DIPLOMA THESIS

Quantisation and source-coding for graph signal processing

submitted at Technische Universität Wien,

Faculty of Electrical Engineering and Information Technology

for the purpose of obtaining the academic degree of a Dipl.-Ing. under supervision of

Univ.-Prof. Dr. Gerald Matz

at the

Institute of Telecommunications

by **Philipp Reingruber**

Student ID: 11709531

e-mail: philipp.reingruber@tuwien.ac.at

in June 2024

Contents

Acknowledgements

I want to thank Gerald Matz for his judicious, inspiring and appreciative feedback and supervision throughout the process of writing this thesis. In our extensive, frequent meeting over the past months, I learned a lot.

Furthermore, I am glad of the valuable exchange with my friend and colleague Dimitrios Kalodikis throughout our studies. From classical signal processing to control engineering to graph signal processing, I always left our lively discussion with a clearer view.

I am very grateful for the emotional and moral (but never moralising) support by my partner Hannah, my family and my friends that gave me lightness even in tougher periods.

Last but not least, thanks to the countless unsung heroes on online communities and forums like StackExchange. Be it trivial or detailed aspects, general and highly specific statements, mathematical or programming-related problems; you make knowledge more accessible to all of us.

Abstract

Instead of modelling data on regular domains as in classical 1-D or 2-D signal processing, the field of graph signal processing allows for data to be described and processed on irregular domains such as in sensor networks or in social networks. However, the question of source coding of graph signals on these domains has not been sufficiently addressed in the literature.

This thesis introduces and discusses transform coding techniques for graph signals. We focus on nonstationary processes designed by filtering and modulating white noise or via banded frequency correlation matrices. The compression methods are tested in simulations on perturbed random regular graphs, random geometric graphs, and Barabási–Albert graphs. Our simulations show that quantisation quality is related to correlation and sparsity of the transform coefficients.

A compression technique that was proposed in literature is found to suffer from numerical instability and high computational complexity. Perfect decorrelation using the Karhunen-Loève transform also has high computational cost. Thus, we propose three compression techniques in three different domains. With the first method, we sample in the vertex domain and recover the signal using bandlimited reconstruction with the quantised samples. This method yields acceptable numerical results only with significant oversampling in scenarios with low vertex-domain correlation.

As an alternative, we propose compression in the graph frequency domain. For the nonstationary signal models considered, this method shows the most robust results. Compression quality remains acceptable despite correlation since (especially bandlimited) signals can be sparsely represented. Both techniques approach the rate-distortion limit only in low-rate scenarios. In many scenarios, outliers increase the mean distortion significantly.

Motivated by its counterpart in classical signal processing, we introduce the graph Gabor transform (GGT), a sampled version of the windowed graph Fourier transform (GFT). The sampling grid should be matched to the correlation in vertex and

frequency domain to obtain less correlated transform coefficients. However, undesirable properties of the GFT impair the grid matching. Moreover, the windowed GFT atoms generally do not form a tight frame and its translation and modulation operators do not have group structure. This limits the use cases of the GGT to high-rate compression scenarios with weakly correlated processes on low-coherence graphs.

Zusammenfassung

Die Graphsignalverarbeitung erlaubt, Daten auf unregelmäßigen Domänen – wie etwa auf Sensor- oder sozialen Netzwerken – darzustellen, anders als in der klassischen 1-D- oder 2-D-Signalverarbeitung. Die Frage der Quellkodierung von Graphsignalen auf diesen Domänen wurde jedoch bisher nicht ausreichend in der Literatur behandelt.

In der vorliegenden Arbeit werden Transformationskodierungsverfahren für Graphsignale vorgestellt und diskutiert. Wir richten unser Augenmerk auf nichtstationäre Prozesse, welche wir durch Filterung und Modulation sowie durch bandförmige Frequenzkorrelationsmatrizen konstruieren. Die Kompressionsmethoden werden in Simulationen auf perturbierten zufälligen regulären Graphen, zufälligen geometrischen Graphen und Barabási-Albert-Graphen getestet. Unsere Simulationen zeigen, dass die Quantisierungsqualität von der Korrelation und dem Besetzungsgrad der Transformationskoeffizienten abhängt.

Ein in der Literatur vorgeschlagenes Kompressionsverfahren stellt sich als numerisch instabil und rechenaufwändig heraus. Perfekte Dekorrelation mittels der Karhunen-Loève-Transformation ist ebenso mit großem Rechenaufwand verbunden. Daher stellen wir drei Kompressionsverfahren in drei unterschiedlichen Domänen vor. In der ersten Methode tasten wir im Knotenbereich ab und stellen das Signal mittels bandbeschränkter Rekonstruktion unter Verwendung der quantisierten Abtastwerte wieder her. Diese Methode liefert nur mit erheblicher Überabtastung bei schwacher Korrelation im Knotenbereich brauchbare Ergebnisse.

Alternativ schlagen wir eine Kompression im Graphfrequenzbereich vor, da dies für die betrachteten nichtstationären Signalmodelle die robustesten Ergebnisse erzielt. Trotz Korrelation bleibt die Kompressionsqualität annehmbar, da (insbesondere bandbeschränkte) Signale durch dünnbesetzte Koeffizientenvektoren dargestellt werden können. Beide Verfahren kommen der Rate-Distortion-Grenze nur in niederratigen Szenarien nahe. In vielen Fällen erhöhen Ausreißer die mittlere Verzerrung deutlich.

Inspiriert von ihrem Pendant in der klassischen Signalverarbeitung führen wir die Graph-Gabor-Transformation (GGT) ein, eine abgetastete Version der gefensterten Graph-Fourier-Transformation (GFT). Das Abtastungsraster sollte an die Korrelation im Knoten- und Frequenzbereich angepasst sein, um schwächer korrelierte Transformationskoeffizienten zu erhalten. Unerwünschte Eigenschaften der GFT beeinträchtigen jedoch die Rasteranpassung. Darüber hinaus bilden weder die gefensterten GFT-Atome einen tight Frame, noch haben ihre Translations- und Modulationsoperatoren Gruppenstruktur. Dies beschränkt den Anwendungsbereich der GGT auf hochratige Kompressionsszenarien mit schwach korrelierten Prozessen auf Graphen mit niedriger Kohärenz.

1 Introduction

Until the advent of image processing, signal processing had only dealt with one-dimensional signals. First in the continuous, then in the discrete domain. As *Video Killed the Radio Star,* signal processing techniques were generalised for multi-dimensional signals. In recent decades, the idea of further generalisations for signals on irregular domains has become popular. Since we can represent these structures as graphs, we refer to this field as graph signal processing [1].

A graph consists of nodes (also called vertices) to which a signal value is associated. Two nodes each are connected by an edge. In regular domains, all signal values[1] have the same number of incident edges, called degree: Depending on whether it is periodic or not, a discrete-time signal can be formulated as a ring or path graph, where one node is connected to its predecessor and successor. An image would correspond to a graph whose nodes are arranged on a square lattice and have edges to their left, right, upper and lower neighbour. In irregular domains, the node degree varies. By generalising signal processing techniques for these cases, information hidden in structures like sensor networks, social networks or protein graphs (to name a few) can be uncovered.

Processing and storing digital signals happens at a finite resolution: The data values are represented by a finite number of so-called *reproducer values* which themselves are represented by a limited number of bits. The question arises which way of representation yields less distortion at a certain bit rate. While this question has been thoroughly discussed in classical signal processing [2], this is not the case for graph signal processing. In this thesis, we propose, implement and compare several strategies for graph signal quantisation:

- Vertex-domain sampling and quantisation;
- The method presented in [3];

[1]This statement does not consider potential border values.

- A modification of the method in [3], where the bit allocation is done more efficiently;
- A method in the graph Fourier transform (GFT);
- A method in the the graph Gabor transform (GGT) domain.

Apart from a rate-distortion perspective, computational complexity is another performance criterium that needs to be assessed. We will pay specific attention to several potentially difficult aspects:

- Transform-based quantisation can be computationally expensive since the transforms may not have favourable structure;
- low-rate quantisation may have potentially disastrous distortion effects;
- some of the methods require repeated computation of non-trivial matrices for optimisation algorithms;
- graph transforms may not have the same desirable properties that are convenient and in conventional signal processing.

The goal is to obtain precise and fast compression algorithms that preserve a substantial amount of the high-dimensional data stored in graph structures for applications like communication networks, social networks or sensor networks.

The remaining thesis is structured as follows: Chapter 2 discusses fundamental concepts of source coding that are not tailored to but also applicable to graph signals. In Chapter 3, we present the basics of graph topology and graph signal processing as well as nonstationary processes on graphs. Chapter 4 covers the above-mentioned graph signal quantisation techniques that have then been tested in simulations that we analyse in Chapter 5. Chapter 6 summarises the thesis. The Python implementations of the compression techniques that we analysed can be found in Appendix A.

With this thesis, we contribute various quantisation techniques that are existing graph signal transforms performed with quantised signals. Furthermore, we generalise the Gabor transform as described in [4] for graphs using the windowed graph Fourier transforms (WGFT) that was introduced by [5]. As mentioned above, we propose another quantisation technique based on this Gabor transform. Finally, we implemented the said methods in Python, as well as the method introduced by [3] and compared all working techniques with the theoretical performance limit of the rate-distortion function.

2 Source coding

This chapter deals with the fundamentals of source coding relevant for the remaining chapters of this thesis. These concepts were neither specifically designed nor later generalised for graphs. Instead, they are applicable to signals regardless of their domains.

In Section 2.1, we discuss the theoretical performance boundaries of quantising known as rate-distortion theory. Section 2.2 covers transform coding and thus answers the question why we deal with sampling in different domains in Chapter 4. Finally, Section 2.3 covers a very practical aspect of compression, namely the (uniform) quantiser that is used in all presented compression techniques.

2.1 Rate-distortion theory

Since signals are stored and processed digitally, there is a need to compute a finite-precision representation of them. Regardless of the number of bits used for the representation (called rate), a loss of information (called distortion) is generally inevitable. Rate-distortion theory [2, Ch. 10] answers the question of the minimum expected distortion given a certain rate. The following presentation of this topic is based on [2].

The (code-specific) encoder $f_N : \mathcal{X}^N \to \left\{0, \dots, 2^{NR} - 1\right\}$ operating at rate R maps a vector of the source alphabet $\mathcal{X}$ to an index. The decoder $g_N : \left\{0, \dots, 2^{NR} - 1\right\} \to \hat{\mathcal{X}}^N$ maps the index to a vector with elements from the reproduction alphabet $\hat{\mathcal{X}}$. The squared-error distortion is the most popular distortion function $d : \mathcal{X}^N \times \hat{\mathcal{X}^N} \to \mathbb{R}$ between a source vector $\mathbf{x} = (x_1, \dots, x_N)^T$ and its reproducer vector $\hat{\mathbf{x}} = (\hat{x}_1, \dots, \hat{x}_N)^T$, i.e., its representation. It is given by

$$d\left(\mathbf{x}, \hat{\mathbf{x}}\right) := \sum_{n=1}^{N} \left(x_n - \hat{x}_n\right)^2 .$$

The average distortion D of a source code for a random vector $\mathbf{x}$ equals

$$D \coloneqq \mathrm{E}\left\{d\left(\mathbf{x}, g_N\left(f_N\left(\mathbf{x}\right)\right)\right)\right\},$$

where $\mathrm{E}\left\{\cdot\right\}$ denotes the expectation.

This way, the rate R is coupled to the distortion D for a specific code; we thus refer to this pair as a rate-distortion pair (R, D). The rate-distortion function $R\left(D\right)$ now describes the operational limit of the smallest possible rate R, given a distortion D. Shannon's rate-distortion theorem states that

$$R\left(D\right) = R^{(I)}\left(D\right) \coloneqq \min_{f_{\hat{\mathbf{x}}|\mathbf{x}}(\hat{\mathbf{x}}|\mathbf{x}):\mathrm{E}\{d(\mathbf{x},\hat{\mathbf{x}})\}\leq D} I\left(\mathbf{x}, \hat{\mathbf{x}}\right),$$

i.e., that the operational rate-distortion function is equal to the information rate-distortion function. Here, $f_{\hat{\mathbf{x}}|\mathbf{x}}\left(\hat{\mathbf{x}} \mid \mathbf{x}\right)$ denotes a joint conditional probability density function that satisfies the stated constraint. The mutual information $I\left(\mathbf{x}, \hat{\mathbf{x}}\right)$ is the reduction in uncertainty about $\mathbf{x}$ due to knowing $\hat{\mathbf{x}}$.

For Gaussian sources, we know an analytic expression for the rate-distortion function $R\left(D\right)$. More specifically, we need to take into account the fact that we will be observing multiple independent Gaussian random variables $\mathbf{x} \sim \mathcal{N}\left(\mathbf{0}, \mathrm{Diag}\left(\boldsymbol{\sigma}^2\right)\right)$ at once on our graph (here, $\boldsymbol{\sigma}^2 = \left(\sigma_1^2, \ldots, \sigma_N^2\right)^T$). We will use this rate-distortion function from literature as a reference for implemented scenarios [2, Theorem 10.3.3]:

$$\begin{aligned} R\left(D\right) &= \sum_{n=1}^{N} \frac{1}{2} \log_2\left(\frac{\sigma_n^2}{D_n}\right), \qquad (2.1) \\ D_n &= \begin{cases} \Xi, & \Xi < \sigma_n^2, \\ 0, & \Xi \geq \sigma_n^2, \end{cases} \\ D &= \sum_{n=1}^{N} D_n. \end{aligned}$$

Note, however, that this bound is formulated for vector quantisation. As explained above, this form of quantisation uses reproducer vectors instead of scalars and thus quantises more than one source sample at once. Complexity of such quantisers, however, is a limiting factor for either rate or dimension (i.e., the number of source samples represented by one quantised sample) [6, Sec. 12.1]. We thus focus on scalar quantisation and transform coding, which we discuss in Sec. 2.2. As a certain signal-to-noise ratio (SNR) advantage of vector quantisation in general remains, we may not be able to achieve rate-distortion pairs close to the function in (2.1).

