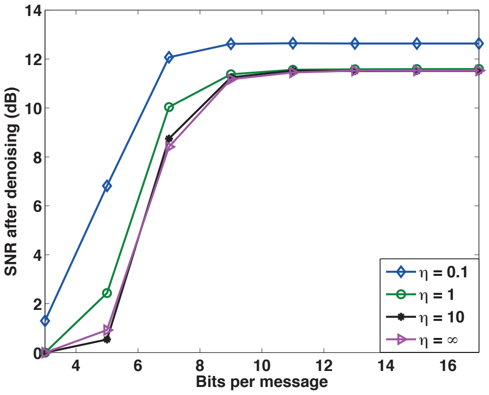


Figure 5.8: Denoising performance (in dB) versus bit rate per message for the traffic dataset using dictionaries learned with different values of η. The initial SNR is 10 dB.

improve the distributed graph signal processing performance is also an interesting research problem. In the next chapter, we focus on a specific distributed graph process, the average consensus process, and we propose an adaptive quantization scheme that goes along the second direction.

Chapter 6

Graph-based Quantization Refinement for Distributed Consensus

6.1 Introduction

In this chapter, we focus on a particular class of graph signals that are generated from an average consensus process. This type of processes, although quite simple, have attracted a lot of research interest due to their applications in wireless network systems. Distributed consensus algorithms [145] are mainly used in ad-hoc sensor networks in order to compute the global average of sensor data in a distributed fashion, using only local inter-sensor communication. Some of their most important applications include distributed coordination and synchronization in multi-agent systems [48], distributed estimation [146], distributed classification [50] and distributed control problems.

The distributed average consensus problem has been typically addressed in the literature through the successive application of local graph filtering operators. It results in a process evolving on the graph that converges to a smooth graph signal that takes the average value of the original data. The choice of the graph filter, also known as the consensus weights, defines the convergence rate of the algorithm. These weights are typically designed based on the structure of the graph, i.e., the graph weights and the graph Laplacian matrix (e.g., Maximum-degree, Metropolis, Laplacian weights) [47] or on convergence criteria. For example, graph filters with polynomial structures have been shown to accelerate the convergence of the average consensus algorithms [59], [147], [61].

While in theory convergence to the global average is mostly dependent on the sensor network topology, the performance of distributed average consensus algorithms in practical systems is largely connected to the communication constraints and limited precision operations. In general, the information exchanged by the network nodes has to be quantized prior to transmission due to limited communication bandwidth and limited computational power. However, this quantization process induces some quantization noise that is accumulated throughout the iterative consensus algorithm and affects its convergence, leading to significant performance degradation [148].

In this chapter, we design a novel distributed progressive quantization algorithm[1] that limits the quantization noise and leads to convergence to the average value even at low bit rates [149],

---

[1] D. Thanou, E. Kokiopoulou, Y. Pu, and P. Frossard. Distributed average consensus with quantization refinement, *IEEE Trans. on Signal Proc.*, vol. 61, no.1, pp. 194-205, Jan. 2013.

[150]. Motivated by the observation that the correlation between the values communicated by the nodes increases along the consensus iterations, we propose to progressively and consistently reduce the range of the quantizer in order to refine the information exchanged in the network and guide the sensors to converge to the average consensus value. The proposed quantization scheme is computationally simple and consistent throughout the iterations as every node implements the same quantization all the time. We describe a method for computing offline the parameters of the quantizers, which depend on the network topology and the communication constraints. Our design is based on an average case analysis, which leads to effective performance in realistic settings. Convergence of the consensus iterations is achieved when the energy of the quantization error decreases with the iterations. We illustrate the performance of the proposed scheme through simulations that demonstrate that the consensus algorithm converges fast to the average value even in the case where the information is hardly quantized.

The structure of the chapter is as follows. Section 6.2 presents our new progressive quantization algorithm. A recursive method for computing the quantization parameters is proposed in Section 6.3, followed by a simple exponential relation for adapting the quantizer step-size. Then, simulations results are presented in Section 6.4. Finally, in Section 6.5 we review the existing work in the literature related to the effect of quantization in the distributed average consensus problem.

6.2 Progressive quantizer for distributed average consensus

We consider a sensor network topology that is modeled as a weighted, undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$, where $\mathcal{V} \in \{1, \ldots, N\}$ represents the sensor nodes and $N = |\mathcal{V}|$ denotes the number of nodes. An edge denoted by an unordered pair $\{i, j\} \in \mathcal{E}$, represents a link between two sensor nodes i and j that communicate with each other. Moreover, a positive weight $W_{ij} > 0$ is assigned to each edge if $\{i, j\} \in \mathcal{E}$. The set of neighbors for node i is finally denoted as $\mathcal{N}_i = \{j | \{i, j\} \in \mathcal{E}\}$.

The node states over the network at time t can be expressed as a vector $z_t = [z_t(1), \ldots, z_t(N)]^T$, where $z_t(i)$ represents a real scalar assigned to node i at time t. The distributed average consensus problem consists in computing iteratively at every node the average

$$\mu = \frac{1}{N} \sum_{i=0}^{N} z_0(i),$$

where $z_0(i)$ is the initial state at sensor i. The consensus can be achieved by linear iterations of the form $z_{t+1} = W z_t$, where the symmetric weight matrix W satisfies the conditions that are required to achieve asymptotic average consensus [47], expressed as

$$\mathbf{1}^T W = \mathbf{1}^T, \; W\mathbf{1} = \mathbf{1}, \; \rho(W - \mathbf{1}\mathbf{1}^T/N) < 1, \tag{6.1}$$

with $\rho(\cdot)$ the spectral radius of the matrix and $\mathbf{1}$ is the vector of ones.

When communication rate is limited between sensors, the value $z_t(i)$ of a sensor node i at each step t is quantized prior to its transmission to neighbor nodes. The quantized value $\tilde{z}_t(i)$ can be written as

$$\tilde{z}_t(i) = z_t(i) + \epsilon_t(i), \tag{6.2}$$

where $\epsilon_t(i)$ models the additive quantization noise of sensor i at iteration t. In particular, in the case of a n-bit uniform quantizer, the quantized values can be written as

$$\tilde{z}_t(i) = \left\lfloor \frac{z_t(i) - z_0^{(\min)}}{\Delta} \right\rfloor \cdot \Delta + \frac{\Delta}{2} + z_0^{(\min)},$$

when the initial sensor states lie in a finite interval of size $S = z_0^{(\max)} - z_0^{(\min)}$. The parameters $z_0^{(\min)}$ and $z_0^{(\max)}$ represent the minimum and the maximum values of the interval respectively. The parameter $\Delta = S/2^n$ is the quantization step-size, which drives the error of the quantizer.

In the presence of quantization noise in the distributed average consensus algorithm, we use the following linear iterations that preserve the average of the initial states [151]

$$z_{t+1} = z_t + (W - I)\tilde{z}_t, \tag{6.3}$$

where I is the identity matrix. An analytical expression of Eq. (6.3) shows that the quantization error propagates through the iterations of the consensus algorithm. More specifically, the states z_{t+1} and $\tilde{z}_t$ are expressed as

$$\tilde{z}_t = W^t z_0 + \sum_{s=0}^{t-1} W^s (W - I)\epsilon_{t-s-1} + \epsilon_t \tag{6.4}$$

$$z_{t+1} = W^{t+1} z_0 + \sum_{s=0}^{t} W^s (W - I)\epsilon_{t-s}. \tag{6.5}$$

The linear iterations defined by Eq. (6.3) preserve the average of the initial states in the network but unfortunately do not guarantee convergence to the average consensus value in the presence of quantization. In order to limit the influence of the quantization noise, we should decrease the step size by either increasing the number of bits or adapting the quantization range for the same number of bits. In the average consensus problem, it can be observed that as the number of iterations increases, the correlation between the sensors' states increases and the values computed by the sensors tend to converge into an interval of decreasing size. Quantization in the full range of size S hence results in a waste of bits or in limited precision that prevents the algorithm to converge to the true average value. We therefore propose to adapt the quantization step-size as the number of linear iterations increases in a new progressive quantization algorithm. We keep a simple uniform quantizer with a fixed number of bits per sensor and we adapt the quantization range so that quantization becomes finer along the iterations.

In more detail, we denote the size of the range of the quantizer in node i at time t as $S_t(i)$. Since the size of the quantization range is always positive, we impose $S_t(i) > 0$. This range decreases in each sensor as the iterations proceed. The quantization range is further centered around the previous state of the consensus algorithm $\tilde{z}_{t-1}(i)$ as the values of the consensus algorithm converge over time to a smooth graph signal. More formally, the sensor i encodes its state $z_{t+1}(i)$ by using a quantization interval that is defined as $[\tilde{z}_t(i) - S_{t+1}(i)/2, \tilde{z}_t(i) + S_{t+1}(i)/2]$. The data is uniformly quantized in this reduced range, which leads to a step-size

$$\Delta_{t+1} = \frac{S_{t+1}(i)}{2^n} \tag{6.6}$$

that decreases over time. The values falling out of the quantization interval are clipped and coded to the nearest quantizer value. In order to simplify the design of the quantizer for realistic settings, we impose the size of this interval to be identical for all the sensors, independently of their previous state and their position in the network (i.e., $S_t(i) = S_t, \quad \forall i = 1, \dots, N$). Since each neighbor node $j \in \mathcal{N}_i$ knows the value $\tilde{z}_t(i)$ received at the previous iteration, it is able to perform inverse quantization and to compute correctly the value $\tilde{z}_{t+1}(i)$. We call the proposed quantization scheme Progressive Quantizer.

The important parameter in the Progressive Quantizer algorithm is clearly the size S_t of the quantization range. It has to be small enough such that the precision of the quantized information is sufficient for convergence to the true average value. On the other hand, it should be chosen large enough such that the values computed in the network nodes fall in the quantization range with high probability in order to avoid clipping that may negatively affect the convergence of the average consensus algorithm.

6.3 Design of the parameters of the progressive quantizer

6.3.1 Average case analysis

In this section, we propose a constructive methodology to compute a priori the size of the quantization range, based on the properties of the network graph topology and the communication constraints. In order to guarantee that the quantizer does not saturate (i.e., all the values fall inside the quantization range), S_{t+1} should satisfy the following inequality

$$\|z_{t+1} - \tilde{z}_t\|_\infty \leq \frac{S_{t+1}}{2}. \tag{6.7}$$

The computation of the worst case interval based on (6.7) typically leads to conservative progressive quantizer design that does not necessarily lead to a better performance than the classical uniform quantizer with a constant range. This observation is supported by simulations in subsection 6.4.3 where a conservative design [8] is unable to lead to fast convergence. Instead of looking for strong guarantees such as those in (6.7), we build our quantizer such that values fall in the quantization range with high probability. This comes at a price of some potential clipping, which however does not significantly penalize the convergence of the algorithm. Moreover, limiting a priori the dynamic range of the sensors' states in a meaningful way is expected to prevent the consensus algorithm from being affected by potential outliers that occur due to quantization noise. The overall procedure could be characterized as an attempt to guide the quantized consensus algorithm such that it converges to the average value with a rate that gets close to the convergence rate of the ideal unquantized consensus algorithm. Since the convergence rate depends on the weight matrix of the graph, the design of the quantizer should also depend on the graph topology.

In more detail, we propose to relate the quantizer range size to the mean square difference

$\|z_{t+1}-\tilde{z}_t\|^2/N$ between two successive graph signals. It leads to the following average case condition

$$\frac{E[\|z_{t+1}-\tilde{z}_t\|^2]}{N} \leq (\frac{S_{t+1}}{2})^2, \tag{6.8}$$

where $\|\cdot\|$ denotes the L2 norm. The expectation of the difference between successive values in the consensus algorithm represents the minimal size of the quantization range at each iteration. Moreover, we model the quantization noise samples as spatially and temporally independent random variables that are uniformly distributed with zero mean and variance $\Delta_t^2/12$. In the sequel, we first derive in Proposition 2 an upper-bound of $E[\|z_{t+1}-\tilde{z}_t\|^2]$ that depends on the previous values $\{S_1, ..., S_t\}$. This upper-bound together with (6.8) permits to estimate S_{t+1}.

Proposition 2. *Let $\tilde{z}_t$ and z_{t+1} be defined as in Eqs. (6.4), (6.5). Let also λ_2 be defined as $\lambda_2 := \rho(W - \frac{\mathbf{1}\mathbf{1}^T}{N})$ and $\lambda_{\min}$ be the smallest algebraically eigenvalue of graph weight matrix W. Then, it holds that*

$$\begin{aligned} E[\|z_{t+1}-\tilde{z}_t\|^2] \leq \|z_0\|^2\lambda_2^{2t}(1-\lambda_{\min})^2 + (1-\lambda_{\min})^2\sum_{s=0}^{t-1}\|W^s(W-I)\|^2 N\frac{S_{t-s-1}^2}{2^{2n}\cdot 12} \\ + (2-\lambda_{\min})^2 N\frac{S_t^2}{2^{2n}\cdot 12}. \end{aligned} \tag{6.9}$$

The proof of Proposition 2 is given in Appendix D.1.

Then, Eqs. (6.8) and (6.9) along with the fact that $\|z_0\|^2 \leq N\|z_0\|_\infty^2$, imply that

$$\begin{aligned} (\frac{S_{t+1}}{2})^2 = \|z_0\|_\infty^2\lambda_2^{2t}(1-\lambda_{\min})^2 + (1-\lambda_{\min})^2\sum_{s=0}^{t-1}\|W^s(W-I)\|^2\frac{S_{t-s-1}^2}{2^{2n}\cdot 12} \\ + (2-\lambda_{\min})^2\frac{S_t^2}{2^{2n}\cdot 12}, \quad t \geq 1. \end{aligned} \tag{6.10}$$

The computation of the quantizer range size with (6.10) implies a recursive computation of S_t at each time step t of the consensus algorithm. We first set S_0 according to the initial range of the quantizer i.e., $S_0 = z_0^{(\max)} - z_0^{(\min)}$ and we compute S_1 from a simplified version of (6.10) where the intermediate term from the right hand side is dropped. Then S_{t+1} is computed recursively according to (6.10), where only positive solutions are kept.

Finally, we note that the terms used in the recursive computation of the quantization range reflect the characteristics of the network and the communication constraints. The values of the estimated quantization range depend on the convergence rate of the average consensus algorithm in the absence of quantization noise λ_2, on the maximum value of the initial data $\|z_0\|_\infty$, on the graph topology W (through $\lambda_2, \lambda_{\min}$) and on the number of quantization bits n for each sensor. Moreover, we exploit the properties of the weight matrix W by taking into account the averaging effect over the successive iterations. We show in the next section that the recursive computation of the quantization range size can be approximated with a simple exponential model.

6.3.2 Exponential quantization range

We build on the convergence behavior of the consensus algorithm and propose an approximate exponential model for the computation of the size of the quantization range. We first pose without loss of generality that $S_t = 2 \cdot e^{-\beta_t}$. In what follows, we show that under a few simplifying assumptions the recursive relation of the previous section leads to an exponential model whose parameters can be determined in closed form. This closed form parameter computation is of great benefit towards deployment in realistic settings.