2.2 Transform coding

Exploiting signal correlation for source coding at higher SNR requires vector quantisation [6, Ch. 11]. A different approach is transform coding, where the signal is first transformed into a domain where the samples are (approximately) uncorrelated. In such a domain, the advantage of vector quantisation is diminished[1] and thus, scalar quantisation can be applied effectively [6, Sec. 8.5].

The Karhunen-Loève transform achieves such a decorrelation, i.e., an analysis relation $\mathbf{y} = \mathbf{V}^H\mathbf{x}$ with a unitary $N \times N$ matrix $\mathbf{V}$ such that $\mathbf{C}_\mathbf{y} = \mathrm{Diag}\left(\sigma_{\mathrm{y}_1}^2, \dots, \sigma_{\mathrm{y}_N}^2\right)$. This leads to the following expression for the covariance matrices,

$$\mathbf{C}_\mathbf{x} = \mathbf{V}\mathbf{C}_\mathbf{y}\mathbf{V}^H = \mathbf{V}\mathrm{Diag}\left(\sigma_{\mathrm{y}_1}^2, \dots, \sigma_{\mathrm{y}_N}^2\right)\mathbf{V}^H.$$

Since $\mathbf{V}$ is unitary, this is equivalent to the eigendecomposition of $\mathbf{C}_\mathbf{x}$, i.e., $\mathbf{V} = \mathbf{U} = (\mathbf{u}_1, \dots, \mathbf{u}_N)$, where $\mathbf{u}_k$ is the k^{th} eigenvector of $\mathbf{C}_\mathbf{x}$ and $\sigma_{\mathrm{y}_k}^2 = \lambda_k$, where λ_k is the k^{th} eigenvalue of $\mathbf{C}_\mathbf{x}$ [6, Sec. 8.6].

As eigendecompositions are computationally expensive [7, Sec. 5.3.1] and $\mathbf{C}_\mathbf{x}$ may not be known, in Chapter 4 we are interested in a transform that approximately achieves such a decorrelation, characterised by very small off-diagonal elements $\left(\mathbf{C}_\mathbf{y}\right)_{kl}, k \neq l$.

2.3 Uniform quantiser

Once the signal is in a domain of our choice, we want to store only the reproducer values, i.e., we want to quantise our signal. A quantiser that maps signal values to W reproducer values[2] is characterised by its reproducer values and its decision boundaries q_w [6, Sec. 5.1]. If the signal value x is in the interval (q_w, q_{w+1}), we represent it as the reproducer value $\hat{x}_w$. Signal values $x > q_{W+1}$ are stored as the largest reproducer value $\hat{x}_W$, signal values $x < q_1$ are stored as the smallest

[1]Even for uncorrelated sources, vector quantisation yields a gain in SNR due to flexible Voronoi regions [6, Ch. 11].

[2]For a binary representation of W reproducer values, we need $Q = \log_2 W$ bits.

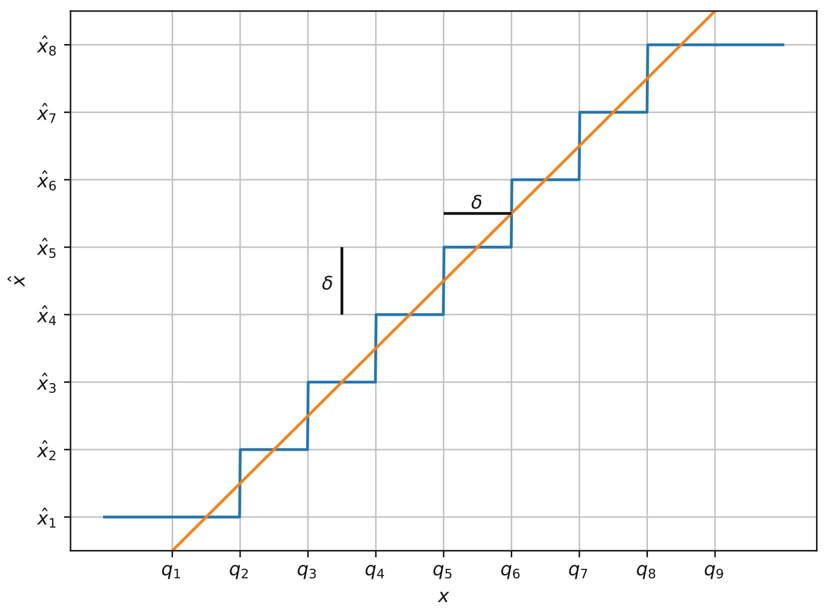


Figure 2.1: Mid-riser uniform quantiser behaviour for $W = 8$.

reproducer value $\hat{x}_1$. These unbounded quantiser intervals are referred to as the overload region.

If all $\hat{x}_w$ are the central points of their quantisation interval, we refer to the quantiser as a mid-riser quantiser.

The most trivial yet common choice for a quantiser is the mid-riser uniform quantiser, as described in [6, Sec. 5.4]. We solely focus on this quantiser since it is computationally efficient and applied in [3] as well. As depicted in Fig. 2.1, the W reproducer values $\hat{x}_w$ are equally spaced over a finite support $[-\gamma, \gamma]$, i.e.,

$$\hat{x}_w = -\gamma + \frac{\delta}{2} + (w-1)\,\delta, \qquad w = 1, \ldots, W,$$

where $\delta = \frac{2\gamma}{W}$ denotes the step size. A quantisation interval is bounded by quantiser decision boundaries q_w that read

$$q_w = -\gamma + \delta\,(w-1)\,, \qquad w = 1, \ldots, W+1.$$

This results in the described mid-riser characteristic

$$\hat{x}_w = \frac{q_w + q_{w+1}}{2}.$$

The error $|\hat{x}_w - x|$ that is introduced by the quantiser cannot exceed $\frac{\delta}{2}$ outside the overload region. In the high resolution case, the overall distortion for the uniform quantiser is

$$D_{\mathcal{Q}} = \frac{\delta^2}{12}. \tag{2.2}$$

Note that the uniform quantiser is not tailored to the probability density function (pdf) of the signal [6, Sec. 5.5]. This leads to a higher distortion at the advantage of computational simplicity and less necessary statistical knowledge about the signal. Only the support of the signal is taken into account. We follow [3] by setting $\gamma = 2\sqrt{\mathrm{E}\left\{x^2\right\}}$ to limit the overloading probability to 5%.

3 Graph signal processing basics

This chapter gives a concise overview of ideas, properties and notation of graph theory that are applied in the remainder of this thesis. Section 3.1 covers fundamental properties of graph structures and their respective mathematical formulations. In Section 3.2, an introduction about signals on graphs based on [8, 5] is presented, including adapted techniques known from classical signal processing. In Section 3.3, we cover bandlimited reconstruction that allows perfect reconstruction for sampled, bandlimited and unquantised signals. Finally, in Section 3.4, concepts for the generation of nonstationary processes on graphs are introduced.

3.1 Topology concepts

Subsection 3.1.1 discusses general structure-related properties of graphs while the second part briefly describes the graph models used for the simulations in Chapter 5.

3.1.1 Foundations of graph topology

A graph is a pair of sets $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \dots, N\}$ is the set of nodes and $\mathcal{E}$ is the set of edges. An edge $e \in \mathcal{E}$ connects two adjacent nodes $m, n \in \mathcal{V}$ in a directed or undirected fashion. In the former case, $\mathcal{G}$ is considered a so-called directed graph; in the latter case, $\mathcal{G}$ is referred to as an undirected graph. Thus, an edge is an unordered pair of nodes, i.e., $e = \{m, n\}$.

Different weights can be associated with the edge set using a weight function $w : \mathcal{E} \to \mathbb{R}$, in case of unweighted graphs, the edge weight is always 1. If every node n can be reached from any node m, we refer to this graph as a connected graph [8, Section 1.1.1]. For simplicity, this thesis solely deals with undirected, connected graphs.

We can fully describe the topology of an unweighted graph by indicating in an adjacency matrix $\mathbf{A}$ which nodes share an edge,

$$(\mathbf{A})_{mn} = \begin{cases} 1, & \{m,n\} \in \mathcal{E}, \\ 0, & \{m,n\} \notin \mathcal{E}. \end{cases}$$

Note that in case of an undirected graph, $\mathbf{A}$ is symmetric. For the topology of a weighted graph, we introduce a weight matrix $\mathbf{W}$ by replacing the non-zero entries of $\mathbf{A}$ by the respective edge weights, i.e.,

$$(\mathbf{W})_{mn} = \begin{cases} w\left(m,n\right), & \{m,n\} \in \mathcal{E}, \\ 0, & \{m,n\} \notin \mathcal{E}. \end{cases}$$

The (weighted) number of nodes connected to node m is referred to as degree $\deg(m)$, where the connections (i.e., the edges) are weighted according to $\mathbf{W}$, i.e.,

$$\deg\left(m\right) \coloneqq \sum_{n=1}^{N} W_{mn} = \sum_{n=1}^{N} W_{nm}.$$

All degrees can be collectively denoted using the diagonal degree matrix [8, Section 2.3.1]

$$\mathbf{D} \coloneqq \operatorname{Diag}\left(\deg\left(1\right), \ldots, \deg\left(N\right)\right).$$

By analogy with the Laplace operator, the Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{W}$ [8, Section 2.3.2] describes information diffusion through a graph. Like its continuous namesake, it can be expressed as the graph divergence of the graph gradient. Furthermore, its eigenvectors and eigenvalues play an essential role in the graph Fourier transform, which will be discussed in Section 3.2.

3.1.2 Graph models

As there are numerous models for graph structures to be found in literature such as [8, Section 1.5, Section B.2], the simulations conducted for this thesis were confined to the distinct graph models covered below.

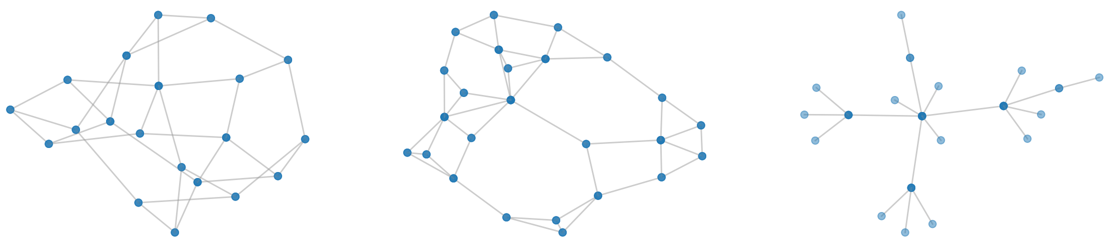


Figure 3.1: Example realisations of some graph models: (a) perturbed random regular graph, (b) random geometric graph, (c) Barabási--Albert graph. The node colour indicates the degree $\deg(m)$.

Perturbed random regular graphs. Random regular graphs feature unweighted edges that are randomly drawn until each node is connected to exactly a fixed number of other nodes [8, Section B.2.2]. In order to introduce a controllable amount of irregularity, we perturb a certain percentage of edges by either excluding an existing edge or adding a non-existing one. An example is depicted in Fig. 3.1a.

Random geometric graphs. For random geometric graphs, nodes are spatially uniformly distributed. Using the Euclidean distance, each node is then connected to a fixed number of nearest neighbours [8, Section B.2.3]. In case of a weighted graph, w can be expressed in terms of a Gaussian kernel of the Euclidean distance. This is the model typically used for sensor networks. An example is shown in Fig. 3.1b.

Barabási–Albert graphs. Unlike the two previously described graph models, this preferential attachment graph model [8, Section 1.5] yields graphs with power-law degree distributions, i.e., most nodes have a small degree while very few so-called *hubs* have lots of connected neighbours. This is achieved by connecting new nodes to existing nodes of a graph. The concept that the probability of a new node being connected to an existing node depends on the existing node's degree results in social-network-like graph with the aforementioned degree distribution as illustrated in Fig. 3.1c.

3.2 Graph signal processing

In order to extend signal processing to potentially irregular domains, a graph signal $x : \mathcal{V} \to \mathbb{R}$ is defined that maps each node n of a graph $\mathcal{G}$ to a signal value x_n. This thesis solely deals with scalar signal values, collected into a vector $\mathbf{x} = (x_1, \ldots, x_N)^T$.

3.2.1 Graph Fourier transform

The basis for the graph Fourier transformation (GFT) is often defined in literature as the eigenvectors $\mathbf{u}_k$ of the Laplacian matrix[1] $\mathbf{L}$, with its (sorted) eigenvalues being the graph frequencies $0 = \lambda_1 \leq \ldots \leq \lambda_N$ [5]. As $\mathbf{L}$ is positive semi-definite, the $\mathbf{u}_k$ form an orthonormal basis and the λ_k are non-negative as indicated.

As mentioned in Section 3.1.1, many similarities to the Laplace operator Δ exist. Most notably in the context of graph signal processing is that the complex sinusoids $e^{-j\omega t}$, which form the basis functions of the time-continuous Fourier transform, are the eigenfunctions of Δ, while the eigenvalues are a function of the frequency ω [9], whence

$$\Delta e^{-j\omega t} = -\omega^2 e^{-j\omega t}.$$

Another intuition for the GFT uses the graph gradient, defined as

$$\left(\nabla_{\mathcal{G}} \mathbf{x}\right)_{mn} := \sqrt{W_{mn}} \left(x_m - x_n\right).$$

Since a sparse gradient matrix indicates a piecewise constant and thus smooth signal, we can use its Euclidean norm $\frac{1}{2} \left\| \nabla_{\mathcal{G}} \mathbf{x} \right\|_2^2$ as smoothness metric. As shown in [8, eq. (3.8)], it holds that $\frac{1}{2} \left\| \nabla_{\mathcal{G}} \mathbf{x} \right\|_2^2 = \mathbf{x}^T \mathbf{L} \mathbf{x}$; normalised to the signal power, we obtain

$$\mathcal{T}_2\left(\mathbf{x}\right) := \frac{\mathbf{x}^T \mathbf{L} \mathbf{x}}{\mathbf{x}^T \mathbf{x}}.$$

This expression is known as the Rayleigh quotient, which is minimised by $\mathbf{x} = \mathbf{u}_k$ if we seek orthogonality to the other $N-1$ basis vectors. As this results in

$$\mathcal{T}_2\left(\mathbf{u}_k\right) = \lambda_k, \qquad k = 1, \ldots, N$$

with $0 = \lambda_1 \leq \ldots \leq \lambda_N$, we observe that the eigenvectors can be considered less smooth at higher eigenvalues (thus called *frequencies*) λ_k. Note that the smallest eigenvalue is $\lambda_1 = 0$, since $\mathbf{L1} = \mathbf{D1} - \mathbf{W1} = \mathbf{0}$. By analogy with classical signal

[1] Another basis would be induced e.g. by the normalised graph Laplacian $\tilde{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}} \mathbf{L} \mathbf{D}^{-\frac{1}{2}}$.

processing, $\mathbf{u}_1 = \mathbf{1}$ can be interpreted as the DC component at frequency $\lambda_1 = 0$ [8, Section 3.2.3].

We can define the GFT of a graph signal $\mathbf{x}$ as

$$\mathfrak{x} := \mathbf{U}^T \mathbf{x},$$

with $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_N)$ and the GFT coefficients $\mathfrak{x} = (\mathfrak{x}_1, \ldots, \mathfrak{x}_N)^T$. The GFT is a linear transform with complexity $\mathcal{O}(N^2)$, unless $\mathbf{U}$ has a facilitating structure. Since $\mathbf{U}$ is an orthonormal matrix, Parseval's relation also holds in the GFT domain, i.e.,

$$\mathbf{x}^T \mathbf{y} = \mathfrak{x}^T \mathfrak{y},$$

where $\mathfrak{x}$ and $\mathfrak{y}$ are the GFT of $\mathbf{x}$ and $\mathbf{y}$, respectively. In order to highlight structural differences between graph models, we can compare the maximal absolute value of their GFT matrices $\mathbf{U}$, i.e., $\mu = \max\limits_{i,k} |(\mathbf{U})_{ik}| \in \left[\frac{1}{\sqrt{N}}, 1\right]$. This metric is called coherence and indicates the level of localization of the Laplacian eigenvectors [5, Section 3.2]. The lower bound occurs in the unweighted ring graph (i.e., a periodic discrete-time signal), where the eigenvectors are globally oscillating vectors. Higher coherences can be observed when the degree of some nodes strongly deviate from the graphs average degree. In Table 3.1, we listed the range of μ of the graphs that we described in Section 3.1.2 and simulated with $N = 64$ in Chapter 5. On a final

Graph model	Regular	Random geometric	Barabási–Albert
Common μ range	$[0.45, 0.75]$	$[0.7, 0.85]$	$[0.8, 0.96]$

Table 3.1: Range of the coherence μ of the graphs simulated in Chapter 5 with $N = 64$.

note, the smoothness metric can be rewritten as $\mathcal{T}_2(\mathbf{x}) = \sum_k \lambda_k \mathfrak{x}_k^2 = \mathfrak{x}^T \mathrm{Diag}(\boldsymbol{\lambda})\, \mathfrak{x}$ with $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_N)^T$ [8, eq. (3.12)]. Thus, for $\mathcal{T}_2(\mathbf{x})$ to be small (or equivalently the signal to be smooth), the signal should feature large GFT coefficients $\mathfrak{x}_k$ only at small frequencies λ_k.

3.2.2 Basic graph signal processing operations

Using a window function that is shifted in time and frequency, windowed Fourier transforms (WFT) can extract localised information from a signal. Shuman et al.

generalised the WFT to graphs by defining both translation (i.e., a time shift) and modulation (i.e., a frequency shift) in a graph setting. Thus, this section is based on their work [5], where they also present a plethora of related properties.

Generalised convolution on graphs. In classical signal processing (i.e., one-dimensional on a regular, continuous domain), convolution is expressed as

$$(x * y)(t) \coloneqq \int_{\mathbb{R}} x(\tau)\, y(t-\tau)\, d\tau = \int_{\mathbb{R}} \mathfrak{x}(j\omega)\, \mathfrak{y}(j\omega)\, e^{j\omega t} d\omega,$$

where $\mathfrak{x}(j\omega) = (\mathcal{F}x)(j\omega)$ and $\mathcal{F}$ denotes Fourier transform. The rightmost expression is suitable for a reformulation of convolution for graph signals using the GFT definition of Section 3.2.1 that reads

$$\mathbf{x} * \mathbf{y} \coloneqq \sum_{l=1}^{N} \mathfrak{x}_l \mathfrak{y}_l \mathbf{u}_l = \mathbf{U}\left(\boldsymbol{\mathfrak{x}} \odot \boldsymbol{\mathfrak{y}}\right),$$

where $\odot$ is the Hadamard product (i.e., component-wise multiplication).

Generalised translation on graphs. A fundamental building block of signal processing is translation. A time shift u of a function $x(t)$ in classical signal processing is defined by

$$\left(\tilde{\mathbb{T}}_u x\right)(t) \coloneqq x(t-u).$$

Since there no intuitive direction in which a signal on a potentially multiply connected graph should be shifted, we can make use of an alternative formulation of translation in classical signal processing, using convolution:

$$\left(\tilde{\mathbb{T}}_u x\right)(t) \coloneqq x(t-u) = (x * \delta_u) = \int_{\mathbb{R}} \mathfrak{x}(j\omega)\, (\mathcal{F}\delta_u)(j\omega)\, e^{j\omega t} d\omega = \int_{\mathbb{R}} \mathfrak{x}(j\omega)\, e^{-j\omega u} e^{j\omega t} d\omega$$

with $\delta(t-u)$ being the Dirac delta function shifted by u.

Using the previously introduced generalised convolution, we can generalize translation for graph signals as follows

$$\mathbb{T}_i\mathbf{x} \coloneqq (\mathbf{x} * \boldsymbol{\delta}_i) = \sqrt{N}\sum_{l=1}^{N} \mathfrak{x}_l \left(\mathbf{u}_l^*\right)_i \mathbf{u}_l = \sqrt{N}\mathbf{U}\left(\operatorname{Diag}\left(\tilde{\mathbf{u}}_i\right)\mathfrak{x}\right) \tag{3.1}$$
$$= \sqrt{N}\mathbf{U}\left(\tilde{\mathbf{u}}_i \odot \mathfrak{x}\right),$$

where $\boldsymbol{\delta}_i$ is the Kronecker delta

$$\left(\boldsymbol{\delta}_i\right)_n = \begin{cases} 1, & i = n \\ 0, & \text{otherwise} \end{cases},$$

and $\tilde{\mathbf{u}}_i$ is the i^{th} row of $\mathbf{U}$. Unlike $\tilde{\mathbb{T}}_u$, however, the generalised translation operator is not an isometric operator, so it is not energy-preserving,

$$\left\|\mathbb{T}_i\mathbf{x}\right\|_2^2 \neq \left\|\mathbf{x}\right\|_2^2.$$

Another inconvenience is that $\mathbb{T}_i$ does not have group structure [5, Section 4.3], i.e.,

$$\mathbb{T}_i\mathbb{T}_j \neq \mathbb{T}_{i+j}.$$

Generalised modulation on graphs. While the generalisations of convolution and translation had to be reformulated in the frequency domain, we can directly use the time-domain description of modulation in classical signal processing, i.e.,

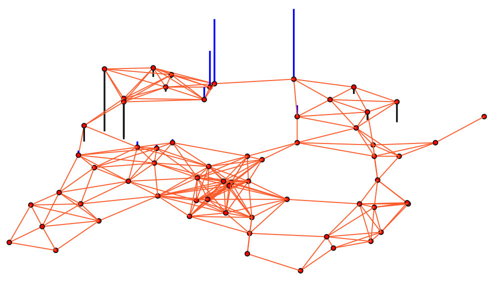
$$\left(\tilde{\mathbb{M}}_\xi x\right)(t) \coloneqq x(t)\, e^{j\xi t}.$$

For graphs, modulation is generalised as

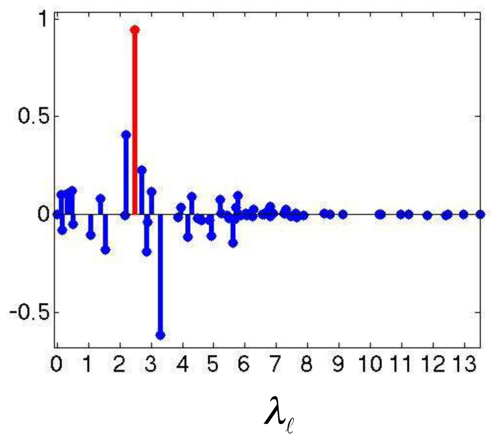
$$\mathbb{M}_k\mathbf{x} \coloneqq \sqrt{N}\operatorname{Diag}\left(\mathbf{u}_k\right)\mathbf{x} = \sqrt{N}\mathbf{u}_k \odot \mathbf{x} \tag{3.2}$$

Like the generalised translation operator $\mathbb{T}_i$, $\mathbb{M}_k$ is neither an isometric operator [5, Section 5.1], nor does it have group structure, since

$$\mathbb{M}_k\mathbb{M}_l\mathbf{x} = N\mathbf{u}_k \odot \mathbf{u}_l \odot \mathbf{x} \neq \sqrt{N}\mathbf{u}_{k+l} \odot \mathbf{x} = \mathbb{M}_{k+l}\mathbf{x}.$$



(a) WFT atom in the vertex domain. Figure taken from [5, Fig. 11(b)]



(b) WFT atom in the frequency domain. Figure taken from [5, Fig. 11(c)]

Figure 3.2: An exponential WFT atom $\mathbf{g}_{27,11}$, centred at vertex 27 and frequency $\lambda_{11} = 2.49$ on the Minnesota road graph with $N = 64$, $\mathfrak{g}_l = Ce^{-\tau\lambda_l}$.

3.2.3 Windowed graph Fourier transform

The classical windowed Fourier atom

$$g_{\tau\xi}\left(t\right) := \left(\tilde{\mathbb{M}}_{\xi}\tilde{\mathbb{T}}_{\tau}g\right)\left(t\right) = g\left(t-\tau\right)e^{j\xi t} \tag{3.3}$$

allows us to perform the WFT in classical signal processing,

$$\mathfrak{x}_g\left(\tau,\xi\right) := \left\langle x, g_{\tau\xi}\right\rangle .$$

Using the definitions introduced in (3.1) and (3.2), the definition of the windowed Fourier atom was generalised in [5] as

$$\mathbf{g}_{ik} := \mathbb{M}_k\mathbb{T}_i\mathbf{g} = N\text{Diag}\left(\mathbf{u}_k\right)\left(\mathbf{U}\left(\tilde{\mathbf{u}}_i \odot \mathfrak{g}\right)\right). \tag{3.4}$$

A depiction of a windowed graph Fourier transform (WGFT) atom on the Minnesota road graph [10] is shown in Fig. 3.2. We can arrange the windowed Fourier atoms in a matrix as follows

$$\mathbf{G} = \left(\mathbf{g}_{11},\dots,\mathbf{g}_{1N},\dots,\mathbf{g}_{N1},\dots,\mathbf{g}_{NN}\right) \in \mathbb{R}^{N\times N^2}.$$

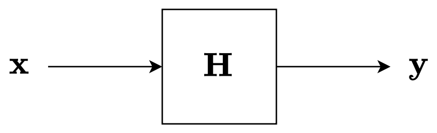


Figure 3.3: Block diagram of a linear filter.

This allows for a concise reformulation of the WFT to

$$\mathfrak{x}_{\mathbf{g}11} \coloneqq \mathbf{g}_{11}^T \mathbf{x}.$$

We can express the GFT coefficients as a vector that reads

$$\mathfrak{x}_{\mathbf{g}} \coloneqq \mathbf{G}^T \mathbf{x} \in \mathbb{R}^{N^2},$$

with $\mathfrak{x}_{\mathbf{g}} = (\mathfrak{x}_{\mathbf{g}11}, \dots, \mathfrak{x}_{\mathbf{g}1N}, \dots, \mathfrak{x}_{\mathbf{g}N1}, \dots, \mathfrak{x}_{\mathbf{g}NN})^T \in \mathbb{R}^{N^2}$.

The reconstruction of the graph signal is then given by [5, Theorem 4]

$$\mathbf{x} = \mathrm{Diag}\left(\frac{1}{N \|\mathbb{T}_1 \mathbf{g}\|_2^2}, \dots, \frac{1}{N \|\mathbb{T}_N \mathbf{g}\|_2^2}\right) \mathbf{G} \mathfrak{x}_{\mathbf{g}}. \tag{3.5}$$

3.2.4 Laplacian polynomial filters

Since we denote graph signals as vectors, a linear filter (i.e., a linear system) acting on graph signals amounts to a matrix-multiplication,

$$\mathbf{y} = \mathbf{H}\mathbf{x}.$$

Here, $\mathbf{x}$ is the input signal and $\mathbf{y}$ is the output signal. The system is depicted in Fig. 3.3. There are numerous classes of graph filters with a certain structure (e.g., GFT multipliers). This thesis solely focuses on Laplacian polynomial filters as the algorithms in [3] that are discussed later avail themselves of this concept. This introduction is based on [11].