When $S_t = 2 \cdot e^{-\beta_t}$, Eq. (6.10) becomes

$$\begin{aligned} e^{-2\beta_{t+1}} = \|z_0\|_\infty^2 \lambda_2^{2t}(1-\lambda_{\min})^2 + (1-\lambda_{\min})^2 \sum_{s=0}^{t-1} \|W^s(W-I)\|^2 \frac{e^{-2\beta_{t-s-1}}}{2^{2n}\cdot 3} \\ + (2-\lambda_{\min})^2 \frac{e^{-2\beta_t}}{2^{2n}\cdot 3}, \quad t \geq 1. \end{aligned} \tag{6.11}$$

The second term in the right-hand side of Eq. (6.11) is due to the accumulated quantization error from the previous t iterations. In particular, the matrix norm $\|W^t(W-I)\|$ that multiplies the quantization noise vectors decays asymptotically to zero. In addition to that, the sequence defined by Eq. (6.11) decays to zero as specified by the following proposition.

Proposition 3. *Let $e^{-\beta_{t+1}}$ be a sequence defined by Eq. (6.11). If the condition $\frac{(1-\lambda_{\min})^4}{(1-\lambda_2^2)} + (2-\lambda_{\min})^2 < 3 \cdot 2^{2n}$ is satisfied, then $\lim_{t\to\infty} e^{-\beta_{t+1}} = 0$.*

The Proposition 3, whose proof is given in Appendix D.2, relates the decay and convergence of the size of the quantizer range to the characteristics of the network graph (through $\lambda_2, \lambda_{\min}$) and to the number of bits used in the quantization. Eventually, it means that the expected squared difference between consecutive iterations in the distributed consensus algorithm as defined by the recursive equation (6.11) goes to zero as long as the number of bits satisfies $n > \frac{\log[\frac{1}{3}(\frac{(1-\lambda_{\min})^4}{(1-\lambda_2^2)} + (2-\lambda_{\min})^2)]}{2\log 2}$. Then, the following proposition shows that the second term in the right-hand side of Eq. (6.11) becomes negligible when the number of iterations increases.

Proposition 4. *Let W be a graph matrix satisfying the conditions defined in (6.1). Let $e^{-\beta_0}, e^{-\beta_1}, ..., e^{-\beta_t}$ be a sequence such that $e^{-\beta_t} \leq \delta, \forall t$ and $\lim\limits_{t\to\infty} e^{-\beta_t} = 0$. Then $\sum_{s=0}^{t-1} \|W^s(W-I)\|^2 e^{-2\beta_{t-s-1}}$ converges asymptotically to zero for $t \to \infty$.*

The proof or Proposition 4 is given in Appendix D.3 and it is based on the proof of Lemma 5.1 in [152].

Due to the presence of the factor $1/2^{2n}$, the second term of the right-hand side of Eq. (6.11) decreases faster to zero when the quantization rate is high. This is expected, as this term captures the propagation of the quantization error. Eq. (6.11) tends to follow an exponential model in this case. We can thus pose $\beta_t = \alpha \cdot t + \gamma$, which leads to an exponential decay of the size of the quantization range. We assume that there exists an iteration t_0 where the second term of the right-hand side of Eq. (6.11) becomes zero. Under this assumption, Eq. (6.11) simplifies to

$$e^{-2\beta_{t+1}} = \|z_0\|_\infty^2 \lambda_2^{2t}(1-\lambda_{\min})^2 + (2-\lambda_{\min})^2 \frac{e^{-2\beta_t}}{2^{2n}\cdot 3}, \quad t \geq t_0, \tag{6.12}$$

and by substituting $\beta_t = \alpha \cdot t + \gamma$ we obtain

$$e^{-2(\alpha\cdot(t+1)+\gamma)} = \|z_0\|_\infty^2 \lambda_2^{2t}(1-\lambda_{\min})^2 + (2-\lambda_{\min})^2 \frac{e^{-2(\alpha\cdot t+\gamma)}}{2^{2n}\cdot 3}, \quad t \geq t_0. \tag{6.13}$$

Since α and γ are constant over the iterations, we choose to determine them from later iterations of the consensus (i.e., $t \geq t_0$). First, we turn λ_2 into an exponential form by determining α such that $\lambda_2^{2t} = e^{-2\alpha\cdot t}$ holds. This leads to

$$\alpha = -\log(\lambda_2). \tag{6.14}$$

Eq. (6.12) then becomes

$$\begin{aligned} e^{-2\beta_{t+1}} &= \|z_0\|_\infty^2 e^{-2\alpha\cdot t}(1-\lambda_{\min})^2 + (2-\lambda_{\min})^2 \frac{e^{-2\beta_t}}{2^{2n}\cdot 3} \\ &= e^{-2\beta_t}\left(\|z_0\|_\infty^2 e^{2\gamma}(1-\lambda_{\min})^2 + \frac{(2-\lambda_{\min})^2}{2^{2n}\cdot 3}\right), \quad t \geq t_0. \end{aligned} \tag{6.15}$$

Since β_t is linear in t, we observe that $e^{-2\beta_{t+1}} = e^{-2(\alpha\cdot(t+1)+\gamma)} = e^{-2\alpha}e^{-2\beta_t}$, which permits to simplify (6.15) to

$$e^{-2\alpha} = \|z_0\|_\infty^2 e^{2\gamma}(1-\lambda_{\min})^2 + \frac{(2-\lambda_{\min})^2}{2^{2n}\cdot 3}.$$

We finally determine γ as

$$\gamma = \frac{1}{2}\log\left(\lambda_2^2 - \frac{(2-\lambda_{\min})^2}{2^{2n}\cdot 3}\right) - \log\left(\|z_0\|_\infty(1-\lambda_{\min})\right). \tag{6.16}$$

We observe that the decay rate α of the exponential function $e^{-(\alpha\cdot t+\gamma)}$ depends on λ_2, which characterizes the convergence rate of the average consensus algorithm in the case of non-quantized communication. On the other hand, the parameter γ, apart from the eigenvalues of W, depends also on the number of quantization bits. It is interesting to note that the parameters of the exponential model are similar to the parameters α', γ' that characterize the difference between two consecutive time steps in the unquantized consensus problem. In this case, the Euclidean difference between two consecutive time steps is guaranteed to be reduced by the factor $\|W - \mathbf{1}\mathbf{1}^T/N\| < 1$ at each iteration i.e.,

$$\begin{aligned} \frac{\|z_{t+1}-z_t\|}{\sqrt{N}} &= \frac{\|Wz_t - Wz_{t-1}\|}{\sqrt{N}} \leq \frac{\|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|\|z_t - z_{t-1}\|}{\sqrt{N}} \\ &\leq \frac{1}{\sqrt{N}}\|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^t\|z_1 - z_0\| \leq \frac{1}{\sqrt{N}}\|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^t\|W - I\|\|z_0\| \\ &\leq \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^t\|W-I\|^2\|z_0\|_\infty, \end{aligned} \tag{6.17}$$

where we observe an exponential decrease over time of the form $e^{-(\alpha' \cdot t+\gamma')}$, where $\alpha' = -\log(\|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|)$ and $\gamma' = -\log\left(\|\|z_0\|_\infty\|\|W - I\|\right)$. In particular, the rate at which the exponential function decays is the same in both cases ($\alpha' = \alpha = -\log(\lambda_2)$) while, in the case of quantization, the initial value γ of this decay depends on the number of bits (which is not the case for γ').

In summary, with our progressive quantizer based on the exponential decay of the quantization range, we force the average difference in the sensors' states between two consecutive iterations to decay with the same rate as in the ideal communication. At the same time, we allow the initial magnitude of this difference to be higher for a smaller bit rate in order to take into consideration the quantization error. We emphasize that the above exponential model for S_t yet reduces the complexity of the proposed Progressive Quantizer. Instead of estimating a priori the values of the quantization range for a large number of iterations, the system can simply use the parameters α and γ of the exponential model.

6.4 Simulation results

In this section we analyze the performance of the Progressive Quantizer in different settings. We consider a network of 40 sensors (i.e., $N = 40$) following the random geographic graph model, i.e., the sensors are uniformly random distributed over the unit square $[0, 1] \times [0, 1]$. We assume that two neighbor sensors are connected if their Euclidean distance is less than $r = \sqrt{(\log N)/N}$, which ensures graph connectivity with high probability [153]. We consider static network topologies, which means that the edge set does not change over the iterations. As an illustration, we consider the Metropolis and the Laplacian weight matrices [47] defined respectively as:

- Metropolis weights

$$W_{ij} = \begin{cases} \frac{1}{1+\max\{d(i),d(j)\}}, & \text{if } \{i,j\} \in \mathcal{E} \\ 1 - \sum_{(i,k)\in\mathcal{E}} W_{ik}, & \text{if } i = j \\ 0, & \text{otherwise,} \end{cases} \tag{6.18}$$

  where $d(i)$ denotes the degree of the i^{th} sensor.

- Laplacian weights

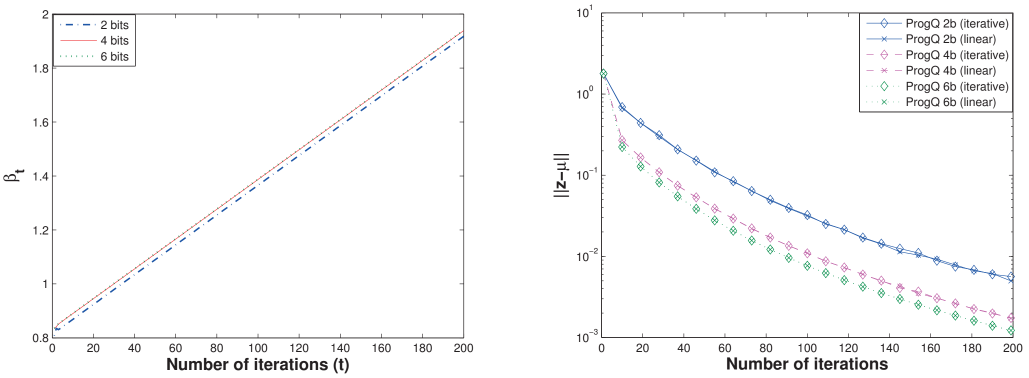
$$W = I - aL \tag{6.19}$$

  where L denotes the Laplacian matrix of the graph $\mathcal{G}$ and the scalar a must satisfy $0 < a < 1/d_{max}$, where d_{max} consists of the maximum degree in the network.

Moreover, the initial states of the sensors are uniformly distributed in the range $[0, 1]$.

6.4.1 Performance of the approximate exponential model

We first validate the exponential model for S_t and we use the Metropolis weight matrix for this experiment. We compute recursively the values S_t from Eq. (6.10) for 200 random realizations of a random network graph topology and communication rates of $n = [2, 4, 6]$ bits. For implementation issues, we fix a parameter $\delta = 10^{-16}$. At iteration t, if the quantization range S_t is smaller than δ, we quantize with the range computed at the previous iteration i.e., we set $S_t = S_{t-1}$. All the reported



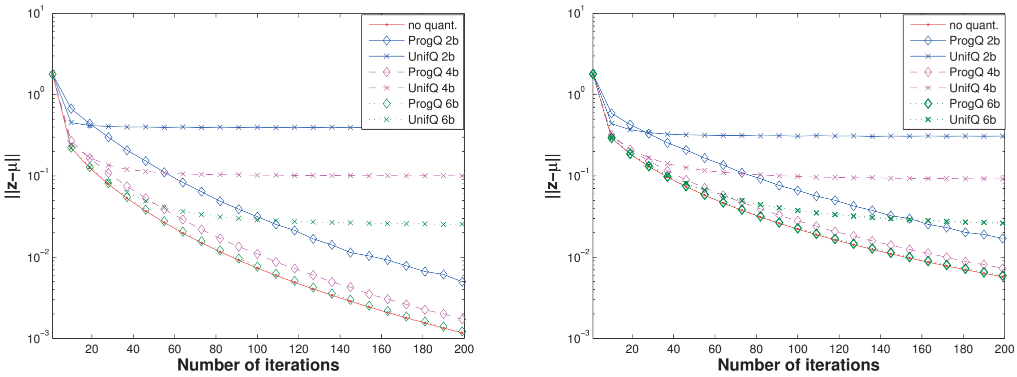
(a) (b)

Figure 6.1: (a) Evolution of the β_t values over the iterations. (b) Comparison of the average consensus performance of the Progressive Quantizer with parameters generated (i) recursively and (ii) with a linear model.

experimental results are averaged over the 200 random realizations of the network topology. In order to compare directly with the proposed exponential model in Section 6.3.2, we plot the values β_t that occur when we express the quantization range, computed from Eq. (6.10), as $S_t = 2 \cdot e^{-\beta_t}$. We observe in Fig. 6.1(a) that the value of β_t appears to increase linearly with the number of iterations, which means that the quantization range follows an exponential function that decreases over time. Moreover, the slope of the function β_t is independent of the bitrate, while the y-intercept value depends on the number of quantization bits. This is consistent with our approximate model (see Section 6.3.2) and Eqs. (6.14) and (6.16), which shows that the slope α is dependent on the convergence rate λ_2 (and hence the graph topology) and that the parameter γ is influenced by the communication rate. We further compare the performance of the Progressive Quantizer whose parameters are computed using the approximate linear model (from Eqs. (6.14) and (6.16)) with the performance achieved when S_t is computed recursively from Eq. (6.10). Fig.(6.1(b)) shows the obtained results. We observe that the performance is rather similar in both cases. This implies that the solutions of Eq. (6.10) can be well approximated with an exponential model whose parameters are easily computed. For this reason, in the rest of our experiments we adopt the approximate exponential model and compute the parameters α and γ of the Progressive Quantizer using Eqs. (6.14) and (6.16).

6.4.2 Comparison to uniform quantization

We compare the proposed quantization scheme ('ProgQ') with a baseline uniform quantization with a constant range $S = 1$ ('UnifQ'), for both the Metropolis and the Laplacian weight matrices. Fig. 6.2 illustrates the average consensus performance corresponding to the absolute error $\|z_t - \mu\mathbf{1}\|_2$ versus the number of iterations for $n = [2, 4, 6]$ bits. In order to obtain statistically meaningful results we average the error over 200 random realizations of the network topology with random



(a) Metropolis weights

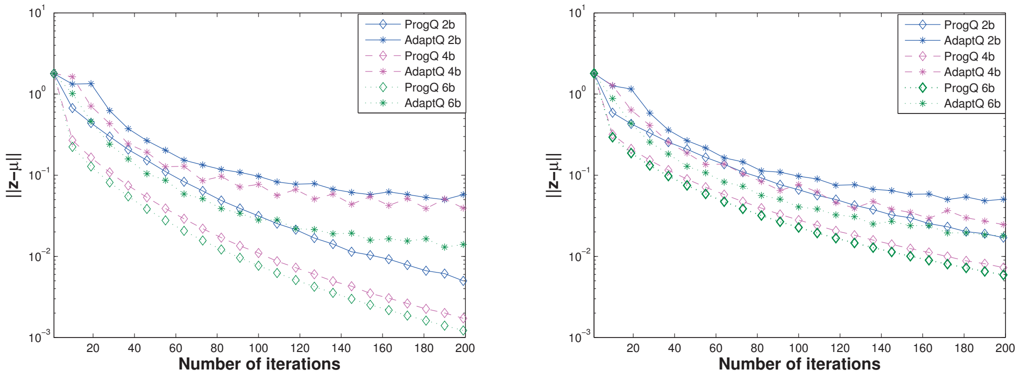
(b) Laplacian weights

Figure 6.2: Average consensus performance of the proposed quantization scheme (ProgQ) vs uniform quantizer with a constant range (UnifQ) for 2, 4 and 6 bits.

initial values. Observe that the performance of the proposed quantization scheme is very satisfactory even at a very low bit rate (2 bits). In particular, the error $\|z_t - \mu\mathbf{1}\|_2$ shows a decreasing behavior over the iterations, which means that the quantizer does not saturate. It rather follows the evolution of the average consensus algorithm in the noiseless case ('no quant.'). On the other hand, the performance of the uniform quantizer with a constant range saturates quickly even at high bit rate.

6.4.3 Comparison to existing quantization schemes for average consensus

We compare the proposed Progressive Quantizer ('ProgQ') with (a) the adaptive quantizer ('AdaptQ') [6], (b) the zoom in-zoom out uniform encoder ('ZoomQ') [7] and (c) the quantization scheme proposed in [8] ('Li et al.'). In particular, the scheme proposed in [6] is based on the Delta modulation with variable step-size. The step-size is adapted throughout the iterations based on the previously sent bits and a constant K. However, the scheme is quite sensitive to the value of K and the performance can deteriorate for non-carefully chosen values. In our experiments we choose $K = 1.2$ as defined in [6]. On the other hand, the differential encoding scheme proposed in [7] uses a uniform quantizer and the transmitted value is the quantized difference of the current value from the previous estimate, scaled by a factor f that is adapted over time. This factor is similar to the step-size of [6] and it grows or decreases depending on the difference between the new state z_{t+1} and the previously quantized state $\tilde{z}_t$. The decrease or the increase depends on the constants k_{in} and k_{out} respectively and the way that these constants have to be determined seems to be an open question. In our experiments we choose the parameters $k_{in} = 0.5, k_{out} = 2$ and the scaling factor $f_0 = 0.5$ as defined in [7]. Finally, the scheme proposed in [8] is also designed by adapting the scaling function of a difference encoder similar to our quantizer. The value that is quantized at each time step is the difference between the new state z_{t+1} and the previously quantized state $\tilde{z}_t$ while the scaling function is assumed to have an exponential decay over time. The



(a) Metropolis weights

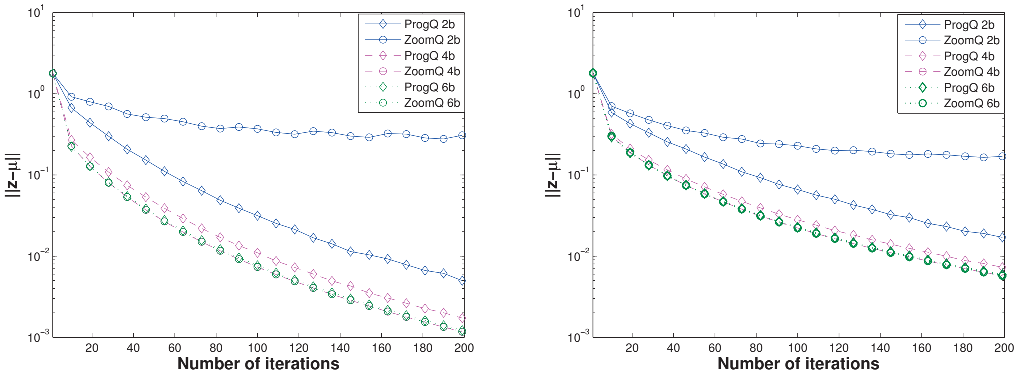
(b) Laplacian weights

Figure 6.3: Average consensus performance of the proposed quantization scheme (ProgQ) vs the adaptive quantizer (AdaptQ) [6] for 2, 4 and 6 bits.

parameters of the scaling function are defined in an interval form in such a way that the quantizer never saturates. However, one limitation of the scheme in [8] is that the algorithm is designed only for the Laplacian weights of the form $W = I - aL$, where the parameter a depends on the number of quantization bits. In order to directly compare the performance of the Progressive Quantizer with the quantization scheme of [8], we implement the version of the latter scheme that corresponds to a fixed number of quantization levels (Algorithm 1 in [8]). The parameters of the scaling function are representative values that belong to the proposed intervals. The edge weights of the matrix W are computed by Eqs. (6.18) and (6.19) for the Progressive Quantizer, the adaptive quantizer and the zoom-in zoom-out quantizer while for the quantization scheme of [8] they are computed as defined in the corresponding paper, as they depend on the selected parameters of the scaling function.

We use the same experimental setup as in the previous experiments. Figs 6.3, 6.4, 6.5 illustrate the simulation results and show performance comparisons for the quantizers with different bit rates. Notice first that our scheme outperforms the three above mentioned schemes in all the cases. AdaptQ appears to saturate especially for a small number of bits. The performance of ZoomQ seems to be quite good for 4 and 6 bits, but it suffers significantly at low bit rate. On the other hand, the performance of the last scheme (Li et al.) is quite poor even for 6 bits. This result is quite expected since the proposed intervals for the parameters of the scaling function are too conservative; they are computed such that no clipping appears during the iterative consensus algorithm. Moreover, we observe that both the selection of the weight matrix as indicated in [8] and its dependence on the bit rate penalize even more the convergence rate and the overall performance of the consensus algorithm.

Our scheme bears some resemblance with these three schemes in the sense that we also propose to adapt a scaling function. The scaling function has a very specific definition in our case where it represents the sensors' dynamic range. Moreover, we impose a consistent decay of the quantizer range size which is intuitively supported by the increasing correlation of the sensors' states throughout the iterations. The parameters α and γ that determine the quantization range in our



(a) Metropolis weights

(b) Laplacian weights

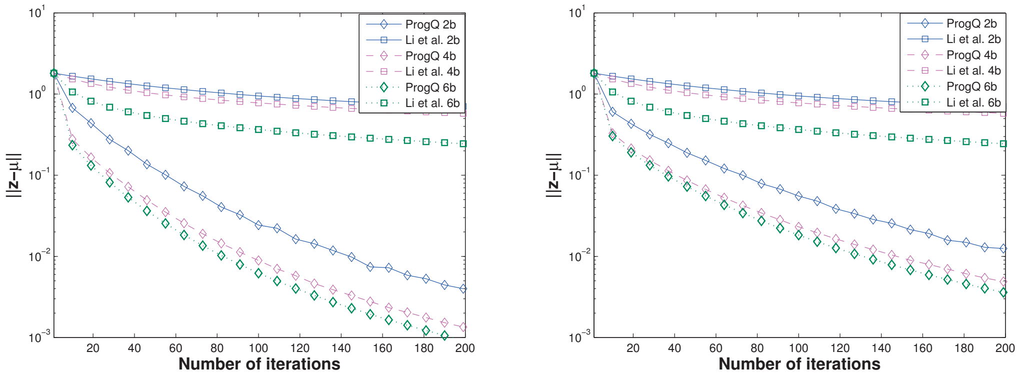
Figure 6.4: Average consensus performance of the proposed quantization scheme (ProgQ) vs the zoom-in, zoom-out uniform quantizer (ZoomQ) [7] for 2, 4 and 6 bits.

Progressive Quantizer have been carefully designed by taking into consideration both the available number of bits and the graph topology and they are automatically determined in closed-form from Eqs. (6.14) and (6.16). Finally, the performance comparison with the scheme proposed in [8] confirms our initial intuition that very conservative bounds do not necessarily improve the average consensus performance, and that average case analysis is more efficient in practical settings.

6.4.4 Convergence of the consensus algorithm

The assumption that the quantizer saturates complicates significantly the convergence analysis of the proposed algorithm. Our Progressive Quantizer may generate some clipping of the values computed by sensors. We have shown through extensive experimental results that this clipping does not significantly penalize the convergence of the consensus algorithm. It can even help in case of strong outliers. Moreover, we have observed that the number of clipping, if any, is small and decays to zero as the iteration of the consensus algorithm increases, as long as the parameter γ is computed according to Eq. (6.16). However, clipping results into some important non-linear effects that are difficult to analyze. In this subsection, we give some intuition about the convergence properties as well as the convergence speed of the Progressive Quantizer. The simulation results provided in the previous subsections verify that the proposed scheme leads the sensors to converge to the average of their initial values. We notice first that the quantization range reduces to zero as time elapses, leading to an accurate average consensus that is reached independently of the number of bits. The latter is verified experimentally in Fig. 6.6(a), where we observe that, as the range reduces to zero, the absolute error from the accurate consensus value μ becomes smaller.

Moreover, the design of the Progressive Quantizer promotes the decrease of the quantization noise variance over time. By properly decreasing the quantization range, we reduce the quantization noise, as long as the computed values to be quantized fall into that range. We finally relate the convergence of the algorithm to the quantization noise in the following proposition. Similar results have been shown in [154].



(a) Metropolis weights (b) Laplacian weights

Figure 6.5: Average consensus performance of the proposed quantization scheme (ProgQ) vs the quantization scheme proposed in [8] (Li et al.) for 2, 4 and 6 bits.

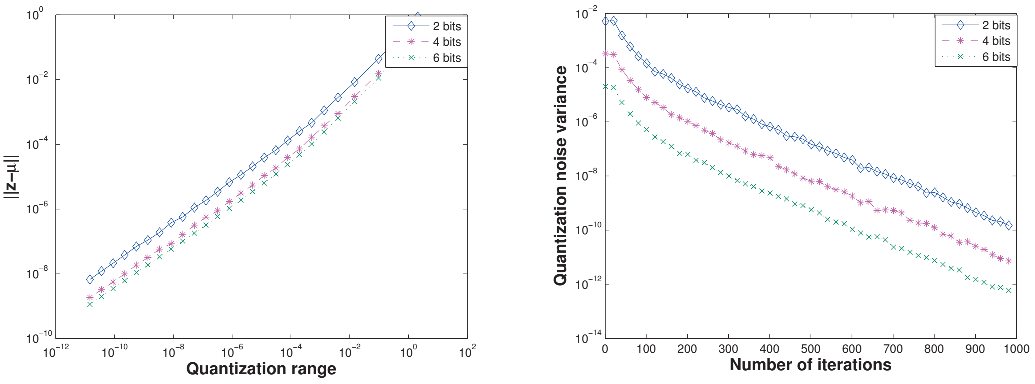
Proposition 5. *Let σ_t^2 be the sample variance of the quantization noise vector ϵ_t at iteration t. If $\sigma_t^2 \to 0$ as $t \to \infty$, the sensor nodes converge asymptotically to the true consensus value μ.*

The proof of Proposition 5 is given in Appendix D.4.

Fig. 6.6(b) verifies that our scheme leads to a quantization noise variance that converges to zero as time elapses with exactly the same behavior as the quantization range. These results are consistent with the ones shown in Fig. 6.2. They confirm that the average consensus performance is directly related to the decay of the quantization noise variance and that an accurate consensus is achieved for a variance that converges to zero. Finally, the decay of the noise variance and thus the convergence speed depend on the number of the quantization bits; more precisely the algorithm converges faster for a large number of bits, which is expected.

6.5 Related Work

A few works have been proposed to address the problem of quantization in consensus averaging algorithms. In particular, it was shown in [148] that if the quantization noise is modeled as white and additive with fixed variance then consensus cannot be achieved. The authors in [155] propose a probabilistic quantization scheme that is shown to reach a consensus almost surely to a random variable whose expected value is equal to the desired average. Unfortunately, the scheme performs poorly at low bit rate. Kashyap et al. [156] designed an average consensus algorithm with the additional constraint that the states of the agents are integers. Thus convergence is achieved to some integer approximation of the average of the initial states. Modifications of the classical consensus algorithm have been proposed in [151, 157], including a compensation error term that guarantees that the average is preserved at each iteration. A coding scheme based on predictive coding is proposed in [154] in order to exploit the temporal correlation among successive iterations. Convergence to the true average is shown to be possible under the condition that the quantization



(a) (b)

Figure 6.6: (a) Evolution of the average consensus performance for an exponentially reducing quantization range. (b) Evolution of the variance of the quantization noise over the iterations for 2, 4 and 6 bits.

noise variance vanishes over the iterations. A different approach for dealing with the quantization noise and additive white noise in general is proposed in [158] and [159] respectively. Both algorithms adapt the weight link sequence in order to guarantee convergence under certain conditions, at a cost of lower convergence rate. In general, all the above mentioned algorithms either maintain the average value in the network but cannot reach a consensus effectively, or converge to a random variable that is not always the target average value.

More recently, the authors in [6], [7] and [8] have proposed quantization strategies that maintain the average of the initial state and at the same time converge asymptotically to the true average value. The quantization scheme introduced in [6] adaptively adjusts the quantization step-size by learning from previous states, at the price of significant complexity and memory requirements. The quantization threshold in [7] is adapted online based on a zoom-in zoom-out strategy, while the set of quantization levels is maintained constant over the iterations. Although these last two solutions perform quite well at high bit rates, the convergence rate appears to be slow when the quantization is coarse. In addition, the stability of both quantization schemes depends on the choice of globally defined parameters that are not easy to determine a priori. Finally, the scheme proposed in [8] is the one that is closer to our work since it is based on the assumption that, as consensus is achieved asymptotically the prediction error of the sensors' states tends to zero. The scaling function is selected in such a way that it decays over time without causing the saturation of the quantizer. However, the proposed scheme leads to very conservative selection of the parameters of the scaling function that prevents reading significant gains in the average consensus performance. On the other hand, our system is able to achieve fast convergence to the true average value in realistic settings, even if it does not provide strict performance guarantees; it relies on average case analysis instead of conservative bounds.

Finally, we note that more attention has been recently given to the case of directed topologies. The authors in [160] propose a communication feedback-based distributed consensus protocol that limits the effect of quantization in directed time-varying topologies. The problem of parameter estimation over sensor networks in the presence of quantized data and directed communication links is studied in the recent work of [161]. A running average technique is used to guarantee that the centralized sample mean estimate is achieved both in the mean square and almost sure senses.

6.6 Conclusions

In this chapter, we have studied the effect of quantization on the distributed average consensus process that converges to a smooth graph signal, consisting of the average of the initial node values. In particular, we have proposed a novel quantization scheme for solving the average consensus problem that takes into account the diffusion of the information in the sensor network graph. Our scheme is based on the progressive reduction of the range of a uniform quantizer that is dictated by the evolution of the consensus process on the graph. We have shown that the range of the values throughout the consensus iterations depends on the graph topology, and in particular on the characteristics of the consensus matrix. The proposed scheme leads to progressive refinement of the information exchanged by the sensors while the process converges to a smooth and constant signal on the graph. Simulation results have shown the effectiveness of our scheme that outperforms other quantized consensus algorithms in terms of convergence rate and accuracy of the computed average. Finally, our quantization design is another significant proof of the usefulness of considering both the graph topology and the characteristics of the signal on the graph in solving distributed signal processing problems. Similar ideas could be exploited to design efficient quantization schemes for more complicated graph signal models such as the one considered in the previous chapters.

Chapter 7

Conclusions

7.1 Thesis achievements

In this thesis, we have addressed several problems related to the representation and processing of structured signals defined on weighted and undirected graphs. In particular, we have studied the problem of compression of 3D point cloud sequences by using useful properties of graph-based transform representations. Next, we have addressed the problem of sparsity on graphs, by proposing a novel framework for learning parametric spectral graph dictionaries that are able to sparsely represent graph signals. Finally, we have studied the distributed processing of graph signals by focusing on the limitation of the quantization noise that is an important issue in realistic settings. Below, we give more details about each of these contributions that study some of the open questions in the emerging field of signal processing on graphs.

The first contribution of this thesis is the use of graph signal representations in removing temporal redundancy between temporally correlated graph signals. In particular, we have proposed a novel graph-based compression framework for 3D point cloud sequences that is based on exploiting correlation between temporally consecutive point clouds. This problem has not been addressed earlier due mainly to the difficulties that arise from the lack of structure in point cloud sequences. Graphs provide a new and flexible framework for modeling this type of data. 3D models are represented by a sequence of weighted and undirected graphs and the geometry and the color of each model are considered as signals residing on the vertices of the corresponding graphs. Correspondences and eventually motions between nodes on two consecutive graphs are determined by matching new descriptors constructed on spectral features that are localized on the graph. Motion compensation is then used to perform geometry and color prediction, which is eventually used to differentially encode both the geometry and the color attributes. Our novel framework is a nice illustration of the application of graph signal processing for complex datasets, and a certainly promising path for removing temporal redundancy from high-dimensional data.