As indicated by its name, a Laplacian polynomial filter is a graph filter that is described as a polynomial in $\mathbf{L}$, i.e.,

$$\mathbf{H} = \sum_{p=0}^{P} \eta_p \mathbf{L}^p = \mathbf{U} \sum_{p=0}^{P} \eta_p \mathrm{Diag}\left(\boldsymbol{\lambda}\right)^p \mathbf{U}^T = \mathbf{U}\mathrm{Diag}\left(\mathbf{h}\right) \mathbf{U}^T, \tag{3.6}$$

where we used the eigendecomposition $\mathbf{L} = \mathbf{U}\mathrm{Diag}\left(\boldsymbol{\lambda}\right)\mathbf{U}^T$ in the second step and defined a filter transfer function as $\mathbf{h} = \left(h_1, \ldots, h_N\right)^T = \sum_{p=0}^{P} \eta_p \boldsymbol{\lambda}^p$. In the simplest example of $P = 1$, $\eta_0 = 0$, $\eta_1 = 1$, we have $\mathbf{H} = \mathbf{L}$. Since

$$L_{mn} = \begin{cases} d\left(n\right), & m = n \\ W_{mn}, & \{m,n\} \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases},$$

the output signal at node n results in

$$y_n = \sum_{m=1}^{N} L_{mn} x_m = d\left(n\right) x_n + \sum_{m:\{m,n\}\in\mathcal{E}} W_{mn} x_m.$$

Hence, only the signal at node n and at its direct neighbours is needed to compute y_n. For a filter order P, only the P-hop neighbourhood of node n is considered for the calculation of y_n, which makes this filter structure efficient both in terms of storage and computation time.

3.3 Bandlimited signal reconstruction

Perfect reconstruction of a graph signal $\mathbf{x} \in \mathbb{R}^N$ from a set of samples $\mathbf{x}_\mathcal{S} = \left(x_{n_1}, \ldots, x_{n_M}\right)^T \in \mathbb{R}^M$, $\mathcal{S} = \{n_1, \ldots, n_M\} \subset \mathcal{V}$, $M < N$ is impossible without further knowledge about the signal structure: If $\mathbf{x}$ has N degrees of freedom, recovering it with M coefficients is not feasible. By making additional assumptions, however, reconstruction becomes possible under certain conditions.

One way of limiting the degrees of freedom is to require a lowpass signal whose first K GFT coefficients $\mathfrak{x}_K = \left(\mathfrak{x}_1, \ldots, \mathfrak{x}_K\right)^T$ are non-zero and $\mathfrak{x}_k = 0$ for $k > K$. This case is discussed in [8, Section 4.3.4]. In this case, the inverse graph Fourier transform (IGFT) can be expressed by $\mathbf{x} = \mathbf{U}_K \mathfrak{x}_K$, with $\mathbf{U}_K = \left(\mathbf{u}_1, \ldots, \mathbf{u}_K\right)^T$ being the first K eigenvectors of $\mathbf{L}$. The sampled signal is then given by

$$\mathbf{x}_\mathcal{S} = \mathbf{U}_{K,\mathcal{S}} \mathfrak{x}_K = \mathbf{S}_\mathcal{S} \mathbf{U}_K \mathfrak{x}_K.$$

where $\mathbf{S}_\mathcal{S}$ is a row-selection matrix used for sampling. Consequently, $\mathfrak{x}_K$ (and thus $\mathbf{x}$ using the IGFT) can be recovered using $\mathbf{U}_{K,\mathcal{S}}^{\#} = \left(\mathbf{U}_{K,\mathcal{S}}^T \mathbf{U}_{K,\mathcal{S}}\right)^{-1} \mathbf{U}_{K,\mathcal{S}}^T$, the pseudoinverse of $\mathbf{U}_{K,\mathcal{S}}$, as follows

$$\begin{aligned}\mathfrak{x}_K &= \mathbf{U}_{K,\mathcal{S}}^{\#}\mathbf{x}_{\mathcal{S}}, \\ \mathbf{x} &= \mathbf{U}_K\mathbf{U}_{K,\mathcal{S}}^{\#}\mathbf{x}_{\mathcal{S}},\end{aligned} \tag{3.7}$$

assuming that $\mathbf{U}_{K,\mathcal{S}}$ has full column rank K. This essentially requires a sufficiently dense sampling set $\mathcal{S}$ (for which a necessary condition is $M \geq K$).

3.4 Nonstationary processes

So far in this chapter, we have covered fundamentals of graph topology and graph signal processing. In the final section, we characterise the graph signals that we later quantise.

We can characterise stationary processes with their power spectral density (PSD) $\mathbf{p}_{\mathfrak{x}}$. For general signals, we instead use the covariance matrix $\mathbf{C}_{\mathsf{x}} = \mathrm{E}\left\{\mathbf{x}\mathbf{x}^T\right\}$. In order to describe the signal in the frequency domain, we perform a GFT on $\mathbf{C}_{\mathsf{x}}$ which (in the graph notation) reads

$$\mathbf{C}_{\mathfrak{x}} = \mathbf{U}^T\mathbf{C}_{\mathsf{x}}\mathbf{U}.$$

In case of a stationary process, the PSD $\mathbf{p}_{\mathfrak{x}}$ [12, eq. (9-133)] reappears in the frequency covariance in the form of

$$\begin{aligned}\mathbf{C}_{\mathfrak{x}} &= \mathrm{Diag}\left(\mathbf{p}_{\mathfrak{x}}\right), \\ \mathbf{C}_{\mathsf{x}} &= \mathbf{U}\mathbf{C}_{\mathfrak{x}}\mathbf{U}^T = \mathbf{U}\mathrm{Diag}\left(\mathbf{p}_{\mathfrak{x}}\right)\mathbf{U}^T.\end{aligned}$$

As explained in Section 2.2, for source coding we are interested in a transform that yields a covariance matrix with very small off-diagonal elements. We measure the level of correlation in a domain with the correlation coefficient

$$\psi_{\mathsf{x}_m\mathsf{x}_n} = \frac{\mathrm{E}\left\{\mathsf{x}_m\mathsf{x}_n\right\}}{\sqrt{\mathrm{E}\left\{\mathsf{x}_m^2\right\}\mathrm{E}\left\{\mathsf{x}_n^2\right\}}} \in [-1, 1],$$

by computing the percentage of $|\psi_{\mathsf{x}_m\mathsf{x}_n}|, m \neq n$ that are smaller than a certain

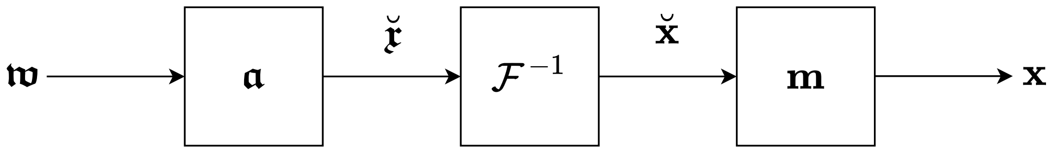


Figure 3.4: Generation of nonstationary process by filtering white noise.

threshold $\theta > 0$. Mathematically, we express this percentage as follows

$$b\left(\mathbf{x}, \theta\right) = \frac{1}{N\left(N-1\right)} \sum_{m \neq n} \mathrm{I}\left\{\left|\psi_{\mathsf{x}_m \mathsf{x}_n}\right| < \theta\right\}, \tag{3.8}$$

with $\mathrm{I}\left\{\cdot\right\}$ denoting the indicator function. Since $\mathbf{C}_{\mathfrak{x}}$ is a diagonal matrix for stationary processes, they are already best decorrelated in the frequency domain ($b\left(\mathfrak{x}, \theta\right) = 1, \forall \theta > 0$).

We now focus on (slowly) nonstationary processes, however, as the preferred transform choice becomes less obvious. Unlike in classical signal processing [4], it is difficult to measure the correlation width on graphs in the vertex domain, denoted τ_0. A quantity that captures the level of vertex-domain correlation is

$$k_{\mathrm{rms}} = \sqrt{\frac{\sum_k c_{\mathfrak{x}kk} k^2}{\sum_k c_{\mathfrak{x}kk}}} \in \left[1, \sqrt{\frac{\left(N+1\right)\left(2N+1\right)}{6}}\right]. \tag{3.9}$$

Large values for k_{rms} show similar power over a large bandwidth and therefore an uncorrelated signal in the vertex domain. Conversely, small values for k_{rms} indicate a steep decline in power over frequency and thus correlated vertex-domain signal values. Of note is that a frequency bandwidth $K < N$ already causes vertex-domain correlation since $\left(c_{\mathfrak{x}K+1\,K+1}, \ldots, c_{\mathfrak{x}NN}\right) = \mathbf{0}$ prohibits k_{rms} to reach its maximum level.

This section covers two methods that yield nonstationary processes and are thus applied in the simulations in Chapter 5.

3.4.1 Class I processes: filtering and modulation

Our first approach for creating a nonstationary process is to filter white noise in the respective domain as depicted in the block diagram in Fig. 3.4. White noise $\mathfrak{w}$ with the covariance matrix $\mathbf{C}_{\mathfrak{w}} = \mathbf{I}_N$ is linearly transformed with a filter $\mathfrak{a}$ in the GFT domain, i.e.,

$$\breve{\mathfrak{x}} = \mathrm{Diag}\left(\mathfrak{a}\right)\mathfrak{w} = \mathfrak{a} \odot \mathfrak{w}.$$

$\mathfrak{a}$ introduces correlation in the vertex domain by weighting the frequencies. In the extreme case of

$$(\mathfrak{a})_k = \begin{cases} 1, & k = 1, \\ 0, & \text{otherwise.} \end{cases}$$

only the DC component of the signal remains, which we can also interpret as $\breve{\mathbf{x}} = \mathbf{U}\breve{\mathfrak{x}}$ being fully correlated signal. The counterpart $\mathfrak{a} = \mathbf{1}$, on the other hand, does not weight the frequencies at all and thus leaves the signal in the vertex domain white, i.e., uncorrelated[2]. We can quantify the level of vertex-domain correlation in $\breve{\mathbf{x}}$ using k_{rms} as defined in (3.9) as it shows how sparse $\mathfrak{a}$ is.

After an IGFT, we modulate the (potentially correlated) vertex-domain signal with a signal vector $\mathbf{m}$ that destroys stationarity and thus induces correlation in the frequency domain, i.e.,

$$\mathbf{x} = \mathbf{m} \odot \breve{\mathbf{x}} = \mathrm{Diag}\left(\mathbf{m}\right)\breve{\mathbf{x}} = \mathrm{Diag}\left(\mathbf{m}\right)\mathbf{U}\mathrm{Diag}\left(\mathfrak{a}\right)\mathfrak{w} = \mathbf{m} \odot \mathbf{U}\left(\mathfrak{a} \odot \mathfrak{w}\right).$$

The covariance matrix of the output $\mathbf{x}$ consequently reads

$$\mathbf{C}_{\mathbf{x}} = \mathrm{Diag}\left(\mathbf{m}\right)\underbrace{\mathbf{U}\mathrm{Diag}\left(\mathfrak{a}\right)^2\mathbf{U}^T}_{\mathbf{C}_{\breve{\mathbf{x}}}}\mathrm{Diag}\left(\mathbf{m}\right).$$

As a byproduct of a high-frequency $\mathbf{m}$, the nominal bandwidth K that was set using $\mathfrak{a}$ might be increased.

In case of $\mathbf{m} = \mathbf{1} = (1, 1, \ldots)^T$, the signal remains stationary and thus uncorrelated in the frequency domain. Conversely, we expected high intensity throughout the entire spectrum of $\mathbf{m}$ to lead to maximum frequency correlation.

This is due to the fact that in classical signal processing, a constant spectrum would

[2]W.l.o.g., we can choose any constant value $a \in \mathbb{R}_*^+$ for both filters.

yield an impulse in the time domain. More specifically, with

$$(\mathbf{m})_i = (\boldsymbol{\delta}_1)_i = \begin{cases} 1, & i = 1, \\ 0, & \text{otherwise,} \end{cases}$$

the discrete Fourier transform (DFT) of $\mathbf{x}$ reads

$$\mathfrak{x} = \mathbf{U}^H \mathbf{x} = \mathbf{U}^H \mathrm{Diag}\left(\mathbf{m}\right) \breve{\mathbf{x}} = \mathbf{U}^H \begin{pmatrix} \breve{x}_1 \\ 0 \\ \vdots \end{pmatrix} = \tilde{\mathbf{u}}_1^* \breve{x}_1,$$

where $\mathbf{U}$ is the inverse discrete Fourier transform (IDFT) matrix and $\tilde{\mathbf{u}}_1^*$ is the complex conjugate of its first row. Then, the covariance matrix is fully correlated in the frequency domain, i.e.,

$$\mathbf{C}_{\mathfrak{x}} = c_{\breve{x}11} \tilde{\mathbf{u}}_1^* \left(\tilde{\mathbf{u}}_1^*\right)^T = c_{\breve{x}11} \mathbf{1},$$

since the first row of the IDFT matrix is constant.

However, as shown in [13], the GFT for an ordinary 1D signal is instead equivalent to the discrete cosine transform (DCT). The first row of the inverse discrete cosine transform (IDCT) matrix is not constant and a constant spectrum does not yield a Kronecker delta in the time domain.

Thus, we do not achieve full correlation in the frequency domain, even less so for general graphs. We can still generate signals using different choices for k_{rms} and the spectrum of $\mathbf{m}$, but we cannot expect a simple relation between the latter and the frequency-domain correlation as it is the case for the ordinary Fourier transform.

3.4.2 Class II processes: banded frequency correlation matrix

Starting from a diagonal frequency covariance matrix $\mathbf{C}_{\mathfrak{x}}$ of a stationary process, we can introduce nonstationarity by symmetrically adding diagonals on either side of the main diagonal, i.e.,

$$\mathbf{C}_{\mathfrak{r}} = \begin{pmatrix} c_{\mathfrak{r}11} & \cdots & c_{\mathfrak{r}1B} & & & \mathbf{0} \\ \vdots & \ddots & & \ddots & & \\ c_{\mathfrak{r}B1} & & \ddots & & \ddots & \\ & \ddots & & \ddots & & c_{\mathfrak{r}(N-B)N} \\ & & \ddots & & \ddots & \vdots \\ \mathbf{0} & & & c_{\mathfrak{r}N(N-B)} & \cdots & c_{\mathfrak{r}NN} \end{pmatrix}. \tag{3.10}$$

If just one diagonal on either side has non-zero entries, the matrix is referred to as tridiagonal matrix; a tridiagonal matrix indicates correlation between adjacent frequencies. In case of B super- and subdiagonals, a frequency is correlated with the neighbouring B frequencies.