Next, we have introduced dictionary learning on graphs for designing meaningful and signal adapted representations for graph signals. Note that the existing graph signal representation designs are based on classical signal transforms such as Fourier and wavelets. We have proposed a new family of parametric graph dictionaries – namely, unions of polynomial matrix functions of the graph Laplacian – to sparsely represent signals on a given weighted graph, and an effective algorithm to learn the parameters of a dictionary from a set of training signals on the graph. Our trained atoms reveal underlying patterns in graph signals. The polynomial nature of the dictionaries permits us to abstract from the actual graph topology and learn dictionaries that can sparsely represent graph signals that live on different weighted graphs. This surely provides important benefits in terms of detecting and interpreting common structure across signals that live on different topologies. The proposed dictionary learning framework certainly represents a promising approach for efficient interpretation and mining of modern graph-structured data.

Then, we have studied the effect of quantization in distributed graph signal processing with polynomial dictionary operators. We have shown that the performance is dependent on the graph geometry and the dictionary structure. In addition, we have proposed an algorithm that learns graph dictionaries to sparsely approximate graph signals while staying robust to quantization noise in distributed processing. The proposed dictionary design results in a tradeoff between effective distributed signal representation in low bit rate communication settings and accuracy of the signal approximation in ideal communication settings. The work done in this chapter is definitely a first important step toward designing quantization-aware dictionaries for distributed signal processing.

Finally, we have attacked the problem of quantization in distributed graph signal processing from the perspective of the quantizer design. We have focused in particular on the distributed average consensus problem, which is a graph filtering process that is widely used in many distributed applications. We have proposed a novel quantization scheme for solving the average consensus problem when sensors exchange quantized state information. Our scheme is based on progressive reduction of the range of a uniform quantizer, which is dictated by the evolution of the averaging process on the graph. It leads to progressive refinement of the information exchanged by the sensors along with the convergence of the average consensus algorithm to a constant graph signal. Our quantizer represents a constructive simple solution with effective performance in realistic settings. Finally, the proposed design confirms the benefits of jointly considering the graph topology and the signal model on the graph for solving distributed signal processing tasks.

To summarize, we have studied in this thesis several important problems related to the emerging field of signal processing on graphs. We have provided novel solutions for processing and analyzing graph signals in both centralized and distributed settings. We believe that the research effort in this thesis gives some new insights about solutions to some of the challenges arising from the irregular graph domain, and makes one more step towards understanding the interplay between signals and graphs.

7.2 Future directions

Signal processing on graphs is a relatively new research field that is still in its infancy. Parts of this field are old as there exists a lot of research mainly in the machine learning and the computer science community on analyzing and understanding the graph structure. However, the concept of a signal on a graph is new and very interesting from a signal processing perspective. While this thesis brings several contributions in the theory of sparse graph signal representation and its applications, it provides answers to only some of the open questions that are related to the interdependence between the graph structure and the signal on the graph in proper data analysis. There are therefore many more exciting directions that graph signal processing research can pursue.

In Chapter 3, we introduced spectral features for finding correspondences between temporally correlated point clouds. The analysis of such features in terms of robustness to particular transformations such as translation, rotation and scaling would provide more insight into the usefulness of similar techniques in capturing transformation invariance in high-dimensional datasets. This would first require an in-depth understanding of the connection between the graph design and the underlying continuous manifold. Second, since the features are defined on the spectral domain, the explicit relation between the spectral characteristics of a graph signal and the signal model on an irregularly sampled manifold is definitely a very interesting and challenging problem.

The connection between sparsity and localization on graphs is very important and it also merits further investigation. In Chapter 4, we have introduced a framework for learning spectral graph dictionaries that leads to the sparse representation of graph signals, and we have shown that localization can be beneficial in effective signal modeling. However, when the signal is not very sparse in the vertex domain, very localized atoms tend to reduce the approximation performance at low sparsity with respect to atoms that have non-local support. Investigating further the tradeoff between the support of the atoms and the sparse representation is certainly an interesting problem. Moreover, additional work is required to apply the polynomial structure in other graph signal processing and data analysis tasks such as signal-based classification, clustering, community detection, or source localization, where we expect that the localization properties of the dictionary can be beneficial, as it is the case of wavelets in the Euclidean domain. The ability to detect commonalities across graphs may require the development of more sophisticated sparse coding algorithms. Intuitively, sparse coding algorithms should take into consideration the structure of the graph topology.

Concerning the analysis of the propagation of the quantization noise in distributed settings of Chapter 5, it would be interesting to study the sensitivity of the sparse representation algorithm with respect to the loss of messages as it happens in practical sensor networks. Moreover, the design of adaptive quantization algorithms, similar to that proposed in Chapter 6, and further developed in [162], are expected to yet improve the distributed graph signal processing performance. The redundancy of the dictionary, the connectivity of the graph, and the relative importance of each polynomial coefficient are some factors that can guide the design of the quantization scheme.

In all the distributed signal processing problems studied in this thesis, we assume that the sensor communication graph is the same with the underlying graph structure of the signal. It could be the case though that these two graphs are different. That would require the development of new distributed signal processing algorithms that are able to combine the communication graph and the graph domain of the signal efficiently. It definitely represents a very interesting and challenging open problem.

Finally, in all the problems studied in this thesis, we have assumed that the graph topology is known and it represents a useful source of information. However, in many applications the graph is not known a priori or it is not necessarily optimal. Learning the graph topology from the signals themselves is definitely a topic that is worth investigating. We have already started investigating these issues for a smooth signal model [163] and other priors. Graph learning can definitely help deepen our understanding of the interaction between graphs and signals on graphs, and permit to explain more complex behaviors in different types of networks. Such research efforts would generalize, extend, and improve the approaches presented in this thesis for applicability to a wider range of graph-structured signal representations and models.

To conclude, we definitely believe that graphs are among the most powerful and promising discrete tools for analyzing complex high-dimensional data sets. However, in order to fully exploit their power, we should further learn how to use them properly. The challenges are many. First, we need to understand the theoretical and empirical role of the graph structure, and define meaningful, possibly application-driven, criteria for constructing the graph. Second, the in-depth understanding of the true role of the graph structure in graph signals is a necessary step for designing more efficient graph-based signal processing and machine learning algorithms that can be used for analysis and inference tasks on complex high-dimensional data sets. Third, an important parameter that was not specifically considered in this thesis is the computational complexity of the graph-based algorithms. These should be designed in a scalable manner in order to handle large-size data (see e.g. [164] for a family of non-convex algorithms that can be used for learning on big graphs).

This only represents a short snapshot of the challenges that should be faced in the future. We believe that the algorithms and the applications presented in this thesis made significant steps towards addressing some of these important challenges.

Appendix A

Analysis of the sparse coding step

The success of the sparse coding step in our dictionary learning algorithm of Chapter 4 depends on the choice of the dictionary, i.e., the choice of the kernels and the structure of the graph. In particular, we recall that OMP selects atoms from the dictionary $\mathcal{D}$ having maximum inner product with the residual. The following theorem [125] gives an efficient recovery condition of OMP for the global dictionary $\mathcal{D}$:

Theorem 2. *([125]) Define the coherence of a dictionary $\mathcal{D}$ as*

$$\phi = \max_{d,d' \in \mathcal{D}, d \neq d'} \frac{| < d, d' > |}{\|d\| \|d'\|},$$

where d, d' are two dictionary atoms. OMP recovers every superposition of T_0 atoms from $\mathcal{D}$ whenever the following condition is satisfied:

$$T_0 < \frac{1}{2}(\phi^{-1} + 1).$$

The coherence ϕ measures the similarity between the dictionary atoms, and values of coherence close to one usually violate the above recovery condition. We examine next the dependence of the coherence of the graph structured polynomial dictionary on the kernels and the structure of the graph.

Theorem 3. *Let $\mathcal{D} = [\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_S]$ be a dictionary, defined on the graph $\mathcal{G}$, which is a concatenation of subdictionaries of the form $\mathcal{D}_s = \widehat{g}_s(\mathcal{L}) = \chi \widehat{g}_s(\Lambda) \chi^T$. The coherence ϕ of the dictionary can be upper-bounded as follows*

$$\phi \leq \max_{n \neq n', s, s'} \frac{\nu (\sum_{\ell=0}^{N-1} |\widehat{g}_s(\ell) \widehat{g_{s'}}(\ell)|^2)^{1/2} \|deg\|^2}{|\widehat{g}_s(\lambda_0)| |\widehat{g}'_s(\lambda_0)| \sqrt{deg_n} \sqrt{deg_{n'}}}, \tag{A.1}$$

where $deg = [deg_1, deg_2, ..., deg_N]$ is a vector containing the degrees of each node of the graph, and ν is the mutual coherence of the graph, defined as the maximum inner product between the basis *of Kronecker deltas on the graph and the basis of graph Laplacian eigenvectors:*

$$\nu = \max_{\ell=\{1,2,\ldots,N\},n=\{0,1,\ldots,N-1\}} | < \chi_\ell, \delta_n > |.$$

Proof: After substituting the polynomial dictionary atoms in the coherence expression, we obtain the following upper-bound

$$\begin{aligned}
\phi &= \max_{n\neq n',s,s'} \frac{| < \widehat{g_s}(\mathcal{L})\delta_n, \widehat{g_{s'}}(\mathcal{L})\delta_{n'} > |}{\|\widehat{g_s}(\mathcal{L})\delta_n\| \|\widehat{g_{s'}}(\mathcal{L})\delta_{n'}\|} \\
&= \max_{n\neq n',s,s'} \frac{| < \chi\widehat{g_s}(\Lambda)\chi^T\delta_n, \chi\widehat{g_{s'}}(\Lambda)\chi^T\delta_{n'} > |}{\|\widehat{g_s}(\mathcal{L})\delta_n\| \|\widehat{g_{s'}}(\mathcal{L})\delta_{n'}\|} \\
&\overset{(a)}{=} \max_{n\neq n',s,s'} \frac{|\delta_n^T\chi\widehat{g_s}(\Lambda)\widehat{g_{s'}}(\Lambda)\chi^T\delta_{n'}|}{\|\widehat{g_s}(\mathcal{L})\delta_n\| \|\widehat{g_{s'}}(\mathcal{L})\delta_{n'}\|} \\
&= \max_{n\neq n',s,s'} \frac{|\sum_{\ell=0}^{N-1} \widehat{g_s}(\ell)\widehat{g_{s'}}(\ell)\chi_\ell^T(n')\chi_\ell(n)|}{\|\widehat{g_s}(\mathcal{L})\delta_n\| \|\widehat{g_{s'}}(\mathcal{L})\delta_{n'}\|} \\
&\overset{(b)}{\leq} \max_{n\neq n',s,s'} \frac{(\sum_{\ell=0}^{N-1} |\widehat{g_s}(\ell)\widehat{g_{s'}}(\ell)|^2)^{1/2}(\sum_{\ell=0}^{N-1} |\chi_\ell^T(n')\chi_\ell(n)|^2)^{1/2}}{\|\widehat{g_s}(\mathcal{L})\delta_n\| \|\widehat{g_{s'}}(\mathcal{L})\delta_{n'}\|} \\
&\leq \max_{n\neq n',s,s'} \frac{\nu(\sum_{\ell=0}^{N-1} |\widehat{g_s}(\ell)\widehat{g_{s'}}(\ell)|^2)^{1/2}}{\|\widehat{g_s}(\mathcal{L})\delta_n\| \|\widehat{g_{s'}}(\mathcal{L})\delta_{n'}\|},
\end{aligned} \tag{A.2}$$

where (a) exploits the orthonormality of the eigenvectors of the graph Laplacian, and (b) relies on the application of the Cauchy-Schwarz inequality. The norm of each atom can be expressed as

$$\|\widehat{g_s}(\mathcal{L})\delta_n\|^2 = \|\widehat{\widehat{g_s}(\mathcal{L})\delta_n}\|^2 = \sum_{\ell=0}^{N-1} |\widehat{g_s}(\lambda_\ell)|^2|\chi_\ell^T(n)|^2, \tag{A.3}$$

where again we have used the orthonormality of the eigenvectors. Using the definition of the graph coherence and the fact that, for connected graphs, $\chi_0(n) = \frac{\sqrt{deg_n}}{\|\sqrt{deg}\|}$, where deg_n is the degree of node n, and deg is the vector containing the degree of all the nodes, we obtain

$$\|\widehat{g_s}(\mathcal{L})\delta_n\|^2 \geq \frac{|\widehat{g_s}(\lambda_0)|^2 deg_n}{\|deg\|^2},$$

which implies that

$$\phi \leq \max_{n\neq n',s,s'} \frac{\nu(\sum_{\ell=0}^{N-1} |\widehat{g_s}(\ell)\widehat{g_{s'}}(\ell)|^2)^{1/2}\|deg\|^2}{|\widehat{g_s}(\lambda_0)||\widehat{g'_s}(\lambda_0)|\sqrt{deg_n}\sqrt{deg_{n'}}}.$$

□

Eq. (A.1) indicates that the coherence of the dictionary depends on the structure of the graph (i.e., the coherence of the graph and the degree distribution of the nodes) and the spectral representation of the underlying kernels. In particular, in order to have a dictionary with incoherent atoms, we require that $\phi < 1$, which implies that

$$\nu \|deg\|^2 < \max_{n \neq n', s, s'} \frac{\sqrt{deg_n}\sqrt{deg_{n'}}|\widehat{g_s}(\lambda_0)||\widehat{g'_s}(\lambda_0)|}{(\sum_{\ell=0}^{N-1} |\widehat{g_s}(\ell)\widehat{g_{s'}}(\ell)|^2)^{1/2}}. \tag{A.4}$$

The latter indicates that the smaller the coherence of the graph, the higher the probability that we recover the support of the signals. We note that the above condition is based on a very conservative bound. Nevertheless, it still gives us an intuition about the type of graphs that lead to better results in the sparse recovery and sequentially in the dictionary update process.

Appendix B

Continuous kernel approximation on graphs from discrete samples

In the graph-based polynomial dictionary learning algorithm of Chapter 4, the problem of recovering the kernels $\{\widehat{g}_s(\cdot)\}_{s=1,2,\dots,S}$, for given sparse codes, can be cast as a continuous function approximation problem by polynomials from a set of discrete samples. In the following, we assume for simplicity that $S = 1$. However, the results can be generalized easily to $S \geq 1$. In particular, the fitting term of the optimization problem can be written as

$$\begin{aligned}\sum_{t=1}^{T} \frac{1}{M_t} \|Y_t - \mathcal{D}_t X_t\|_F &= \sum_{t=1}^{T} \frac{1}{M_t} \|\widehat{g}(\mathcal{L}_t) X_t - \widehat{p}(\mathcal{L}_t) X_t\|_F \\ &\leq \sum_{t=1}^{T} \frac{1}{M_t} N_t \|\widehat{g}(\mathcal{L}_t) - \widehat{p}(\mathcal{L}_t)\|_2 \|X_t\|_F \\ &= \max_{\lambda \in [0,\ 2]} |\widehat{g}(\lambda) - \widehat{p}(\lambda)| \sum_{t=1}^{T} \frac{N_t}{M_t} \|X_t\|_F, \end{aligned} \tag{B.1}$$

where the last two inequalities follow from basic matrix analysis [165]. We thus observe that the representation performance depends on how well the polynomials can approximate the kernels, and in particular on the minimax approximation. Since these kernels are defined in the spectral domain, we assume every kernel $\widehat{g}(\cdot)$ to be a continuous real-valued function defined on the interval $[\lambda_{min},\ \lambda_{max}]$, which in our case, due to the spectrum of the normalized Laplacian, is $[0,\ 2]$. According to the Weierstrass approximation theorem [166], for every $\epsilon > 0$ there exists a polynomial $\widehat{p^K}(\lambda)$ such that for all $\lambda \in [0,\ 2]$, we have $|\widehat{g}(\lambda) - \widehat{p^K}(\lambda)| < \epsilon$. The magnitude of the approximation error however depends on the interpolation points, which in our case correspond to the samples obtained from the different topologies.