In order to generate vertex-domain correlation, i.e., to lower k_{rms}, the diagonal entries of $\mathbf{C}_{\mathfrak{r}}$ are varied. Since the power of real-life signals is mostly concentrated at lower frequencies, we consider the entries along a diagonal to be monotonically decreasing.

According to the correlation in the vertex domain, we need to carefully select the parameters steering the correlation in the frequency domain. The first of these is B, i.e., the number of super- and subdiagonals. In order to increasingly lower the level of correlation to frequencies further away, we introduce another parameter $\zeta \in [0, 1]$ that weights the super- and subdiagonals as follows

$$\left(c_{\mathfrak{r}b1}, \ldots, c_{\mathfrak{r}N(N-b)}\right) = \left(c_{\mathfrak{r}1b}, \ldots, c_{\mathfrak{r}(N-b)N}\right) = \zeta^b \left(c_{\mathfrak{r}12}, \ldots, c_{\mathfrak{r}(N-1-b)(N-b)}\right), \quad b = 1, \ldots, B.$$

Band matrices are not inherently positive semidefinite (i.e., $\mathbf{z}^T\mathbf{K}\mathbf{z} \geq 0$) which is a prerequisite for $\mathbf{C}_{\mathfrak{r}}$ to be a valid covariance matrix. We can show, however, that diagonally dominant matrices with real positive diagonal entries, i.e., $\mathbf{K}$, where $K_{ii} \geq \sum_{j\neq i} |K_{ij}|, \forall i$, are positive semidefinite,

$$\begin{aligned} \mathbf{z}^T\mathbf{K}\mathbf{z} &= \sum_{i=1}^{N} K_{ii} z_i^2 + \sum_{j\neq i} K_{ij} z_j z_i \geq \sum_{i=1}^{N} \sum_{j\neq i} |K_{ij}|\, z_i^2 - \sum_{j\neq i} |K_{ij}|\, |z_j|\, |z_i| \\ &= \sum_{j>i} \left(|K_{ij}| \left(z_i^2 + z_j^2 - 2\, |z_j|\, |z_i| \right) \right) = \sum_{j>i} \left(|K_{ij}| \left(|z_j| - |z_i| \right)^2 \right) \geq 0. \end{aligned}$$

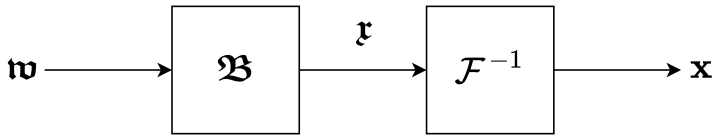


Figure 3.5: Generation of nonstationary process with innovations system $\mathfrak{B}$.

We therefore create $\mathbf{C}_{\mathfrak{x}}$ by first generating super- and subdiagonals[3] whose entries we then summed up to get the main diagonal. We also added a number $\varepsilon_i > 0$ to not get a critically diagonally dominant matrix, i.e.,

$$K_{ii} = \sum_{j \neq i} |K_{ij}| + \varepsilon_i, \qquad \mathbf{C}_{\mathfrak{x}} = \mathbf{K}.$$

Another way to create a process with banded frequency correlation matrix is to first design an upper band matrix $\mathfrak{B}$ as illustrated in Fig. 3.5, where only the main diagonal and B superdiagonals have non-zero values[4]. If we then filter white noise in the frequency domain $\mathfrak{w}$ with the covariance matrix $\mathbf{C}_{\mathfrak{w}} = \mathbf{I}_N$ using the innovations system $\mathfrak{B}$, we obtain the following covariance matrix

$$\mathbf{C}_{\mathfrak{x}} = \mathfrak{B}\mathbf{I}_N\mathfrak{B}^T = \mathfrak{B}\mathfrak{B}^T.$$

This matrix has the same band matrix structure as the matrix in (3.10) and is positive semidefinite, since

$$\mathbf{z}^T\mathfrak{B}\mathfrak{B}^T\mathbf{z} = \left(\mathfrak{B}^T\mathbf{z}\right)^T\left(\mathfrak{B}^T\mathbf{z}\right) = \left\|\mathfrak{B}^T\mathbf{z}\right\|_2^2 \geq 0.$$

[3]We gave more weight to diagonals closer to the main diagonal to reduce correlation with more distant frequencies.

[4]The respective diagonals entries are still considered to be monotonically decreasing.

4 Compression techniques

In this chapter we discuss various compression techniques for signals on graphs. They vary in bit allocation and the sampling domain, i.e., the linear filter acting on the vertex-domain signal prior to sampling and compression. While transformation into the frequency-domain includes the GFT matrix as defined in Section 3.2.1, optimisation is necessary to find the sampling filter for the techniques described in Section 4.2 and Section 4.4[1].

The goal of sampling in a certain domain is to approximately decorrelate the signal without applying the computationally expensive Karhunen-Loève transform (as explained in Section 2.2). Another property that can serve as indicator for good compressibility in the respective domain is the level of sparsity of the monotonically decreasing diagonal entries $c_{\mathfrak{x}_0 kk}$ of the covariance matrix, i.e.,

$$k_{\text{eff}} = \sqrt{\frac{\sum_k c_{\mathfrak{x}_0 kk} k^2}{\sum_k c_{\mathfrak{x}_0 kk}}}.$$

High values for k_{eff} show that many samples carry similar power and thus need to be sampled. For signals with low k_{eff} values, it is sufficient to sample at fewer positions (at a higher bit rate per sampled coefficient). Note that the Karhunen-Loève transform (see Section 2.2) would yield the lowest k_{eff}. Similar to what we outlined for $\mathcal{T}_2(\mathbf{x})$ and the GFT matrix in Section 3.2.1, the Rayleigh coefficient of the covariance matrix is continuously maximised by the eigenvector corresponding to the next-largest eigenvalue. As the next-largest eigenvalue is also the result of the maximisation, we observe that using the eigenvalues as $c_{\mathfrak{x}_0 kk}$ (as done by the Karhunen-Loève transform) concentrates the maximum possible amount of power in the first coefficients.

While the procedures for sampling (and thus interpolation) differ, the quantisation

[1]The level of complexity of the two optimisation problems differs vastly, however, as explained presently.

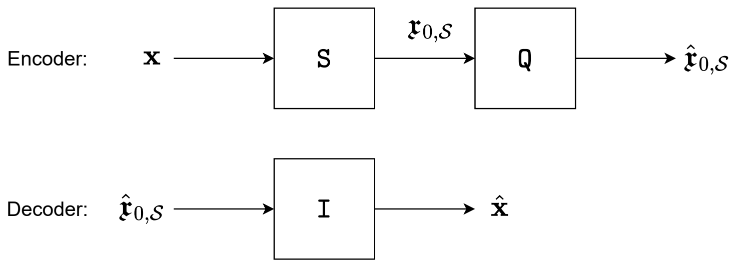

Encoder:
$\mathbf{x}$
S
$\mathfrak{x}_{0,\mathcal{S}}$
Q
$\hat{\mathfrak{x}}_{0,\mathcal{S}}$
Decoder:
$\hat{\mathfrak{x}}_{0,\mathcal{S}}$
I
$\hat{\mathbf{x}}$

Figure 4.1: Block diagram of the source coding techniques (i.e., sampling `S`, quantisation `Q`, interpolation `I`) applied in this thesis.

for a given bit allocation is always done using a uniform quantiser as described in Section 2.3. As mentioned, the bit allocation techniques differ, however, the bit budget, denoted as R, remains the same for comparability. A schematic overview of the compression system is shown in Fig. 4.1.

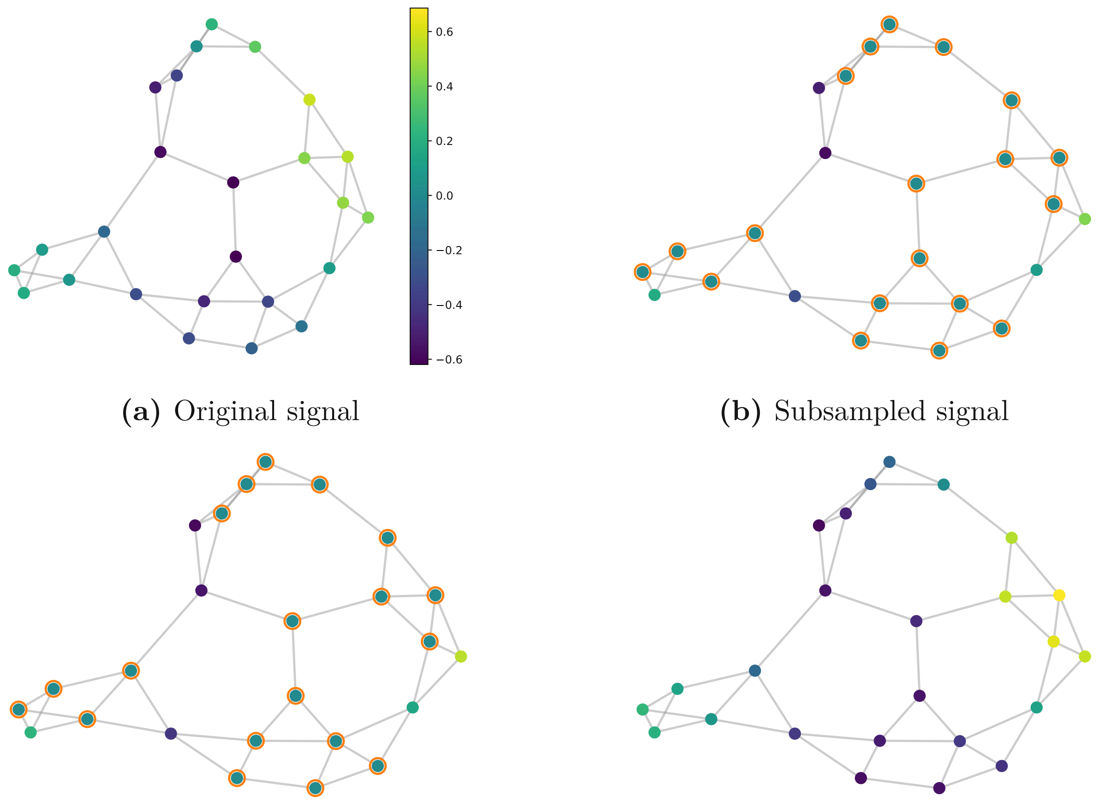
The techniques presented in this thesis comprise vertex-domain compression in Section 4.1, joint quantisation and sampling as introduced by [3] in Section 4.2, frequency-domain compression in Section 4.3, and compression using the Gabor transform for graphs in Section 4.4.

4.1 Vertex-domain compression

A straightforward way of compression is to sample a signal (as shown in Fig. 4.2a) at M positions in the vertex domain (see Fig. 4.2b), such that $\operatorname{rank}(\mathbf{U}_{K,\mathcal{S}}) = K$. Since all nodes in perturbed random regular graphs and random geometric graphs on average have a similar number of neighbours, for these graph models we chose a uniform probability distribution for a node m to be included in the sampling set $\mathcal{S}$, i.e.,

$$\mathrm{P}\{m \in \mathcal{S}\} = \frac{M}{N}, \qquad \sum_{m=1}^{N} \mathrm{P}\{m \in \mathcal{S}\} = M.$$

In a Barabási–Albert graph, hubs contain more valuable information due to their high degree. Thus, we set the probability of $m \in \mathcal{S}$ proportional to $\deg(m)$. Since an edge $e = \{m, n\}$ increases both $\deg(m)$ and $\deg(n)$ by one, the sum of all degrees



(a) Original signal **(b)** Subsampled signal

(c) Quantised subsampled signal **(d)** Recovered signal

Figure 4.2: Vertex-domain compression of a signal with $K = \frac{N}{5}$. The node colour indicates the signal value. Unsampled signal values are highlighted.

reads $\sum_{v \in \mathcal{S}} \deg(v) = 2|\mathcal{E}|$. We thus have

$$\mathrm{P}\{m \in \mathcal{S}\} = \frac{\deg(m)}{2|\mathcal{E}|}.$$

If the assumption that $\mathbf{x}$ is bandlimited held true and $\operatorname{rank}(\mathbf{U}_{K,\mathcal{S}}) = K$, perfect reconstruction in the absence of quantisation would be possible, as explained in Section 3.3.

This is, however, no longer the case, when each sample gets distorted due to quantisation at a finite rate, as indicated in Fig. 4.2c, or with the process generation as described in Section 3.4.1. In this naïve reference technique, each of the M samples of $\mathbf{x}_{\mathcal{S}}$ is represented by the reproducer values of the uniform quantiser (see Section 2.3) at the sample locations, i.e., $\hat{\mathbf{x}}_{\mathcal{S}}$. This uses $Q = Q'\frac{N}{M} = \frac{R}{M}$ bits since the no bits are assigned to the $N - M$ omitted signal values $x_{m'}$, $m' \notin \mathcal{S}$. Using (3.7),

Algorithmus 4.1 Signal compression in the vertex domain.

For each signal $\mathbf{x}$:

- Sampling `S`: Draw M vertex samples $\mathbf{x}_{\mathcal{S}}$ according to $\mathrm{P}\{m \in \mathcal{S}\}$, such that $\operatorname{rank}(\mathbf{U}_{K,\mathcal{S}}) = K$.
- Quantisation `Q`: Uniform quantisation of $\mathbf{x}_{\mathcal{S}}$ with $Q = \frac{R}{M}$ bits per sample according to Section 2.3.
- Interpolation `I`: Perform bandlimited reconstruction (cf. Section 3.3) with quantised signal, i.e.,

$$\hat{\mathbf{x}} = \mathbf{U}_K \mathbf{U}_{K,\mathcal{S}}^{\#} \hat{\mathbf{x}}_{\mathcal{S}}.$$

we reconstruct a distorted version of $\mathfrak{x}_K$ and consequently $\mathbf{x}$ (cf. Fig. 4.2d). The compression procedure is summarised in Algorithm 4.1.

Note that this technique does not consider the correlation present in the vertex domain and therefore is expected to be suboptimal in case of non-white processes. We implemented it nonetheless as a fast reference technique as it requires no transform of the signal to a different domain.

4.2 Joint quantisation and sampling

In their paper [3], Li et al. propose a compression technique for stationary processes that minimises the mean square error (MSE) of a reconstructed bandlimited signal by alternately optimising the number of quantiser levels and a sampling filter. The MSE to be minimised reads

$$\mathrm{E}\left\{\left\|\mathbf{U}_K \mathfrak{x}_K - \hat{\mathbf{x}}\right\|^2\right\}$$

In a claim to facilitate a distributed implementation, they also present a similarly performing specialisation of their approach that uses only Laplacian polynomial filters [3, Algorithm 4].

Instead of R bits, $Z = 2^R$ quantiser levels are incrementally allocated for a given graph filter. Using a greedy algorithm, the next quantiser level is given to the signal value whose increased resolution maximally reduces the above-mentioned MSE. This requires multiple matrix multiplications for every allocated level. Since the

Algorithmus 4.2 Signal compression as described in [3, Section IV].

```
For each graph $\mathcal{G}, \mathrm{C}_{\mathbf{x}}$:
    • Initialise $\mathbf{h} = \mathbf{1}^T$ for $\mathbf{H} = \mathbf{I}$, i.e., vertex-domain sampling.
Repeat:
    • Perform iterative greedy quantiser level allocation for a given $\mathbf{H}$ based on highest reduction in $\mathrm{E}\left\{\|\mathbf{U}_K \mathfrak{x}_K - \hat{\mathbf{x}}\|^2\right\}$.
    • Optimise $\mathbf{h}$ for a given level allocation based on highest reduction in $\mathrm{E}\left\{\|\mathbf{U}_K \mathfrak{x}_K - \hat{\mathbf{x}}\|^2\right\}$.
    • Update $\mathbf{H} = \mathbf{U}\mathrm{Diag}(\mathbf{h})\,\mathbf{U}^T$.
Until convergence of $\mathbf{H}$.

For each signal $\mathbf{x}$:
    • Sampling S: Filter the signal with computed filter,
          $\mathfrak{x}_{\mathbf{H}} = \mathbf{H}\mathbf{x}$.
    • Quantisation Q: Uniform quantisation of $\mathfrak{x}_{\mathbf{H}}$ with precomputed level allocation according to Section 2.3.
    • Interpolation I: Perform signal reconstruction using the optimal MSE minimizing linear recovery matrix $\mathbf{\Phi}^*$ as defined in [3, (13)], i.e.,
          $\hat{\mathbf{x}} = \mathbf{U}_K \mathbf{\Phi}^* \hat{\mathfrak{x}}_{\mathbf{H},\mathcal{S}}$.
```

number of levels grows exponentially with R, individually allocating *quantiser levels* is severely limited to a low number of bits and therefore to small graphs. This motivated us to modify the algorithm by allocating quantiser *bits* instead of levels.

Given the allocated reproducer values, the Laplacian polynomial filter is then optimised, again with the goal of minimising the MSE. This is achieved by individually optimising every sampled value of the filter transfer function $\mathbf{h}$ in (3.6). The compression technique is summarised in Algorithm 4.2.

As explained in [3, Section IV-C], optimising every h_{n_m} requires several matrix multiplications as well as the computation of eigenvalues, eigenvectors and the square root of a matrix. Each h_{n_m} has to be optimised multiple times until a convergence condition is met. In addition, the concave-convex fractional nature of the cost function requires a reformulation, which we discuss presently, that involves multiple iterations of a convex optimisation itself. Consequently, the computational complexity is expected to make this algorithm infeasible for graphs of reasonable size.

Another key issue with [3, Algorithm 4] was the lack of numerical stability in the optimisation of the filter transfer function $\mathbf{h}$. The optimisation problem for noiseless observation reads [3, (29)]

$$\min_{h_{n_m}^2 \geq 0} \sum_{m'=1}^{M} \frac{a_{m'}}{h_{n_m}^2 + (\mathbf{\Lambda}_0)_{m'm'}},$$

where $a_{m'} = (\mathbf{A})_{m'n_m}^2 (\mathbf{\Lambda}_0)_{m'm'} - \sum_{n=1,n\neq n_m}^{N} (\mathbf{A})_{m'n}^2 h_n^2$. The authors define $\mathbf{A} = \mathbf{U}_0^T \mathbf{B}^{-1/2} \mathbf{U}_{\mathcal{S}} \mathrm{Diag}(\mathrm{diag}\,(c_{\mathfrak{x}11}, \ldots, c_{\mathfrak{x}KK}, 0, \ldots))$. $\mathbf{\Lambda}_0$ is the diagonal eigenvalue matrix of $\mathbf{B}^{-1/2}\mathbf{C}\mathbf{B}^{-1/2}$, while the columns of $\mathbf{U}_0^T$ are the eigenvectors of $\mathbf{B}^{-1/2}\mathbf{C}\mathbf{B}^{-1/2}$ (see [3, Section IV-C] for the definitions of $\mathbf{B}, \mathbf{C}$).

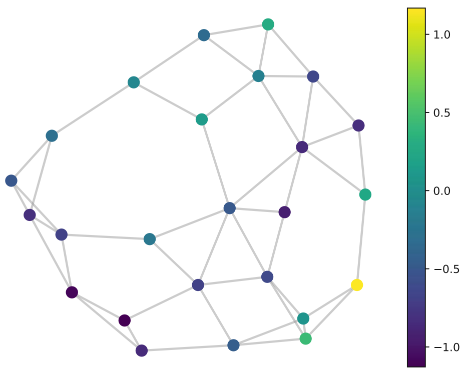
The authors claim that this problem can be solved with a quadratic transform technique introduced in [14]. This technique, however, requires the numerator $a_{m'}$ to be nonpositive[2] and the denominator $h_{n_m}^2 + (\mathbf{\Lambda}_0)_{m'm'}$ to be positive[3]. In our preliminary simulations, this condition was not met, not even in the stationary scenario that is listed in [3, Table I] and the provided source code. Thus, we neither considered the original nor our modified version of this technique for the simulations presented in Chapter 5.

4.3 GFT-domain compression

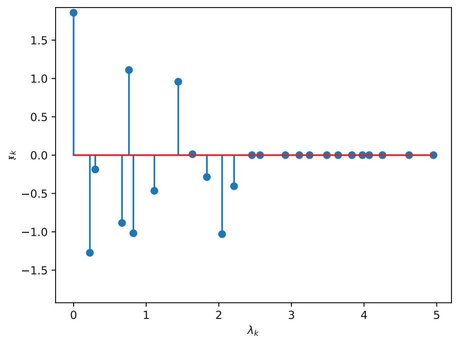
While vertex-domain samples are perfectly decorrelated for white processes, sampled GFT coefficients $\mathfrak{x}$ are decorrelated for stationary processes as these processes are characterised by uncorrelated coefficients in the frequency domain (see Section 3.4). Despite dealing with nonstationary processes, we implemented a transform coding technique using GFT coefficients $\mathfrak{x}_K$ up to the nominal bandwidth K. We expect that due to a sparse coefficient vector, (especially in bandlimited scenarios) this technique could yield acceptable results, even in case of moderate correlation in the

[2]Note the sign change due to the reformulation as a maximisation problem in [14].

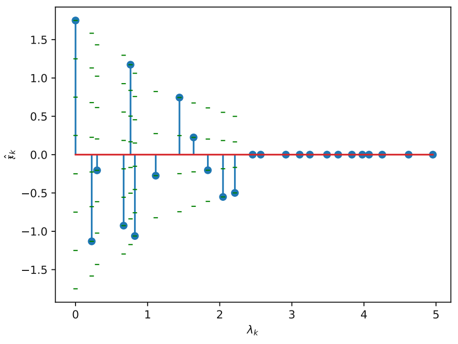
[3]Since the signs of numerator and denominator could be swapped, $a_{m'} > 0$ and $h_{n_m}^2 + (\mathbf{\Lambda}_0)_{m'm'} < 0$ would be possible too



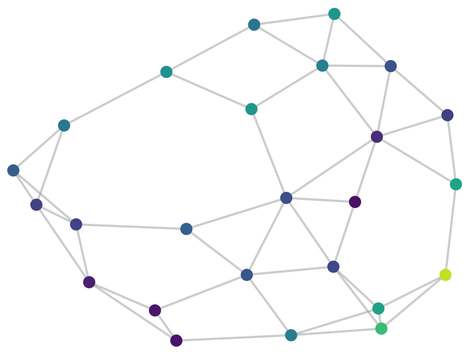
(a) Original signal in vertex domain



(b) Original signal in frequency domain



(c) Quantised signal in frequency domain, reproducer levels in green



(d) Reconstructed signal in vertex domain

Figure 4.3: Frequency-domain compression

frequency domain.

In a first step, we perform a GFT of the vertex-domain covariance matrix, i.e.,

$$\mathbf{C}_{\mathfrak{x}} = \mathbf{U}^T \mathbf{C}_{\mathbf{x}} \mathbf{U},$$

which is needed to limit the overloading probability of the uniform quantiser, as mentioned in Section 2.3. Similar to the greedy technique discussed in Section 4.2, the bits available to the coefficients $\mathfrak{x}_K = (\mathfrak{x}_1, \ldots, \mathfrak{x}_K)^T$ are allocated incrementally: Out of the already sampled frequencies and the next-highest unsampled frequency, the next bit is given to the frequency that leads to the maximum decrease of the expected quantiser distortion D_Q. This procedure is continued until all bits are allocated.

After uniformly quantising (see Section 2.3) using this bit allocation (see Fig. 4.3c), an IGFT of the quantised GFT coefficients is performed. Thereby, we obtain the

Algorithmus 4.3 Signal compression using the GFT.

```
For each graph $\mathcal{G}, \mathbf{C}_\mathbf{x}$:
    • Compute frequency correlation matrix $\mathbf{C}_\mathfrak{x} = \mathbf{U}^T \mathbf{C}_\mathbf{x} \mathbf{U}$.
    • Perform iterative greedy bit allocation based on highest reduction in $D_Q$.
For each signal $\mathbf{x}$:
    • Sampling S: Perform bandlimited GFT, i.e.,
          $\mathfrak{x}_K = \mathbf{U}_K^T \mathbf{x}$.
    • Quantisation Q: Uniform quantisation of $\mathfrak{x}_K$ with precomputed bit allocation
      according to Section 2.3.
    • Interpolation I: Perform IGFT with quantised signal, i.e.,
          $\hat{\mathbf{x}} = \mathbf{U}_K^T \hat{\mathfrak{x}}_K$.
```

reconstructed vertex-domain signal (see Fig. 4.3d) that is then considered for the distortion calculation. We summarised the procedure in Algorithm 4.3.

Compressing in the GFT domain is expected to offer better performance than vertex-domain compression in case of many real-life processes since they resemble stationary processes rather than white processes. This comes at the cost of calculating a computationally expensive GFT.

4.4 GGT-domain compression

In case of vertex- and Fourier-domain sampling, we rely on the signal structure to provide decorrelated data in either domain. In [3], the authors suggest an alternating optimisation algorithm that is computationally expensive.

4.4.1 Motivation

Since decorrelating a signal using the Karhunen-Loève transform is statistically effective yet computationally inefficient as described in Section 2.2, we investigate a different approach based on the Gabor transform. As described by [4] for the case of continuous time signals, the Gabor analysis consists of a sampled WFT where the time period and frequency period are matched to the temporal and spectral correlation of the signal.

The Gabor analysis in classical signal processing reads

$$\mathfrak{x}_g\left(iT, kF\right) = \langle x, \gamma_{iT,kF}\rangle = \int_t x\left(t\right)\gamma^*_{iT,kF}\left(t\right)dt,$$

where i is the time index, k is the frequency index and $\gamma_{iT,kF}\left(t\right)$ is a analysis window, translated by iT and modulated by kF, as described in (3.3).

The Gabor synthesis using a synthesis window $g\left(t\right)$ is formulated as follows

$$x\left(t\right) = \sum_{i,k}\mathfrak{x}_g\left(i,k\right)g_{iT,kF}\left(t\right).$$

[4] assumes an underspread process, which in ordinary signal processing can be expressed as

$$\tau_0\nu_0 \leq \frac{1}{4},$$

where τ_0 and ν_0 are the correlation widths in vertex and frequency domain, i.e., the support of the expected ambiguity function

$$EA\left(\tau,\nu\right) = \int_t r_x\left(t, t-\tau\right)e^{-\omega\nu t}dt$$

that indicates correlation of different temporal and spectral components. Here, $r_x\left(t,t'\right) \coloneqq \mathrm{E}\left\{x\left(t\right)x^*\left(t'\right)\right\}$ denotes the autocorrelation function of $x\left(t\right)$.