Theorem 4 ([167]). *Suppose f is a real-valued function defined and continuous on the closed real interval $[a,\ b]$ and such that the derivative of f of order $n+1$ is continuous on $[a,\ b]$. Let $p_n \in \mathcal{P}_n$*

denote the Lagrange interpolation polynomial of f *with the Chebyshev interpolation points*

$$x_j = \frac{1}{2}(b-a)\cos\Big(\frac{(j+1/2)\pi}{n+1}\Big) + \frac{1}{2}(b+a), \quad j = 0, 1, ..., n \tag{B.2}$$

then

$$\|f - p_n\|_\infty \le \frac{(b-a)^{n+1}}{2^{2n+1}(n+1)!} \max_{x^* \in [a,\ b]} |f^{n+1}(x^*)|. \tag{B.3}$$

By combining Eqs. (B.1, B.2, B.3), we obtain the following result

$$\sum_{t=1}^{T} \frac{1}{M_t} \|Y_t - \mathcal{D}_t X_t\|_F \le \frac{1}{2^n(n+1)!} \max_{\lambda^* \in [0,\ 2]} |\widehat{g^{n+1}}(\lambda^*)| \sum_{t=1}^{T} \frac{N_t}{M_t} \|X_t\|_F, \tag{B.4}$$

which indicates that when the kernels are interpolated from polynomials defined in the Chebyshev nodes, the representation error is bounded and converges to zero as the degree of the polynomial increases. Since we do not have access to the samples of the kernel but only to the signal values, we aim through the dictionary learning process to estimate the values of the kernels $\widehat{g}(\cdot)$ in the Chebyshev nodes. The polynomial approximation of the function can then be computed though interpolation using Lagrange polynomials.

The kernel update step of the dictionary learning process estimates the values of the polynomials from the signal observations. We define as $\sigma(\Lambda) := \{\Lambda_1 \cup \Lambda_2, ..., \cup \Lambda_T\}$ the discrete set of eigenvalues contained in all the different topologies. We show next that the dictionary update step can be written as a weighted discrete least square problem between the underlying kernels $\widehat{g}$ and the approximation polynomial $\widehat{p}$ in the discrete set of points that are contained in $\sigma(\Lambda)$. In particular,

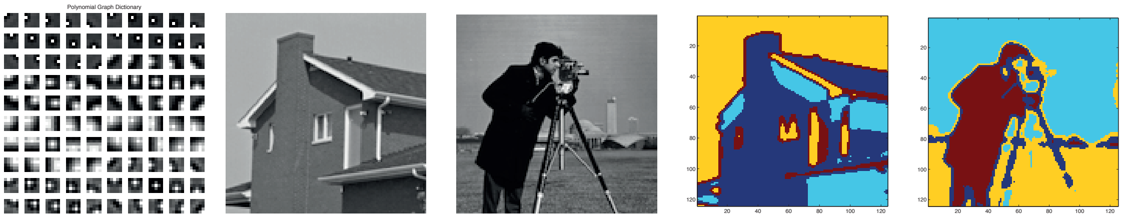
$$\begin{aligned}
\sum_{t=1}^{T} \frac{1}{M_t} \|Y_t - \mathcal{D}_t X_t\|_F &= \sum_{t=1}^{T} \frac{1}{M_t} \|\chi_t \widehat{g}(\Lambda_t) \chi_t^T X_t - \chi_t \widehat{p}(\Lambda_t) \chi_t^T X_t\|_F \\
&= \sum_{t=1}^{T} \frac{1}{M_t} \|\widehat{g}(\Lambda_t) \chi_t^T X_t - \widehat{p}(\Lambda_t) \chi_t^T X_t\|_F \\
&= \sum_{t=1}^{T} \frac{1}{M_t} \sum_{\ell_t=1}^{N_t} \sum_{m=1}^{M_t} (\widehat{g}(\lambda_{\ell_t}) \widehat{X_t^m}(\lambda_{\ell_t}) - \widehat{p}(\lambda_{\ell_t}) \widehat{X_t^m}(\lambda_{\ell_t}))^2 \\
&= \sum_{t=1}^{T} \sum_{\ell_t=1}^{N_t} (\widehat{g}(\lambda_{\ell_t}) - \widehat{p}(\lambda_{\ell_t}))^2 \sum_{m=1}^{M_t} \frac{1}{M_t} \widehat{X_t^m}(\lambda_{\ell_t})^2,
\end{aligned} \tag{B.5}$$

where $\widehat{X_t^m}(\lambda_{\ell_t})$ is the graph Fourier coefficient of the m^{th} signal of the t^{th} topology that corresponds to the eigenvector whose eigenvalue is λ_{ℓ_t}. Eq. (B.5) indicated that in order to have an accurate approximation of the kernel in the Chebyshev points we need two conditions: (i) the Chebyshev points should belong in $\sigma(\Lambda)$, and (ii) the weights that are given by the graph Fourier transform of the sparse codes in those points should be big.

Appendix C

Illustrative application of polynomial dictionaries: Image segmentation

As an illustrative application of the polynomial dictionary proposed in Chapter 4, we provide some results in image segmentation. We take the 128×128 house and 128×129 cameraman images and from each of them we extract overlapping block patches of size 5×5 pixels, covering all the pixels of the original image. Each patch is centered in one pixel and, for the sake of simplicity, we ignore the pixels on the boundary that do not have both horizontal and vertical neighbors. For each of the two images, the training signals are constructed as a collection of 15376 and 15625 such patches respectively. We fix the number of subdictionaries to $S = 4$, the polynomial degree to $K = 15$ and the sparsity level to $T_0 = 4$. The graph for each patch is the binary graph defined by connecting each pixel to its horizontal and vertical neighbors. For each of the images, we apply our polynomial dictionary learning algorithm, training a dictionary of dimensionality 25×100. Since the number of training signals is large, we apply ADMM to solve the quadratic program in the learning phase. In order to extract the features for the segmentation of the image, we compute the inner product of each patch with the atoms of the learned dictionary. If y_j is the patch corresponding to pixel j, then $\mathcal{D}_s^T y_j = \sum_{\ell=0}^{N-1} \widehat{y_j}(\lambda_\ell)\widehat{g_s}(\lambda_\ell)\chi_\ell$, which implies that we filter each patch with all four filters $\{\widehat{g_s}(\cdot)\}_{s=1,2,3,4}$ in order to modify its frequency characteristics. For each filtered version of the



(a) (b) (c) (d) (e)

Figure C.1: Learned atoms (a) and segmentation results (d,e) obtained using the polynomial dictionary on the house (b) and cameraman (c) images.

patch, we compute the mean and the variance. We define as feature for each patch a vector in $\mathbb{R}^{2S}$ that contains the mean and variance of each filtered versions. The features, and consequently the nodes, are then clustered in $S = 4$ clusters, using K-means.

The obtained segmentations and some of the learned atoms are shown in Fig. C.1. We observe that the segmentation results in both images are quite promising as the edges of the images are preserved most of the time. This is mainly due to the localization of the atoms. Further work and more extensive studies are required to deploy the proposed algorithm in image segmentation applications.

Appendix D

Supplementary material for Chapter 6

D.1 Proof of Proposition 2

Eq. (6.4) implies that

$$z_{t+1} - \tilde{z}_t = W^t(W - I)z_0 + (W - 2I)\epsilon_t + (W - I)\sum_{s=0}^{t-1} W^s(W - I)\epsilon_{t-s-1}. \tag{D.1}$$

Let us define

$$A_1 := W^t(W - I)z_0$$

and

$$A_2 := (W - I)\sum_{s=0}^{t-1} W^s(W - I)\epsilon_{t-s-1} + (W - 2I)\epsilon_t.$$

Then, we observe that

$$E[\|z_{t+1} - \tilde{z}_t\|^2] = E[\|A_1\|^2] + E[A_1^\top A_2] + E[A_2^\top A_1] + E[\|A_2\|^2]. \tag{D.2}$$

In the formula above, $\|A_1\|^2$ is a deterministic quantity depending on the (fixed) network topology and the initial states of the sensors. We assume that the quantization noise samples $\epsilon_t(i)$ in (6.2) are (spatially and temporally) independent random variables that are uniformly distributed with zero mean and variance $\Delta_t^2/12$, where Δ_t is the quantization step-size at step t. The latter assumption is widely used in the literature [154, 6, 151] for modeling the quantization noise and is true under the conditions specified in [168]. Due to this independence assumption, the two cross terms $E[A_1^\top A_2]$ and $E[A_2^\top A_1]$ become zero, and (D.2) simplifies to

$$E[\|z_{t+1} - \tilde{z}_t\|^2] = \|A_1\|^2 + E[\|A_2\|^2]. \tag{D.3}$$

In the sequel, we work out each individual term separately. First, note that

$$\begin{aligned}
\|A_1\|^2 &= \|W^t(W-I)z_0\|^2 \\
&\overset{(a)}{=} \|W^t(W-I)z_0 - \frac{\mathbf{1}\mathbf{1}^T}{N}(W-I)z_0\|^2 \\
&\overset{(b)}{\leq} \left\| W - \frac{\mathbf{1}\mathbf{1}^T}{N} \right\|_2^{2t} \|W-I\|^2 \|z_0\|^2,
\end{aligned} \tag{D.4}$$

where (a) follows from the fact that W is doubly stochastic and (b) holds because $\left\|W^t - \frac{\mathbf{1}\mathbf{1}^T}{N}\right\|_2 = \lambda_2^t$, where $\lambda_2 := \rho(W - \frac{\mathbf{1}\mathbf{1}^T}{N})$. Moreover,

$$\begin{aligned}
E[\|A_2\|^2] &= E[\|(W-I)\sum_{s=0}^{t-1} W^s(W-I)\epsilon_{t-s-1} + (W-2I)\epsilon_t\|^2] \\
&\leq \|W-I\|^2 \sum_{s=0}^{t-1} \|W^s(W-I)\|^2 E[\|\epsilon_{t-s-1}\|^2] + \|W-2I\|^2 E[\|\epsilon_t\|^2],
\end{aligned}$$

where we have again exploited the fact that the quantization noise variables are independent and zero mean. We compute the expectation of the quantization error norms by exploiting the consistently reduced range of the Progressive Quantizer. In particular, the expectation of the error norm at each time step t can be expressed as $E[\|\epsilon_t\|^2] \leq NE[\epsilon_t(i)^2] \leq N\Delta_t^2/12$ [168], where Δ_t is the quantization step-size at step t. Hence, the expectation of the second term is bounded as follows

$$E[\|A_2\|^2] \leq \|W-I\|^2 \sum_{s=0}^{t-1} \|W^s(W-I)\|^2 N \frac{\Delta_{t-s-1}^2}{12} + \|W-2I\|^2 N \frac{\Delta_t^2}{12}. \tag{D.5}$$

We finally derive an upper-bound for $E[\|z_{t+1} - \tilde{z}_t\|^2]$ by combining both (D.4) and (D.5) and taking into consideration that $\Delta_t^2 = \left(\frac{S_t}{2^n}\right)^2$ (see also Eq. (6.6)). Altogether, we obtain

$$\begin{aligned}
E[\|z_{t+1} - \tilde{z}_t\|^2] \leq \|z_0\|^2 \lambda_2^{2t} \|W-I\|^2 + \|W-I\|^2 \sum_{s=0}^{t-1} \|W^s(W-I)\|^2 N \frac{S_{t-s-1}^2}{2^{2n} \cdot 12} \\
+ \|W-2I\|^2 N \frac{S_t^2}{2^{2n} \cdot 12}.
\end{aligned}$$

Since the eigenvalues of the matrix W lie in the interval $[-1, 1]$, the Proposition 2 follows from the fact that $\|W-I\| = 1 - \lambda_{\min}$ and $\|W-2I\| = 2 - \lambda_{\min}$, where $\lambda_{\min}$ is the smallest algebraically eigenvalue of W. □

D.2 Proof of Proposition 3

Firstly, we define a new sequence $P(t)$ to be the upper-bound of $e^{-2\beta_t}$.

Definition 1. *The sequence $P(t)$ is defined as:*

$$\begin{aligned}
P(0) &= \frac{(z_0^{(\max)} - z_0^{(\min)})^2}{4} \\
P(1) &= \|z_0\|_\infty^2 (1-\lambda_{\min})^2 + \frac{(2-\lambda_{\min})^2}{2^{2n}\cdot 3} P(0) \\
P(t) &= \|z_0\|_\infty^2 (1-\lambda_{\min})^2 \lambda_2^{2(t-1)} + (1-\lambda_{\min})^4 \sum_{s=0}^{t-2} \lambda_2^{2s} \frac{P(t-2-s)}{2^{2n}\cdot 3} + (2-\lambda_{\min})^2 \frac{P(t-1)}{2^{2n}\cdot 3}, \quad t \geq 2.
\end{aligned}$$

Moreover,

$$\begin{aligned}
P(0) &= e^{-2\beta_0} \\
P(1) &= e^{-2\beta_1} \\
P(t) &\geq e^{-2\beta_t}, \quad t \geq 2.
\end{aligned}$$

We can then write the sequence in the following form

$$P(t+1) = (c+\gamma)P(t) + (b - c\gamma)P(t-1), \tag{D.6}$$

where $c = \frac{(2-\lambda_{\min})^2}{2^{2n}\cdot 3}$, $b = \frac{(1-\lambda_{\min})^4}{2^{2n}\cdot 3}$ and $\gamma = \lambda_2^2$ are positive constants. Expressing the above sequence in a matrix form we obtain

$$\begin{bmatrix} P(t+1) \\ P(t) \end{bmatrix} = \underbrace{\begin{bmatrix} c+\gamma & b-c\gamma \\ 1 & 0 \end{bmatrix}}_{A} \begin{bmatrix} P(t) \\ P(t-1) \end{bmatrix}. \tag{D.7}$$

The eigenvalues of the matrix A are $\frac{c+\gamma\pm\sqrt{(c-\gamma)^2+4b}}{2}$ and the dynamical system (D.7) converges to zero if its eigenvalues are strictly smaller than 1 [169]. Since c, b and γ are three positive constants, it is enough to find conditions that guarantee that $\frac{c+\gamma+\sqrt{(c-\gamma)^2+4b}}{2} < 1$. After substitution of the values of the constants c, b and γ, it leads to the following inequality

$$\frac{(2-\lambda_{\min})^2}{3\cdot 2^{2n}} + \lambda_2^2 + \sqrt{(\frac{(2-\lambda_{\min})^2}{3\cdot 2^{2n}} - \lambda_2^2)^2 + 4\frac{(1-\lambda_{\min})^4}{3\cdot 2^{2n}}} < 2.$$