Furthermore, [4] assumes an overcritical Gabor sampling grid to achieve a complete Gabor basis. This is mathematically expressed by

$$TF \leq 1.$$

Finally, the author proposes a sampling grid ratio that is matched to the correlation characteristics, that is given by

$$\frac{T}{F} = \frac{\tau_0}{\nu_0}.$$

Using the WGFT as introduced in Section 3.2.3, we propose to apply the concept described in [4] to graph signals, referred to as the graph Gabor transform (GGT). For the Gabor analysis, instead of considering all N^2 WGFT coefficients $\boldsymbol{\mathfrak{x}}_{\mathbf{g}}$, we pick $M \gtrsim N$ samples $\boldsymbol{\mathfrak{x}}_{\mathbf{g},\mathcal{S}}$ by leaving out columns in the shifted window matrix $\boldsymbol{\Gamma}$, i.e.,

$$\boldsymbol{\mathfrak{x}}_{\mathbf{g},\mathcal{S}} = \boldsymbol{\Gamma}_{\mathcal{S}}^T \mathbf{x} = \left(\boldsymbol{\gamma}_{i_1 k_1}, \ldots, \boldsymbol{\gamma}_{i_{\check{m}} k_{\breve{m}}}, \ldots, \boldsymbol{\gamma}_{i_J k_L}\right)^T \mathbf{x},$$

where $\check{m}$ and $\breve{m}$ are the frequency and vertex shift indices at sample m. To simplify notation, we denote the columns of $\boldsymbol{\Gamma}_{\mathcal{S}}$ as

$$\boldsymbol{\mathfrak{x}}_{\mathbf{g},\mathcal{S}} = \boldsymbol{\Gamma}_{\mathcal{S}}^T \mathbf{x} = \left(\bar{\boldsymbol{\gamma}}_1, \ldots, \bar{\boldsymbol{\gamma}}_M\right)^T \mathbf{x} \tag{4.1}$$

instead, where $\bar{\boldsymbol{\gamma}}_m = \boldsymbol{\gamma}_{i_{\check{m}} k_{\breve{m}}}$ and thus the index conversion reads

$$m = \check{m}\,(L-1) + \breve{m}.$$

4.4.2 Choice of graph Gabor window and lattice

[4] assumed an underspread process and an overcritical Gabor sampling grid. The former is satisfied with moderate choices for the nonstationarity parameters presented in Section 3.4. The latter is satisfied since $M \gtrsim N$.

In order to achieve decorrelated coefficients that we can effectively quantise, we intend to adhere to the matched sampling grid ratio [4, eq. (11)], i.e., we want to select the sampling set according to the signal correlation in the vertex and frequency domain. When the signal is generated according to the *filtering and modulation* approach (cf. Section 3.4.1), we chose $J \in [1, N]$, the number of samples, directly proportional to $k_{\mathrm{rms}} \in \left[1, \sqrt{\frac{(N+1)(2N+1)}{6}}\right]$. We then set the bandwidth of $\mathbf{m}$ to J, with the intention of introducing weak frequency-domain correlation for signals strongly correlated in the vertex domain (i.e., $J \approx 1$) and vice versa (i.e., $J \approx N$). Finally, we set the number of regularly-spaced frequency samples to $L = \frac{M}{J}$. However, as explained in Section 3.4.1, it is hard to predict the correlation based on the presented parameters, especially the bandwidth of $\mathbf{m}$.

In case of the frequency domain band matrix of Section 3.4.2, only the B neighbouring frequencies are correlated and thus approximately every B^{th} frequency is

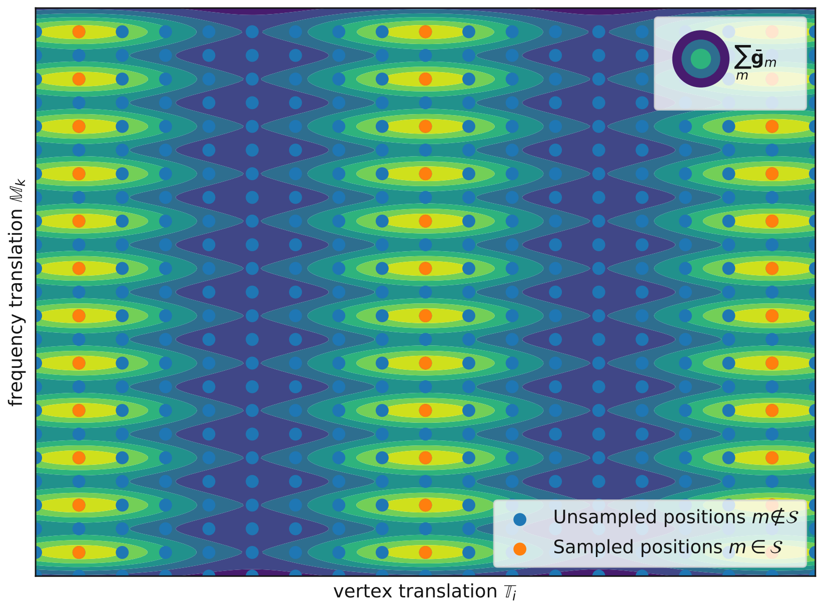


Figure 4.4: Adaption of the decay parameter of an exponential WFT atom according to the matched sampling grid ratio

included in the sampling set[4]. With $L = \frac{M}{B}$ frequency shifts taken into account, we sample at $J = B$ nodes to total up to $LJ = M$ coefficients. If $\frac{M}{B} > N$, we choose $L = \frac{N}{B}$ and $J = \frac{BM}{N}$ instead.

The analysis window $\boldsymbol{\gamma}$ must be designed according to the signal statistics and thus L and J. If the weakly correlated frequencies require a high number of sampled frequency shifts, the effective support of $\boldsymbol{\gamma}$ in the vertex domain matters as an atom needs to stretch over a large number of graph nodes. This is illustrated in Fig. 4.4: With $L > J$, the analysis window $\boldsymbol{\gamma}$ features a large support in the vertex domain such that vertex-translated versions cover all unsampled graph nodes. Since every second frequency is sampled, the support in the frequency domain is narrow to limit correlation of neighbouring GGT coefficients.

The choice of the parameter(s) of $\boldsymbol{\gamma}$ turned out to be delicate when considering the goal to achieve a lower condition number $\kappa\left(\boldsymbol{\Gamma}_{\mathcal{S}}\right)$ and thus accuracy with fewer, less correlated samples. For every graph realisation, L and J are chosen in the manner of the matched sampling grid ratio. Then, we partition the graph into J subgraphs using spectral clustering [8, Sec. 7.5.2] and sample at the centre nodes of each subgraph. We define the centre node as the node that has a path to each other

[4]The exact choice of L, J is a matter of tuning according to $\mathbf{C}_{\mathbf{x}}$ and $\mathbf{C}_{\mathfrak{r}}$.

node in its subgraph with a minimum number of hops. We implemented this by multiplying the clusters adjacency matrix $\mathbf{A}_j$ with itself until all off-diagonal entries of a row are non-zero.

In order to spread out the L frequency sampling grid more evenly, we sampled at equidistant graph frequency *values* λ_k instead of equidistant graph frequency *indices* k, i.e.,

$$k_{\check{m}} = \arg\min_k \left| \lambda_k - (\check{m} - 1) \frac{\lambda_N + \lambda_1}{L} \right|, \qquad \check{m} \in [1, L] \, .$$

We minimise $\kappa\left(\mathbf{\Gamma}_{\mathcal{S}}\right)$ for this set of sampled vertices and frequencies by varying the parameters that define $\boldsymbol{\gamma}$. Since we used an exponential window $\boldsymbol{\gamma} = \mathbf{U}^T e^{-\tau_g \boldsymbol{\lambda}}$ with only one parameter τ_g, we approached the optimisation problem with the golden section search [15, Sec. 7.1]. Still, we choose L and J from the nonstationarity parameters (e.g., k_{rms} and the bandwidth of $\mathbf{m}$) and not the actual level of correlation. In Section 3.4.1, we explained that k_{rms} obtained from our vertex-correlation filter $\mathfrak{a}$ might be altered by the frequency-domain correlation filter $\mathbf{m}$. Thus, there are cases where our chosen τ_g still yields inaccurate results.

4.4.3 Quantisation and reconstruction

The bit allocation happens in a similar fashion as for the frequency-domain sampling described in Section 4.3. We can, however, no longer expect that the next-highest unsampled vertex-frequency shift is also the unsampled shift with the most power. Thus, we must always consider all M graph Gabor coefficients[5] when allocating another bit according to maximal reduction in expected quantisation distortion $D_{\mathcal{Q}}$. Since the GGT grid is spread over all nodes and graph frequencies, the incremental bit allocation might spend bits on frequency shifts $k > K$ in case K is just the nominal but not the effective bandwidth as described in Section 3.4.1.

The WFT reconstruction stated in (3.5) no longer holds for the GGT. Thus, we formulate an optimisation problem to find an almost dual Gabor synthesis window $\mathbf{g} \in \mathbb{R}^N$ to perform Gabor synthesis with the same shift structure as in Gabor

[5]This aspect already hints higher values for k_{eff} and thus a less sparse sampling domain.

analysis (cf. (4.1)), analogous to the method described in [4], i.e.,

$$\mathbf{G}_{\mathcal{S}} = \mathbf{S}_{\mathcal{S}}\mathbf{U}^T\mathbf{g}$$

The problem formally reads

$$\min_{\mathbf{g}} \left\| \mathbf{G}_{\mathcal{S}}\mathbf{\Gamma}_{\mathcal{S}}^T - \mathbf{I}_N \right\|_F^2 ,$$

where $\mathbf{I}_N \in \mathbb{R}^{N\times N}$ is the identity matrix and $\mathbf{S}_{\mathcal{S}} \in \mathbb{R}^{N\times M\times N}$ is a third-order tensor that performs all vertex-frequency shifts. The entry $(\mathbf{S}_{\mathcal{S}})_{nml}$ is given by

$$(\mathbf{S}_{\mathcal{S}})_{nml} = N\,(\mathbf{U})_{nk_{\breve{m}}}\,(\mathbf{U})_{i_{\breve{m}}l}\,(\mathbf{U})_{nl}\,.$$

Although we are able to solve this convex problem numerically in a computationally efficient manner, only sampling sets with either all vertices and/or all frequencies in them (i.e., $L = N$ and/or $J = N$) yielded results that led to a cost function close to zero and thus a well-recovered signal. This might be due to the undesirable properties of the WGFT mentioned in Section 3.2.2. Thus, we stick to the unstructured and less efficient method of reconstructing the original signal $\mathbf{x}$ with $\left(\mathbf{\Gamma}_{\mathcal{S}}^T\right)^{\#}$, the pseudoinverse of $\mathbf{\Gamma}_{\mathcal{S}}^T$. The reconstruction then reads

$$\hat{\mathbf{x}} = \left(\mathbf{\Gamma}_{\mathcal{S}}^T\right)^{\#}\hat{\mathfrak{r}}_{\mathbf{g},\mathcal{S}},$$

where $\left(\mathbf{\Gamma}_{\mathcal{S}}^T\right)^{\#} = \left(\mathbf{\Gamma}_{\mathcal{S}}\mathbf{\Gamma}_{\mathcal{S}}^T\right)^{-1}\mathbf{\Gamma}_{\mathcal{S}}$ is the pseudoinverse of $\mathbf{\Gamma}_{\mathcal{S}}^T$ and $\hat{\mathfrak{r}}_{\mathbf{g},\mathcal{S}}$ are the quantised GGT coefficients. This operation thus requires $\operatorname{rank}(\mathbf{\Gamma}_{\mathcal{S}}) = N$. We summarised the GGT compression system in Algorithm 4.4.

Algorithmus 4.4 Signal compression using the GGT.

For each graph $\mathcal{G}, \mathbf{C}_\mathbf{x}$**:**

- Pick M as a tradeoff between reconstruction robustness ($M \gg N$) and coefficient correlation ($M \gtrsim N$).
- Pick L, J indirectly proportional to (estimated) correlation in vertex and frequency domain, s.t. $LJ = M$.
- Set vertex sampling indices $i_{\breve{m}}$ to J cluster centres of spectral clustering.
- Choose L evenly spread graph frequencies $\lambda_{k_{\breve{m}}}$.
- Choose window $\boldsymbol{\gamma}$ and find its parameter(s) that minimise(s) $\kappa\left(\boldsymbol{\Gamma}_\mathcal{S}\right)$, e.g. using the golden section search. Keep $\boldsymbol{\Gamma}_\mathcal{S}$ of optimal $\boldsymbol{\gamma}$.
- Perform iterative greedy bit allocation based on highest reduction in $D_\mathcal{Q}$.

For each signal $\mathbf{x}$:

- Sampling `S`: Perform Gabor analysis, i.e.,

$$\mathfrak{x}_{\mathbf{g},\mathcal{S}} = \boldsymbol{\Gamma}_\mathcal{S}^T \mathbf{x}.$$

- Quantisation `Q`: Uniform quantisation of $\mathfrak{x}_{\mathbf{g},\mathcal{S}}$ with precomputed bit allocation according to Section 2.3.
- Interpolation `I`: Perform Gabor synthesis with quantised signal, i.e.,

$$\hat{\mathbf{x}} = \left(\boldsymbol{\Gamma}_\mathcal{S}^T\right)^{\#} \hat{\mathfrak{x}}_{\mathbf{g},\mathcal{S}}.$$

5 Numerical results

This chapter is dedicated to presenting and discussing simulation results that cover the techniques presented above. Due to the large number of parameters available to modify with regard to both process generation and compression techniques, we can cover only a selected number of sweeps over single parameters. While we implemented the technique introduced by [3] (cf. Section 4.2), it did not lead to satisfactory results due to issues with numerical stability and computational complexity. Thus, it is not included in the simulations analysed in this chapter.

All simulations of graph signal compression techniques shown are for on graphs of size $N = 64$. The outcomes for other graph sizes were observed to be qualitatively similar. We simulated over 10 graph realisations and $5N = 320$ signal realisations per graph realisation. We set the number of samples for vertex-domain sampling to $M_{\mathbf{x}} = \min(1.1K, N)$ to facilitate $\operatorname{rank}(\mathbf{U}_{K,\mathcal{S}_{\mathbf{x}}}) = K$. With Q bits per sample, this results in a total bit budget of

$$R = M_{\mathbf{x}} Q = \min(1.1KQ, NQ)\,, \tag{5.1}$$

equivalent to a rate of $Q' = \frac{R}{N}$ bits per node. The same rate was used for the other techniques.

Throughout this chapter, we use the same figure layout; thus we discuss it here in more detail. The plots compare the simulation results of the compression techniques of Chapter 4 in terms of rate Q' and distortion D' (see below). For reasons of clarity and comprehensibility, each plot only shows results of one simulation run, i.e., a sweep over just one parameter while the others were fixed.