Equivalently, we can write

$$\frac{4(1-\lambda_{\min})^4 - 4(\lambda_2^2 - 1)(2-\lambda_{\min})^2}{3\cdot 2^{2n}} < (2\lambda_2^2 - 2)^2 - 2\lambda_2^2(2\lambda_2^2 - 2), \tag{D.8}$$

which indicates that $P(t)$ converges to zero if the following condition

$$\frac{(1-\lambda_{\min})^4}{(1-\lambda_2^2)} + (2-\lambda_{\min})^2 < 3\cdot 2^{2n} \tag{D.9}$$

is satisfied. Since the sequence $P(t)$ is an upper-bound of the positive sequence $e^{-2\beta_t}$, if the condition (D.9) is satisfied, $e^{-2\beta_t}$ also converges to zero. $\square$

D.3 Proof of Proposition 4

Recall from (D.4) that $\|W^t(W-I)\|^2 \leq \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{2t}\|W-I\|^2 = \lambda_2^{2t}(1-\lambda_{\min})^2$. Moreover, under the condition specified in Proposition 3, we have that $\lim_{t\to\infty} e^{-\beta_t} = 0$. Given $\epsilon > 0$, there exists a time instant $k > 0$ such that $\forall t \geq k$, $e^{-\beta_t} < \epsilon$. Then,

$$\begin{aligned}\sum_{s=0}^{t-1} \|W^s(W-I)\|^2 e^{-2\beta_{t-s-1}} &\leq \sum_{s=0}^{t-1} \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{2s}\|W-I\|^2 e^{-2\beta_{t-s-1}} \\ &= \|W-I\|^2 \sum_{s=0}^{t-1} \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{2s} e^{-2\beta_{t-s-1}}.\end{aligned} \tag{D.10}$$

Moreover, since the eigenvalues of the matrix W lie in the interval $[-1, 1]$ we have that $\|W - I\| = 1 - \lambda_{\min} \leq 2$. Hence, using these observations and setting $l = t - s - 1$, the above inequality can be written as

$$\begin{aligned}&\sum_{s=0}^{t-1} \|W^s(W-I)\|^2 e^{-2\beta_{t-s-1}} \leq 4\sum_{l=0}^{t-1} \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{2(t-l-1)} e^{-2\beta_l} \\ &< 4\sum_{l=0}^{k} \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{2(t-l-1)} e^{-2\beta_l} + 4\sum_{l=k+1}^{t-1} \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{2(t-l-1)} \epsilon^2.\end{aligned} \tag{D.11}$$

We notice that, since $\|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\| < 1$, $e^{-\beta_t} \leq \delta, \forall t$ and k is finite, the first term of the right-hand side of the inequality converges to 0 for $t \to \infty$. Finally, we can write

$$\begin{aligned}\lim_{t\to\infty} \sum_{s=0}^{t-1} \|W^s(W-I)\|^2 e^{-2\beta_{t-s-1}} &< \lim_{t\to\infty} 4 \sum_{l=k+1}^{t-1} \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{2(t-l-1)} \epsilon^2 \\ &< \lim_{t\to\infty} 4C\epsilon^2 \to 0, \text{ for } \epsilon \to 0,\end{aligned} \tag{D.12}$$

where we have used the fact that $\lim_{t\to\infty} \sum_{l=k+1}^{t-1} \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{2(t-l-1)}$ converges to a constant $C < \infty$. $\square$

D.4 Proof of Proposition 5

The proof of Proposition 5 is similar to the one of Proposition 4. Assume that the sample variance σ_t^2 of the quantization noise components converges to zero after some specific iterations. In particular, given $\delta > 0$, there exists a time instant $k > 0$ such that $\forall t \geq k$, $\sigma_t^2 < \delta$. Using Eq.(6.4), the

Euclidean deviation of the node states can be expressed as

$$\begin{aligned}
\|z_{t+1} - \frac{\mathbf{1}\mathbf{1}^T}{N} z_0\| &= \|W^{t+1} z_0 + \sum_{s=0}^{t} W^s (W-I)\epsilon_{t-s} - \frac{\mathbf{1}\mathbf{1}^T}{N} z_0\| \\
&\leq \|W^{t+1} z_0 - \frac{\mathbf{1}\mathbf{1}^T}{N} z_0\| + \sum_{s=0}^{t} \|W^s (W-I)\epsilon_{t-s}\| \\
&\leq \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{t+1} \|z_0 - \frac{\mathbf{1}\mathbf{1}^T}{N} z_0\| \\
&+ \sum_{s=0}^{t} \|(W - \frac{\mathbf{1}\mathbf{1}^T}{N})^s (W-I)(\epsilon_{t-s} - \frac{\mathbf{1}\mathbf{1}^T}{N}\epsilon_{t-s})\| \\
&\leq \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{t+1} \|z_0 - \frac{\mathbf{1}\mathbf{1}^T}{N} z_0\| + \\
&+ \sum_{s=0}^{t} \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^s \|W-I\| \|\epsilon_{t-s} - \frac{\mathbf{1}\mathbf{1}^T}{N}\epsilon_{t-s}\|,
\end{aligned}$$

where we have used again the properties of the matrix W as defined in Eq. (6.1). Notice that $\|\epsilon_{t-s} - \frac{\mathbf{1}\mathbf{1}^T}{N}\epsilon_{t-s}\|$ can be written in terms of the sample variance σ^2_{t-s} of the quantization noise at iteration $t-s$ such as $\|\epsilon_{t-s} - \frac{\mathbf{1}\mathbf{1}^T}{N}\epsilon_{t-s}\| = \sqrt{N\sigma^2_{t-s}}$. Moreover, using the fact that $\|W - I\| = 1 - \lambda_{\min} \leq 2$ and setting $l = t - s$, the above inequality can be written as

$$\begin{aligned}
\|z_{t+1} - \frac{\mathbf{1}\mathbf{1}^T}{N} z_0\| &\leq \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{t+1} \|z_0 - \frac{\mathbf{1}\mathbf{1}^T}{N} z_0\| + 2\sqrt{N} \sum_{l=0}^{t} \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{t-l} \sqrt{\sigma_l^2} \\
&= \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{t+1} \|z_0 - \frac{\mathbf{1}\mathbf{1}^T}{N} z_0\| + 2\sqrt{N} (\sum_{l=0}^{k} \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{t-l} \sqrt{\sigma_l^2} \\
&+ \sum_{l=k+1}^{t} \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{t-l} \sqrt{\sigma_l^2}).
\end{aligned}$$

We notice that since $\|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\| < 1$ and k is finite, the first two terms of the right-hand side of the inequality converge to 0 for $t \to \infty$. Furthermore, by assumption we have that $\sigma_t^2 < \delta, \forall t > k$, which implies that for $t \to \infty$, we have

$$\lim_{t\to\infty} \|z_{t+1} - \frac{\mathbf{1}\mathbf{1}^T}{N} z_0\| < 2\sqrt{N}\delta \lim_{t\to\infty} \sum_{l=k+1}^{t} \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{t-l} < 2\sqrt{N}\delta C,$$

where again we have used the fact that $\lim_{t\to\infty} \sum_{l=k+1}^{t} \|W - \frac{\mathbf{1}\mathbf{1}^T}{N}\|^{t-l}$ converges to a constant $C < \infty$. Finally, as the variance converges to 0, the sensors reach a consensus on the average $\mu = \frac{\mathbf{1}\mathbf{1}^T}{N} z_0$ i.e.,

$$\lim_{t\to\infty} \|z_{t+1} - \frac{\mathbf{1}\mathbf{1}^T}{N} z_0\| \to 0 \text{ for } \delta \to 0.$$

□

Bibliography

[1] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *Proc. IEEE Int. Conf. on Robotics and Automation*, May 2012, pp. 778–785.

[2] C. Zhang, D. Florêncio, and C. Loop, "Point cloud attribute compression with graph transform," in *Proc. IEEE Int. Conf. on Image Process.*, Sep. 2014, pp. 2066 – 2070.

[3] D. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Appl. Comput. Harmon. Anal.*, vol. 30, no. 2, pp. 129–150, Mar. 2010.

[4] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006.

[5] X. Zhang, X. Dong, and P. Frossard, "Learning of structured graph dictionaries," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process.*, Mar. 2012, pp. 3373 – 3376.

[6] J. Fang and H. Li, "An adaptive quantization scheme for distributed consensus," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process.*, Apr. 2009, pp. 2777–2780.

[7] R. Carli, F. Bullo, and S. Zampieri, "Quantized average consensus via dynamic coding/decoding schemes," *Int. Journal of Robust and Nonlinear Control*, vol. 20, no. 2, pp. 156–175, Jan. 2010.

[8] T. Li, M. Fu, L. Xie, and J. Zhang, "Distributed consensus with limited communication data rate," *IEEE Trans. on Automatic Control*, vol. 56, no. 2, pp. 279–292, Feb. 2011.

[9] D. I Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, May 2013.

[10] A. Sandryhaila and J.M.F. Moura, "Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 80–90, Sept. 2014.

[11] F. Chung, *Spectral Graph Theory*, American Mathematical Society, 1997.

[12] T. Bıyıkoğlu, J. Leydold, and P. F. Stadler, *Laplacian eigenvectors of graphs: Perron-Frobenius and Faber-Krahn type theorems*, Lecture Notes in Mathematics, Springer Verlag, 2007.

[13] F. Chung, “Laplacians and the cheeger inequality for directed graphs,” *Annals of Combinatorics*, vol. 9, pp. 1–19, Apr. 2005.

[14] C. Zhang, D. Florêncio, and P. Chou, “Graph signal processing - a probabilistic framework,” Tech. Rep., Apr. 2015.

[15] X. Zhu and M. Rabbat, “Approximating signals supported on graphs,” in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process.*, Mar. 2012, pp. 3921–3924.

[16] D. I Shuman, B. Ricaud, and P. Vandergheynst, “Vertex-frequency analysis on graphs,” *accepted for publication in Appl. Comput. Harmon. Anal.*, Feb. 2015.

[17] D. I Shuman, B. Ricaud, and P. Vandergheynst, “A windowed graph Fourier transform,” in *Proc. IEEE Stat. Signal Process. Wkshp.*, Aug. 2012.

[18] M. Gavish, B. Nadler, and R. R. Coifman, “Multiscale wavelets on trees, graphs and high dimensional data: Theory and applications to semi supervised learning,” in *Proc. Int. Conf. on Machine Learning*, Jun. 2010.

[19] I. Ram, M. Elad, and I. Cohen, “Generalized tree-based wavelet transform,” *IEEE Trans. Signal Process.*, vol. 59, no. 9, pp. 4199–4209, Sep. 2011.

[20] I. Ram, M. Elad, and I. Cohen, “Redundant wavelets on graphs and high dimensional data clouds,” *IEEE Signal Process. Lett.*, vol. 19, no. 5, pp. 291–294, May 2012.

[21] M. Crovella and E. Kolaczyk, “Graph wavelets for spatial traffic analysis,” in *Proc. of INFOCOM*, Apr. 2003, vol. 3, pp. 1848–1857.

[22] G. Shen and A. Ortega, “Transform-based distributed data gathering,” *IEEE Trans. Signal Process.*, vol. 58, no. 7, pp. 3802–3815, Jul. 2010.

[23] R. R. Coifman and M. Maggioni, “Diffusion wavelets,” *Appl. Comput. Harmon. Anal.*, vol. 21, pp. 53–94, Mar. 2006.

[24] S. K. Narang and A. Ortega, “Perfect reconstruction two-channel wavelet filter banks for graph structured data,” *IEEE Trans. Signal Process.*, vol. 60, no. 6, pp. 2786–2799, Jun. 2012.

[25] D. I Shuman, C. Wiesmeyr, N. Holighaus, and P. Vandergheynst, “Spectrum-adapted tight graph wavelet and vertex-frequency frames,” *IEEE Trans. Signal Process.*, vol. 63, no. 16, pp. 4223–4235, Aug. 2015.

[26] N. Leonardi and D. Van De Ville, “Tight wavelet frames on multislice graphs,” *IEEE Trans. Signal Process.*, vol. 61, no. 13, pp. 3357–3367, Jul. 2013.

[27] D. I Shuman, M. J. Faraji, and P. Vandergheynst, "A multiscale pyramid transform for graph signals," *Accepted for publication in IEEE Trans. Signal Process.*, Aug. 2015.

[28] J. C. Bremer, R. R. Coifman, M. Maggioni, and A. D. Szlam, "Diffusion wavelet packets," *Appl. Comput. Harmon. Anal.*, vol. 21, no. 1, pp. 95–112, Jun. 2006.

[29] R. M. Rustamov and L. Guibas, "Wavelets on graphs via deep learning," in *Adv. Neural Inf. Process. Syst.*, Dec. 2013.

[30] D. Zhou and B. Schölkopf, "Regularization on discrete spaces," in *Pattern Recognition*, 2005, pp. 361–368.

[31] D. Zhou, O. Bousquet, T .N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Adv. Neural Inf. Process. Syst.*, Dec. 2004, vol. 16, pp. 321–328.

[32] M. Belkin, I. Matveeva, and P. Niyogi, "Regularization and semi- supervised learning on large graphs," *Learning Theory, Lecture Notes in Computer Science*, pp. 624–638, 2004.

[33] D. Zhu and B. Schölkopf, "A regularization framework for learning from graph data," in *Proc. Int. Conf. on Machine Learning, Wrkshp. on Statistical Relational Learning*, Jun. 2004, pp. 132–137.

[34] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proc. of Inter. Conf. on Machine Learning*, Jul. 2003, vol. 13, pp. 912–919.

[35] A. J. Smola and R. Kondor, "Kernels and regularization on graphs," in *Proc. of the Annual Conf. on Computational Learning Theory, Lecture Notes in Computer Science*, 2003, pp. 144–158.

[36] A. Gadde, A. Anis, and A. Ortega, "Active semi-supervised learning using sampling theory for graph signals," in *ACM SIGKDD Intern. Conf. on Knowledge Discovery and Data Mining*, Aug. 2014, pp. 492–501.

[37] M. Zheng, J. Bu, C. Chen, C. Wang, L. Zhang, G. Qiu, and D. Cai, "Graph regularized sparse coding for image representation," *IEEE Trans. Image Process.*, vol. 20, no. 5, pp. 1327–1336, May 2011.

[38] V. Kalofolias, X. Bresson, M. M. Bronstein, and P. Vandergheynst, "Matrix completion on graphs," *ArXiv, available at http://arxiv.org/abs/1408.1717*, 2014.

[39] S. Segarra, A. G. Marques, G. Leus, and A. Ribeiro, "Reconstruction of graph signals through percolation from seeding nodes," *Submitted to IEEE Trans. Signal Process.*, 2015.

[40] D. I Shuman, M. J. Faraji, and P. Vandergheynst, "Semi-supervised learning with spectral graph wavelets," in *Proc. of the Int. Conf. on Sampling Theory and Applications*, May 2011.

[41] V. N. Ekambaram, G. C. Fanti, B. Ayazifar, and K. Ramchandran, "Wavelet-regularized graph semi-supervised learning," in *IEEE Global Conf. on Sign. and Infor. Process.*, Dec. 2013, pp. 423–426.

[42] C. Hu, L. Cheng, J. Sepulcre, G. El Fakhri, Y. M. Lu, and Q. Li, "Matched signal detection on graphs: Theory and application to brain network classification," in *Intern. Conf., Inform. Process. in Medical Imag.*, Jun. 2013, pp. 1–12.

[43] S. K. Narang, A. Gadde, E. Sanou, and A. Ortega, "Localized iterative methods for interpolation in graph structured data," in *IEEE Global Conf. on Sign. and Inform. Process.*, Dec. 2013, pp. 491–494.

[44] S. Chen, F. Cerda, P. Rizzo, J. Bielak, J. H. Garrett Jr., and J. Kovacevic, "Semi-supervised multiresolution classification using adaptive graph filtering with application to indirect bridge structural health monitoring," *IEEE Trans. Signal Process.*, vol. 62, no. 11, pp. 2879–2893, Jun. 2014.

[45] N. Tremblay and P. Borgnat, "Graph wavelets for multiscale community mining," *IEEE Trans. Signal Process.*, vol. 62, no. 20, pp. 5227–5239, Aug. 2014.

[46] X. Dong, A. Ortega, P. Frossard, and P. Vandergheynst, "Inference of mobility patterns via spectral graph wavelets," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process.*, May 2013, pp. 3118 – 3122.

[47] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *System and Control Letters*, vol. 53, pp. 65–78, Feb. 2004.

[48] V. D. Blondel, J. M. Hendrickx, A. Olshevsky, and J. N. Tsitsiklis, "Convergence in multiagent coordination, consensus and flocking," in *IEEE Int. Conf. Decision Contr., Eur. Contr. Conf.*, Dec. 2005, pp. 2996–3000.

[49] A. G. Dimakis, S. Kar, J. Moura, M. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *IEEE Trans. Signal Process.*, vol. 98, no. 11, pp. 1847–1864, Nov. 2010.

[50] E. Kokiopoulou and P. Frossard, "Distributed classification of multiple observation sets by consensus," *IEEE Trans. Signal Process.*, vol. 59, no. 1, pp. 104–114, Jan. 2011.

[51] K. Flouri, B. Beferull-Lozano, and P. Tsakalides, "Distributed consensus algorithms for SVM training in wireless sensor networks," in *Proc. European Signal Process. Conf.*, Aug. 2008, pp. 1 – 5.

[52] A. Shmidt and J. Moura, "A distributed sensor fusion algorithm for the inversion of sparse fields," in *Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers*, Nov. 2009.

[53] L. Li, A. Scaglione, and J. Manton, "Distributed principal subspace estimation in wireless sensor networks," *IEEE Journal of Selected Topics in Signal Process.*, vol. 5, no. 4, pp. 725 – 738, Aug. 2011.

[54] D. Ustebay, R. Castro, and M. Rabbat, "Efficient decentralized approximation via selective gossip," *IEEE Journal of Selected Topics in Signal Process.*, vol. 5, no. 4, pp. 805–816, Aug. 2011.

[55] D. I Shuman, P. Vandergheynst, and P. Frossard, “Chebyshev polynomial approximation for distributed signal processing,” in *Proc. IEEE Int. Conf. Distr. Comput. Sensor Sys.*, Jun. 2011, pp. 1–8.

[56] D. I Shuman, P. Vandergheynst, and P. Frossard, “Distributed signal processing via Chebyshev polynomial approximation,” *ArXiv*, Nov. 2011.

[57] X. Wang, M. Wang, and Y. Gu, “A distributed tracking algorithm for reconstruction of graph signals,” *IEEE Journal of Selected Topics in Signal Process.*, vol. 9, no. 4, pp. 728–740, Jun. 2015.

[58] S. Chen, A. Sandryhaila, and J. Kovacevic, “Distributed algorithm for graph signal inpainting,” in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process.*, Apr. 2015, pp. 3731–3735.

[59] E. Kokiopoulou and P. Frossard, “Polynomial filtering for fast convergence in distributed consensus,” *IEEE Trans. Signal Process.*, vol. 57, no. 1, pp. 342–354, Jan. 2009.

[60] A. Sandryhaila, S. Kar, and J. M. F. Moura, “Finite-time distributed consensus through graph filters,” in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process.*, May 2014, pp. 1080–1084.

[61] S. Segarra, A. G. Marques, and A. Ribeiro, “Distributed linear network operators using graph filters,” *ArXiv, available at http://arxiv.org/abs/1510.03947*, 2015.

[62] M. Hein, J.-Y. Audibert, and U. von Luxburg, “From graphs to manifolds - weak and strong pointwise consistency of graph Laplacians,” in *Proc. of the Annual Conf. on Computat. Learn. Theory*, Jun. 2005, pp. 470–485.

[63] A. Singer, “From graph to manifold Laplacian: The convergence rate,” *Applied and Comput. Harm. Anal.*, vol. 21, no. 1, pp. 128–134, Jul. 2006.

[64] M. Belkin and P. Niyogi, “Towards a theoretical foundation for Laplacian-based manifold methods,” *Journal of Computer and System Sciences*, vol. 74, no. 8, pp. 1289–1308, Dec. 2008.

[65] J. Sun, M. Ovsjanikov, and L. Guibas, “A concise and provably informative multi-scale signature based on heat diffusion,” in *Symposium on Geometry Process.*, Jul. 2009, pp. 1383–1392.

[66] M. M. Bronstein and I. Kokkinos, “Scale-invariant heat kernel signatures for non-rigid shape recognition,” in *IEEE Conf. on Comp. Vision and Pattern Recogn.*, Jun., 2010, pp. 1704–1711.

[67] M. Aubry, U. Schlickewei, and D. Cremers, “The wave kernel signature: A quantum mechanical approach to shape analysis,” in *Proc. IEEE Int. Conf. Computer Vision*, Nov. 2011, pp. 1626–1633.

[68] R. Litman and A. M. Bronstein, “Learning spectral descriptors for deformable shape correspondence,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 1, pp. 171–180, Jan. 2014.

[69] N. Hu, R. M. Rustamov, and L. Guibas, “Stable and informative spectral signatures for graph matching,” in *IEEE Conf. on Comp. Vision and Pattern Recogn.*, Jun. 2014, pp. 2313 – 2320.

[70] W. H. Kim, M. K. Chung, and V. Singh, “Multi-resolution shape analysis via non-euclidean wavelets: Applications to mesh segmentation and surface alignment problems,” in *IEEE Conf. on Comp. Vision and Pattern Recogn.*, Jun. 2013, pp. 2139–2146.

[71] W. Kim, S. Narang, and A. Ortega, “Graph based transforms for depth video coding,” in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process.*, Mar. 2012, pp. 813–816.

[72] W. Hu, G. Cheung, A. Ortega, and O. C. Au, “Multiresolution graph Fourier transform for compression of piecewise smooth images,” *IEEE Trans. Image Process.*, vol. 24, no. 1, pp. 419–433, Jan. 2015.

[73] G. Shen, W. Kim, S. Narang, A. Ortega, J. Lee, and H. Wey, “Edge-adaptive transforms for efficient depth map coding,” in *Proc. of the Pict. Coding Sympos.*, Dec. 2010, pp. 566–569.

[74] C. Zhang and D. Florêncio, “Analyzing the optimality of predictive transform coding using graph-based models,” *IEEE Signal Process. Lett.*, vol. 20, no. 1, pp. 106–109, Jan. 2013.

[75] T. Maugey, A. Ortega, and P. Frossard, “Graph-based representation for multiview image geometry,” *IEEE Trans. Image Process.*, vol. 24, no. 5, pp. 1573–1586, May 2015.

[76] T. Maugey, Y. H. Chao, A. Gadde, A. Ortega, and P. Frossard, “Luminance coding in graph-based representation of multiview images,” in *Proc. IEEE Int. Conf. on Image Process.*, Sept. 2014, pp. 130–134.

[77] D. Tian, H. Mansour, A. Knyazev, and A. Vetro, “Chebyshev and conjugate gradient filters for graph image denoising,” in *Proc. of IEEE Inter. Conf. on Multimedia and Expo Workshops*, Jul. 2014, pp. 1–6.

[78] P. Wan, G. Cheung, D. Florêncio, C. Zhang, and O. C. Au, “Image bit-depth enhancement via maximum-a-posteriori estimation of graph AC component,” in *Proc. IEEE Int. Conf. on Image Process.*, Sept. 2014, pp. 4052–4056.

[79] H. Q. Nguyen, P. A. Chou, and Y. Chen, “Compression of human body sequences using graph wavelet filter banks,” in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process.*, May 2014, pp. 6152 – 6156.

[80] C. Loop, C. Zhang, and Z. Zhang, “Real-time high-resolution sparse voxelization with application to image-based modeling,” in *High-Performance Graphics Conf.*, Jul. 2013, pp. 73–79.

[81] Y. Huang, J. Peng, C. C. J. Kuo, and M. Gopi, “A generic scheme for progressive point cloud coding,” *IEEE Trans. Vis. Comput. Graph.*, vol. 14, no. 2, pp. 440–453, Mar. 2008.

[82] R. Schnabel and R. Klein, “Octree-based point-cloud compression,” in *Proc. of Symposium on Point-Based Graphics*, Jul. 2006.

[83] D. Thanou, P. A. Chou, and P. Frossard, “Graph-based motion estimation and compensation for dynamic 3D point cloud compression,” in *Proc. IEEE Int. Conf. on Image Process.*, Sep. 2015.

[84] D. Thanou, P. A. Chou, and P. Frossard, “Graph-based compression of dynamic 3d point cloud sequences,” *Submitted to IEEE Trans. Image Process.*, Jun. 2015.

[85] D. I Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains,” *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, May 2013.

[86] S.-C. T. Choi and M. A. Saunders, “MINRES-QLP for symmetric and hermitian linear equations and least-squares problems,” *ACM Trans. Math. Softw.*, vol. 40, no. 2, Feb. 2014.

[87] X. Zhu and M. Rabbat, “Approximating signals supported on graphs,” in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process.*, Jul. 2012, pp. 3921–3924.

[88] H. S. Malvar, “Adaptive run-length / golomb-rice encoding of quantized generalized gaussian sources with unknown statistics,” in *Proc. of Data Compression Conf.*, Mar. 2006.

[89] A. Collet, M. Chuang, P. Sweeney, D. Gillett, D. Evseev, D. Calabrese, H. Hoppe, A. Kirk, and S. Sullivan, “High-quality streamable free-viewpoint video,” *ACM Trans. Graph.*, vol. 34, no. 4, pp. 69:1–69:13, Aug. 2015.

[90] J. Katajainen and E. Mäkinen, “Tree compression and optimization with applications,” *Int. Journ. Found. Comput. Science*, vol. 1, no. 4, pp. 425–448, 1990.

[91] T. Ochotta and D. Saupe, “Compression of point-based 3D models by shape-adaptive wavelet coding of multi-height fields,” in *Proc. of Eurographics Conf. on Point-Based Graphics*, Jun. 2004, pp. 103–112.

[92] O. Devillers and P-M. Gandoin, “Geometric compression for interactive transmission,” in *Proc. of IEEE Visualization*, Oct. 2000, pp. 319–326.

[93] J. Digne, R. Chaine, and S. Valette, “Self-similarity for accurate compression of point sampled surfaces,” *Comput. Graph. Forum*, vol. 33, no. 2, pp. 155–164, 2014.

[94] J. Ahn, K. Lee, J. Sim, and C. Kim, “Large-scale 3d point cloud compression using adaptive radial distance prediction in hybrid coordinate domains,” *IEEE Journal Selec. Topics Signal Process.*, vol. 9, no. 3, pp. 422–434, Apr. 2015.

[95] M. Irani and P. Anandan, “About direct methods,” in *Inter. Workshop on Vision Algorithms*, Sept., 1999, pp. 267–277.

[96] P. H. S. Torr and A. Zisserman, “Feature based methods for structure and motion estimation,” in *Vision Algorithms: Theory and Practice*, 2000, pp. 278–294.

[97] D. Lowe, “Distinctive image features from scale-invariant keypoints,” *Inter. Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[98] H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” in *European Conf. on Comp. Vision*, May 2006, pp. 404–417.

[99] J. Matas, O. Chum, M. Urban, and T. Pajdla, “Robust wide-baseline stereo from maximally stable extremal regions,” *Image and Video Computing*, vol. 22, no. 10, pp. 761–767, Sept. 2004.

[100] P. Scovanner, S. Ali, and M. Shah, “A 3-dimensional sift descriptor and its application to action recognition,” in *Proc. Inter. Conf. on Multimedia*, Sept. 2007, pp. 357–360.

[101] F. Tombari, S. Salti, and L. di Stefano, “Unique signatures of histograms for local surface description,” in *European Conf. on Comp. Vision*, Sept. 2010, pp. 356–369.

[102] A. Zaharescu, E. Boyer, K. Varanasi, and R. Horaud, “Surface feature detection and description with applications to mesh matching,” in *IEEE Conf. on Comp. Vision and Pattern Recogn.*, Jun. 2009, pp. 373–380.

[103] F. Tombari, S. Salti, and L. di Stefano, “A combined texture-shape descriptor for enhanced 3D feature matching,” in *Proc. IEEE Int. Conf. on Image Process.*, Sept. 2011, pp. 809–812.

[104] H. Chen and B. Bhanu, “3D free-form object recognition in range images using local surface patches,” *Pattern Recogn. Lett.*, vol. 28, no. 10, pp. 1252–1262, Jul. 2007.

[105] I. Sipiran and B. Bustos, “A robust 3D interest points detector based on Harris operator,” in *Eurographics Conf. on 3D Object Retrieval*, 2010, pp. 7–14.

[106] L. A. Alexandre, “3D descriptors for object and category recognition: a comparative evaluation,” in *IEEE/RSJ Inter. Conf. on Intelligent Robots and Systems*, Oct. 2012, pp. 1–6.

[107] J. Peng, C. Kim, and C. Jay Kuo, “Technologies for 3D mesh compression: A survey,” *Journal of Vis. Comun. and Image Represent.*, vol. 16, no. 6, pp. 688–733, Dec. 2005.

[108] J. Rossignac, “Edgebreaker: Connectivity compression for triangle meshes,” *IEEE Trans. on Visualization and Computer Graphics*, vol. 5, no. 1, pp. 47–61, Jan. 1999.

[109] M. Alexa and W. Müller, “Representing animations by principal components,” *Comput. Graph. Forum*, vol. 19, no. 3, pp. 411–418, Sep. 2000.

[110] L. Váša and V. Skala, “Geometry driven local neighborhood based predictors for dynamic mesh compression,” *Comput. Graph. Forum*, vol. 29, no. 6, pp. 1921–1933, 2010.

[111] S.-R. Han, T. Yamasaki, and K. Aizawa, “Time-varying mesh compression using an extended block matching algorithm,” *IEEE Trans. Circuits Syst. Video Techn.*, vol. 17, no. 11, pp. 1506–1518, Nov. 2007.

[112] S. Gupta, K. Sengupta, and A. A. Kassim, "Registration and partitioning-based compression of 3D dynamic data," *IEEE Trans. Circuits Syst. Video Techn.*, vol. 13, no. 11, pp. 1144–1155, Nov. 2003.

[113] A. Doumanoglou, D. S. Alexiadis, D. Zarpalas, and P. Daras, "Toward real-time and efficient compression of human time-varying meshes," *IEEE Trans. Circuits Syst. Video Techn.*, vol. 24, no. 12, pp. 2099–2116, Dec. 2014.

[114] X. Gu, S. J. Gortler., and H. Hoppe, "Geometry images," in *Annual Conf. on Computer Graphics and Interactive Techniques*, 2002, pp. 355–361.

[115] H. M. Briceno, P. V. Sander, L. McMillan, S. Gortler, and H. Hoppe, "Geometry Videos: A New Representation for 3D Animations," in *ACM Symp. on Computer Animation*, 2003, pp. 136–146.

[116] H. Habe, Y. Katsura, and T. Matsuyama, "Skin-off: Representation and compression scheme for 3D video," in *Picture Coding Symp.*, Jan. 2004, pp. 301–306.

[117] J. Hou, L. Chau, N. Magnenat-Thalmann, and Y. He, "Compressing 3D human motions via keyframe-based geometry videos," *IEEE Trans. Circuits Syst. Video Techn.*, vol. 25, no. 1, pp. 51–62, Jan. 2015.

[118] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *Annual Conf. on Comp. Graphics and Interactive Techniques*, 1987, pp. 163–169.

[119] R. Rubinstein, A. M. Bruckstein, and M. Elad, "Dictionaries for sparse representation modeling," *Proc. of the IEEE*, vol. 98, no. 6, pp. 1045 –1057, Apr. 2010.

[120] K. Engan, S. O. Aase, and J. H. Husoy, " Method of optimal directions for frame design," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process.*, Mar. 1999, vol. 5, pp. 2443–2446.

[121] N. J. Higham, *Functions of Matrices*, Society for Industrial and Applied Mathematics, 2008.

[122] D. Thanou, D. I Shuman, and P. Frossard, "Parametric dictionary learning for graph signals," in *Proc. IEEE Glob. Conf. Signal and Inform. Process.*, Dec. 2013.

[123] D. Thanou, D. I Shuman, and P. Frossard, "Learning parametric dictionaries for signals on graphs," *IEEE Trans. Signal Process.*, vol. 62, no. 15, pp. 3849–3862, Aug. 2014.

[124] D. Thanou and P. Frossard, "Multi-graph learning of spectral graph dictionaries," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process.*, Apr. 2015, pp. 3397–3401.

[125] J. A. Tropp, "Greed is good: Algorithmic results for sparse approximation," *IEEE Trans. Inform. Theory*, vol. 50, no. 10, pp. 2231–2242, Oct. 2004.

[126] M. Bruckstein, D. Donoho, and M. Elad, "From sparse solutions of systems of equations to sparse modeling of signals and images," *SIAM Rev.*, vol. 51, no. 1, pp. 34–81, Feb. 2009.

[127] M. Elad, *Sparse and Redundant Representations - From Theory to Applications in Signal and Image Processing*, Springer, 2010.

[128] S. Boyd and L. Vandenberghe, *Convex Optimization*, New York: Cambridge University Press, 2004.

[129] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[130] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Commun. Pure Appl. Math.*, vol. 57, no. 11, pp. 1413–1457, Nov. 2004.

[131] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal Royal Stat. Society Ser. B*, vol. 58, pp. 267–288, 1994.

[132] K.C. Toh, M.J. Todd, and R.H. Tutuncu, "SDPT3 — a MATLAB software package for semidefinite programming," in *Optimization Methods and Software*, 1999, vol. 11, pp. 545–581.

[133] J. Lofberg, "YALMIP: A toolbox for modeling and optimization in MATLAB," in *Proc. of IEEE Comp. Aided Control Syst.Design Conf.*, Sept. 2004, pp. 284 – 289.

[134] T. Choe, A. Skabardonis, and P. P. Varaiya, "Freeway performance measurement system (PeMS): an operational analysis tool," in *Annual Meeting of Transportation Research Board*, Jan. 2002.

[135] H. Eryilmaz, D. Van De Ville, S. Schwartz, and P. Vuilleumier, "Impact of transient emotions on functional connectivity during subsequent resting state: A wavelet correlation approach," *NeuroImage*, vol. 54, no. 3, pp. 2481–2491, Feb. 2011.

[136] J. Richiardi, H. Eryilmaz, S. Schwartz, P. Vuilleumier, and D. Van De Ville, "Decoding brain states from fMRI connectivity graphs," *NeuroImage*, vol. 56, no. 2, pp. 616–626, May 2011.

[137] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proc. of the ACM SIGKDD Inter. Conf. on Knowledge Discovery in Data Mining*, 2005, pp. 177–187.

[138] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.

[139] R. Kondor and J. Lafferty, "Diffusion kernels on graphs and other discrete structures," in *Inter. Conf. of Machine Learn.*, Jul. 2002, pp. 315–322.

[140] S. ChenS, D. Donoho, and M. Saunders, "Atomic decomposition by basis pursuit," *SIAM Rev.*, vol. 43, no. 1, pp. 129–159, Jan. 2001.

[141] A. Beck and M. Teboulle, “A fast iterative shrinkage-thresholding algorithm for linear inverse problems,” *SIAM Journ. Image Science*, vol. 2, no. 1, pp. 183–202, Mar. 2009.

[142] P. Combettes and J-C. Pesquet, “Proximal Splitting Methods in Signal Processing,” in *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, pp. 185–212. Springer, 2011.

[143] S. Chen, D. Donoho, and M. Saunders, “Atomic decomposition by basis pursuit,” *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 33–61, Aug. 1999.

[144] S. Sra, “Scalable nonconvex inexact proximal splitting,” in *Adv. Neural Inf. Process. Syst.*, Dec. 2012, pp. 530–538.

[145] D. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Englewood Cliffs, NJ: Prentice-Hall, 1989.

[146] I. D. Schizas, A. Ribeiro, and G. Giannakis, “Consensus in ad hoc WSNs with noisy links Part I: Distributed estimation of deterministic signals,” *IEEE Trans. Signal Process.*, vol. 56, pp. 350–364, Jan. 2008.

[147] A. Sandryhaila, S. Kar, and J. M. F. Moura, “Finite-time distributed consensus through graph filters,” in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process.*, May 2014, pp. 1080–1084.

[148] L. Xiao, S. Boyd, and S-J. Kim, “Distributed average consensus with least-mean-square deviation,” *Journ. of Parallel and Distributed Computing*, vol. 67, no. 1, pp. 34–46, Jan. 2007.

[149] D. Thanou, E. Kokiopoulou, and P. Frossard, “Progressive Quantization in Distributed Average Consensus,” in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process.*, Mar. 2012, pp. 2677–2680.

[150] D. Thanou, E. Kokiopoulou, Y. Pu, and P. Frossard, “Distributed average consensus with quantization refinement,” *IEEE Trans. Signal Process.*, vol. 61, no. 1, pp. 194–205, Jan. 2013.

[151] P. Frasca, R. Carli, F. Fagnani, and S. Zampieri, “Average consensus on networks with quantized communication,” *Int. Journal of Robust and Nonlinear Control*, vol. 19, no. 16, pp. 1787–1816, Nov. 2009.

[152] S. Tetikonda and S. Mitter, “Control under communication constraints,” *IEEE Trans. on Automatic Control*, vol. 49, no. 7, pp. 1056–1068, Jul. 2004.

[153] P. Gupta and P. R. Kumar, “The capacity of wireless networks,” *IEEE Trans. on Information Theory*, vol. 46, no. 2, pp. 388–404, Mar. 2000.

[154] M. Yildiz and A. Scaglione, “Coding with side information for rate-constrained consensus,” *IEEE Trans. Signal Process.*, vol. 56, no. 8, Aug. 2008.

[155] T. Aysal, M.Coates, and M. Rabbat, "Distributed Average Consensus with Dithering Quantization," *IEEE Trans. Signal Process.*, vol. 56, no. 10, Oct. 2008.

[156] T. Basar A. Kashyap and R. Srikant, "Quantized consensus," *Automatica*, vol. 43, no. 7, pp. 1192–1203, Apr. 2007.

[157] J. Fang and H. Li, "Distributed consensus with quantized data via sequence averaging," *IEEE Trans. Signal Process.*, vol. 58, no. 2, Feb. 2010.

[158] S. Kar and J. M. F. Moura, "Distributed consensus algorithms in sensor networks: quantized data and random link failures," *IEEE Trans. Signal Process.*, vol. 58, no. 3, pp. 1383–1400, Mar. 2010.

[159] C. Mosquera, R. López-Valcarce, and S. Jayaweera, "Stepsize Sequence Design for Distributed Average Consensus," *IEEE Signal Processing Letters*, vol. 17, no. 2, pp. 169–172, Feb. 2010.

[160] Q. Zhang and J. Zhang, "Quantized data-based distributed consensus under directed time-varying communication topology," *SIAM Journ. Control Optim.*, vol. 51, no. 1, pp. 332–352, Jul. 2013.

[161] S. Zhu, Y. Chai Soh, and L. Xie, "Distributed parameter estimation with quantized communication via running average," *IEEE Trans. Signal Process.*, vol. 63, no. 17, pp. 4634–4646, Sept. 2015.

[162] Y. Pu, M. N. Zeilinger, and C. N. Jones, "Quantization design for distributed optimization," *ArXiv, available at http://arxiv.org/abs/1504.02317*, 2015.

[163] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst, "Learning laplacian matrix in smooth graph signal representations," *Submitted to IEEE Trans. Signal Process.*, Jun. 2015.

[164] J. Bolte, S. Sabach, and M. Teboulle, "Proximal alternating linearized minimization for nonconvex and nonsmooth problems," *Math. Program.*, vol. 146, no. 1-2, pp. 459–494, Aug. 2014.

[165] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, 1996.

[166] D. Mayers and E. Süli, *An introduction to numerical analysis*, Cambridge Univ. Press, Cambridge, 2003.

[167] K. Atkinson, *An Introduction to Numerical Analysis*, Wiley, 1989.

[168] A. Sripad and D. Snyder, "A necessary and sufficient condition for quantization errors to be uniform and white," *IEEE Trans. Acoust., Speech and Signal Processing*, vol. 25, no. 5, pp. 442–448, Oct. 1977.

[169] D. G. Luenberger, *Introduction to Dynamic Systems: Theory, Models, and Applications*, Wiley, NY, 1979.

Curriculum Vitae

Personal Information

Dorina Thanou
EPFL STI IEL LTS4
ELE 238, Station 11
CH-1015, Lausanne, Switzerland

Date of birth: 25.06.1985
Nationality: Greek
http://lts4.epfl.ch/thanou
dorina.thanou@epfl.ch

Research Interests

Signal processing on graphs, Sparse representations, Machine learning, Distributed signal processing

Education

EPFL, Lausanne, Switzerland **Sept. 2010-Dec. 2015**
Signal Processing Laboratory (LTS4), Department of Electrical Engineering
Ph.D. student

- PhD thesis: "Graph Signal Processing: Sparse representation and applications". Thesis advisor: Prof. Pascal Frossard

EPFL, Lausanne, Switzerland **Sept. 2008-Aug. 2010**
School of Computer and Communications Sciences
MSc in Communication Systems, Specialization in Wireless Communications

- Master Thesis: "Linear dimensionality reduction for classification of compressed observation sets".
  Thesis advisor: Prof. Pascal Frossard

University of Patras, Patras, Greece **Sept. 2003-July 2008**
Department of Electrical and Computer Engineering
Diploma of Electrical and Computer Engineering, Specialization in Telecommunications and Information Technology

- Diploma thesis: "Architecture Implementations of Ultra-Wideband Communication Systems based on multiband OFDM modulation techniques".
  Thesis advisor: Prof. Thanos Stouraitis

Work experience

EPFL, Lausanne, Switzerland **Sept. 2010-Present**
Signal Processing Laboratory (LTS4)
Research and teaching assistant

- Researched the representation, analysis and efficient processing of high-dimensional structured data. Developed a novel framework for sparse representation of signals living on irregular, graph domains.
- Proposed an algorithm for learning the underlying graph structure of the data and inferring connections on networks. The algorithm was successfully applied in mining political data, and predicting weather temperatures.
- Studied the effect of quantization in distributed graph signal processing. Developed an algorithm that exploits the network topology for efficient quantization of signals in distributed sensor networks under communication constraints.
- Supervised bachelor and master students.

Microsoft Research, Redmond, WA **June 2014-Aug. 2014**
Multimedia, Interaction, and Communication (MIC) group
Research Intern
Mentor: Dr. Philip Chou

- Developed a novel algorithm for motion estimation in point cloud sequences, by finding correspondences between consecutive frames of the sequence. The algorithm was used for the efficient compression of such sequences.

Honors and Awards

Best student paper award in the IEEE International Conference on Speech and Signal Processing (ICASSP) 2015.

Top 10% paper award in the IEEE International Conference in Image Processing (ICIP), 2015.

Technical Skills

Programming: Matlab, C, Python
Optimization Packages: CVX, YALMIP
Configuration tools: SVN, Git
Operating Systems: MS-Windows, Mac OS X, Linux, UNIX.

Academic services

Reviewer: IEEE Transactions on Signal Processing, IEEE Transactions on Image Processing, ICASSP, EUSIPCO.

Teaching assistant: Digital Signal Processing (EPFL): Spring 2011/2012/2013

Language Skills

Greek (native), English (C2), Italian (C2), French (B1)

Publications

Journal publications

1. D. Thanou, P. A. Chou, and P. Frossard. Graph-based compression of dynamic 3D point cloud sequences, *Submitted to IEEE Trans. on Image Proc.*, June 2015.
2. X. Dong, D. Thanou, P. Frossard and P. Vandergheynst. Learning Laplacian Matrix in Smooth Graph Signal Representations, *Submitted to IEEE Trans. on Signal Proc.*, June 2015.
3. D. Thanou, D. I Shuman, and P. Frossard. Learning parametric dictionaries for signals on graphs, *IEEE Trans. on Signal Proc.*, vol. 62, no. 15, pp. 3849-3862, Aug. 2014.
4. D. Thanou, E. Kokiopoulou, Y. Pu, and P. Frossard. Distributed average consensus with quantization refinement, *IEEE Trans. on Signal Proc.*, vol. 61, no.1, pp. 194-205, Jan. 2013.

Conference publications

1. D. Thanou and P. Frossard. Distributed signal processing with graph spectral dictionaries, in *Proc. of Annual Allerton Conf. on Comm., Control, and Computing*, UIUC, IL, USA, Oct. 2015.
2. D. Thanou, P. A. Chou, and P. Frossard. Graph-based motion estimation and compensation for dynamic 3D point cloud compression, in *Proc. of IEEE Inter. Conf. on Image Proc. (ICIP)*, Quebec City, Canada, Sept. 2015. (**top 10%**)
3. D. Thanou and P. Frossard. Multi-graph learning of spectral graph dictionaries, in *Proc. of IEEE Inter. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*, Brisbane, Australia, Apr. 2015. (**Best student paper award**)
4. X. Dong, D. Thanou, P. Frossard and P. Vandergheynst. Laplacian matrix learning for smooth graph signal representation, in *Proc. of IEEE Inter. Conf.*

on Acoustics, Speech and Signal Proc. (ICASSP), Brisbane, Australia, Apr. 2015.

5. D. Thanou, D. I Shuman, and P. Frossard. Parametric dictionary learning for graph signals, in *Proc. of IEEE Glob. Conf. Signal and Inform. Proc. (GlobalSIP)*, Austin, Texas, Dec. 2013.
6. D. Thanou, E. Kokiopoulou, and P. Frossard. Progressive quantization in distributed average consensus, in *Proc. of IEEE Inter. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*, Kyoto, Japan, Mar. 2012.
7. D. Thanou and P. Frossard. Compressed classification of observation sets with linear subspace embeddings, in *Proc. of IEEE Inter. Conf. on Acoustics, Speech and Signal Proc. (ICASSP)*, Prague, Czech Republic, May 2011.
8. D. Thanou, H. Park, E. Kokiopoulou, P. Frossard, Polynomial filter design for quantized consensus, in *Proc. of European Signal Proc. Conf. (EUSIPCO)*, Aalborg, Denmark, Aug. 2010.
9. E. Fotopoulou, D. Thanou, Th. Stouraitis, Comparison of time and frequency domain interpolation implementations for MB-OFDM UWB transmitters, in *Proc. of IEEE Inter. Symp. on Circuits and Systems (ISCAS)*, Paris, France, Jun. 2010.