Unlike the distortion D presented in (2.1), the distortion in the following plots was normalised to the mean signal energy $E_{\mathbf{x}}$, i.e., $D' = \frac{D}{E_{\mathbf{x}}}$. As $D' > 1$ means distortion that exceeds the signal energy and thus indicates total destruction of the source signal, the exact values of these results are not displayed. Instead, these outcomes are indicated by a marker to the right of the plot.

Quantisation quality depends on the signal correlation in the compression domain as mentioned in Section 2.2; we therefore colour the markers according to the percentage of small correlation coefficients in the vertex and frequency domain, i.e., $b\left(\mathbf{x}, \frac{1}{5}\right)$ (cf. (3.8)).This colour map indicating the correlation levels is shown on the plot as well, with the simulation runs also marked there (in green). The more transparent dots correspond to low-rate simulations[1].

5.1 Class I processes

In the following examples, we created processes with the method described in Section 3.4.1. Note that the effective bandwidth generally exceeds the nominal bandwidth K due to the vertex-domain filter $\mathbf{m}$. For each set of parameters, we simulated the compression behaviour for all graph models described in Section 3.1.2 with their coherence ranges listed in Table 3.1.

For regular and geometric graphs, we chose moderate a moderate average node degree of 3. We perturbed $\frac{1}{6}$ of the nodes of a regular graph. The GGT requires $M_{\mathfrak{r}_\mathbf{g}} > N$ samples; preliminary results showed that $M_{\mathfrak{r}_\mathbf{g}} - N = 0.85N$ is a good tradeoff between coefficient sparsity and precision for the unquantised GGT.

5.1.1 Correlated signals

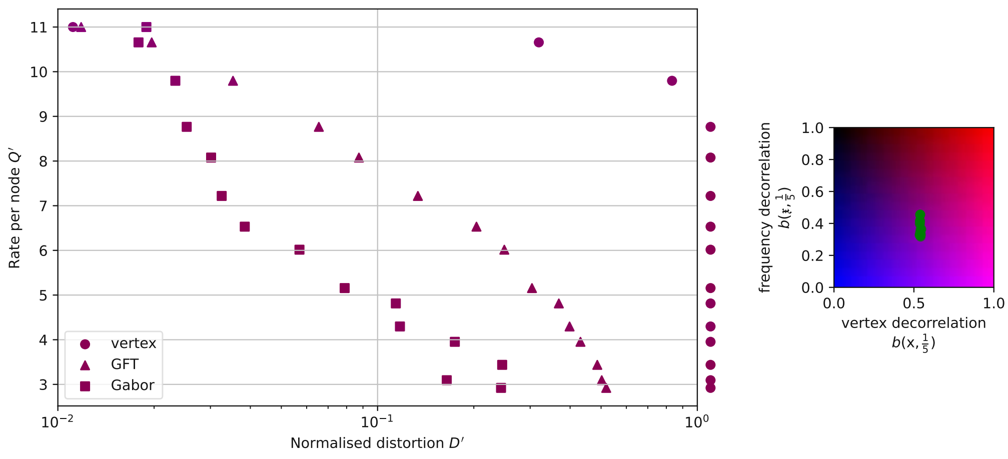
5.1.1.1 Varying bandwidth

Fig. 5.1 displays the simulation results for the high-rate scenario summarised in Table 5.1[2]. All parameters were fixed except for the nominal bandwidth, which was varied as $K \in \left[\frac{N}{4}, N\right]$. We used an exponential decay for $\mathfrak{a}$, i.e.,

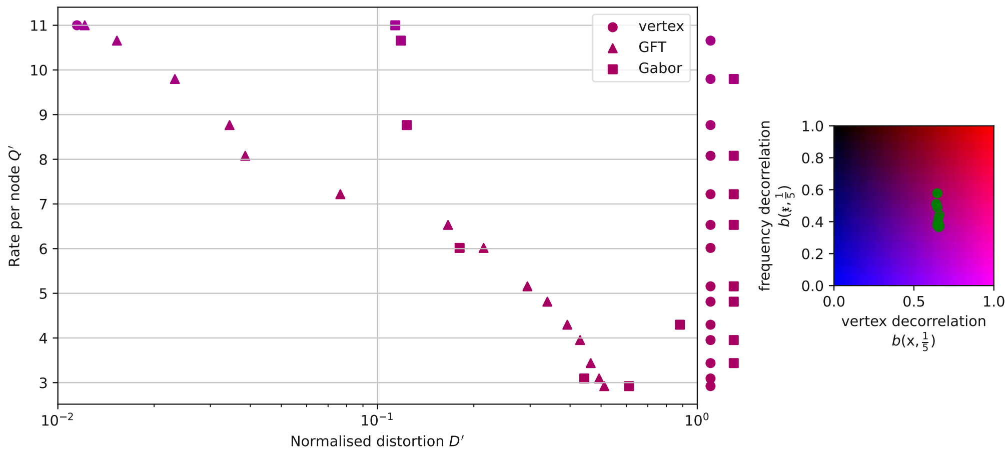
$$(\mathfrak{a})_k = \begin{cases} e^{-\frac{\tau_{\mathfrak{a}}}{2}(k-1)} & k = 1, \ldots, K, \\ 0 & \text{otherwise.} \end{cases}$$

[1]This detailed legend intends to facilitate readability when correlation remains similar throughout the simulation.

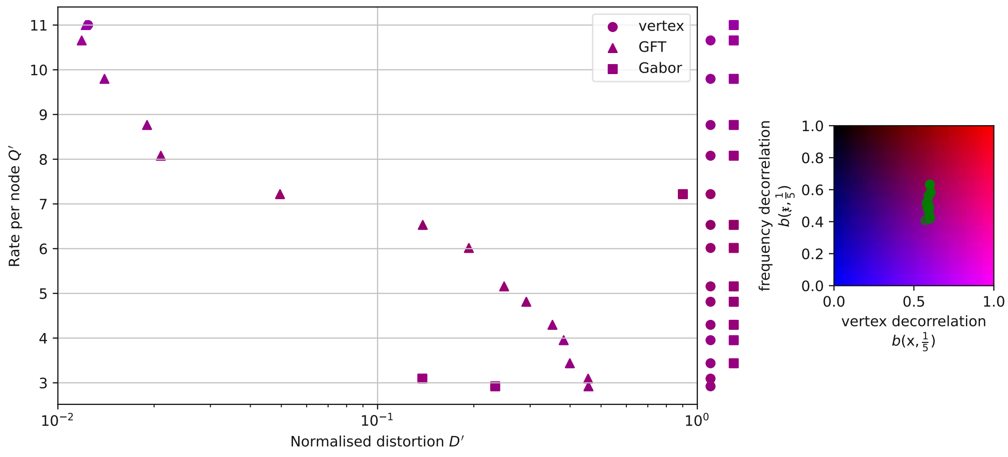
[2]Average degree for regular, geometric graphs.



(a) Perturbed random regular graphs



(b) Random geometric graphs



(c) Barabási–Albert graphs

Figure 5.1: Rate-distortion comparison for varying K using the parameters from Table 5.1 for the three different graph models. The rate-distortion function yields $D' \in [1.3 \cdot 10^{-9}, 3.7 \cdot 10^{-8}]$.

Parameter	**Value**
graph size, N	64
nominal bandwidth, K	$[16, 64]$
average node degree	3
perturbed regular edges	$16.6\,\%$
vertex-correlation filter rate, $\tau_{\mathfrak{a}}$	0.125
GGT oversampling, $M_{\mathfrak{x}_{\mathbf{g}}} - N$	$0.85N$
bits per vertex sample, Q	11

Table 5.1: Parameters for scenario in Section 5.1.1.1.

Graph model	$b\left(\mathfrak{x}_{\mathbf{g},\mathcal{S}}, \frac{1}{5}\right)$
Perturbed random regular	$57\,\%$
Random geometric	$66\,\%$
Barabási–Albert	$53\,\%$

Table 5.2: Percentage of small cross-correlation Gabor coefficients $b\left(\mathfrak{x}_{\mathbf{g},\mathcal{S}}, \frac{1}{5}\right)$, averaged over nominal bandwidth K. Parameters of this scenario are listed in Table 5.1.

We chose $\tau_{\mathfrak{a}}$ such that $c_{\tilde{\mathfrak{x}}NN} = 1.28{\cdot}10^{-7}$ for $K = N$; we are far off the maximum value of k_{rms} and thus expect noticeable vertex-domain correlation. This holds true as indicated by the colour map. However, vertex correlation remained almost constant throughout the bandwidth sweep, even though k_{rms} is limited to a greater extent for smaller nominal bandwidths. This suggests that the contribution to vertex-domain correlation is mainly caused by the rapid decay in power over the first (dominant) frequencies.

There is noticeable but not extreme vertex-domain correlation, we set a high bandwidth of $\mathbf{m}$ and a constant spectrum, with the intention of generating significant frequency-domain correlation. The correlation plots in our simulations show that this strategy works more effectively for graph models with lower coherence.

Since we chose $J > L$, the correlation of the Gabor coefficients is similar to vertex correlation; as shown in Table 5.2. Our choice for L and J were based on the frequency correlation we expected. Hence, the GGT sampling grid ratio is not well-matched to the correlation characteristics as more frequency samples would have been necessary for uncorrelated frequencies. This yields poor results for the method introduced in Section 4.4 for random geometric graphs and even more so for Barabási–Albert graphs.

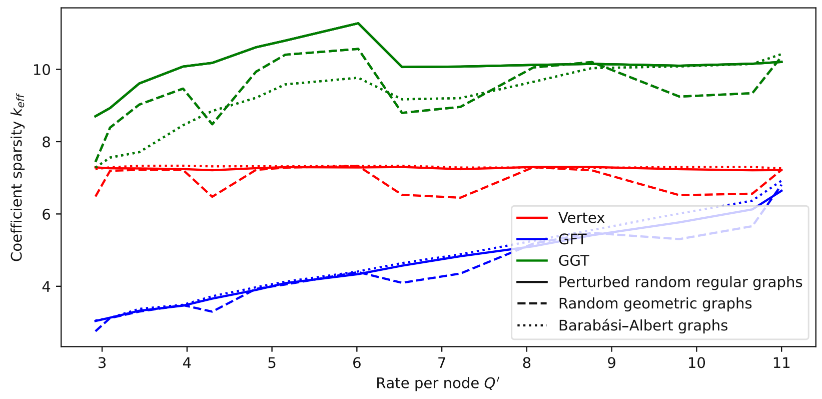


Figure 5.2: Coefficient sparsity k_{eff} versus rate for various transforms using parameters in Table 5.1.

For perturbed random regular graphs (see Fig. 5.1a), the signals have significant correlation in both the vertex domain and the frequency domain. The GGT compression that samples incompletely in both domains, skipped correlated samples in each domain and outperformed the other two compression techniques. Since frequency correlation exceeded vertex correlation, sampling at more vertices than frequencies proved correct for this graph model.

The uncorrelated frequencies that occurred for high-coherence graph models led to acceptable results of GFT-domain compression (see Figure 5.1b and Figure 5.1c) that was described in Section 4.3. Growing bandwidth (and thus rate R, see (5.1)) leads to a further decrease in frequency correlation, which causes improving GFT compression results in turn.

Vertex-domain compression (see Section 4.1) appears to be unusable unless $K = N$. Due to quantisation, the effective bandwidth of $\hat{\mathbf{x}}_{\mathcal{S}}$ is higher than the nominal bandwidth, which causes $D' > 1$ and thus, total destruction. The use of $\mathbf{m}$ can also cause a higher effective bandwidth. Since vertex-domain compression also fails for scenarios created with a frequency domain band matrix (see Section 5.2), we suspect quantisation to mainly cause issues for this technique.

Especially for this high-rate scenario, all simulation results lie far off the rate-distortion function in the area of $D' \approx 10^{-8}$, which marks the theoretical boundary as explained in Section 2.1.

Figure 5.2 shows k_{eff} versus R for all sampling domains and graph models, indicated by different colours and line styles, respectively. Since the power remains evenly

Parameter	Value
graph size, N	64
nominal bandwidth, K	$\left\lceil 1 - \frac{2}{\tau_{\mathfrak{a}}} \ln c \right\rceil$
average node degree	3
perturbed regular edges	16.6 %
vertex-correlation filter rate, $\tau_{\mathfrak{a}}$	$[0.136, 0.535]$
GGT oversampling, $M_{\mathfrak{r}_{\mathfrak{g}}} - N$	$0.85N$
bits per vertex sample, Q	11

Table 5.3: Parameters for scenario in Section 5.1.1.2.

spread over all nodes, k_{eff} for vertex-domain sampling does not change over bandwidth.

In the GFT domain, new frequencies with non-zero power are taken into consideration for sampling. Thus, k_{eff} grows over K. The quantity remains smaller than for other domains, however, which explains why GFT-domain compression works despite being the most correlated domain in this scenario. Conversely, the Gabor samples show the lowest correlation, however, they lack the sparsity of the GFT domain.

5.1.1.2 Varying frequency power decay

Another approach to simulate scenarios with different levels of correlation is a sweep over $\tau_{\mathfrak{a}}$. Once $(\mathfrak{a})_K = c \ll 1$, in our case $c = \frac{1}{44}$, we set the remaining values of $\mathfrak{a}$ to zero for a bandlimited setting[3] with

$$K = \left\lceil 1 - \frac{2}{\tau_{\mathfrak{a}}} \ln c \right\rceil .$$

High values for $\tau_{\mathfrak{a}}$ correspond to high correlation (also in the frequency domain, see colour map of Fig. 5.3) but low bit budget due to a low bandwidth (cf. (5.1)). Low values for $\tau_{\mathfrak{a}}$ indicate more moderate levels of correlation and a higher bit budget. Of note is that correlation in both domains grow at a similar pace, despite only varying the vertex-domain correlation parameter.

While vertex-domain compression remains unusable due to insufficient oversampling, GFT compression benefits both from the increasing rate and the decreasing frequency-

[3]Note, however, that $\mathbf{m}$ can still increase the effective bandwidth.

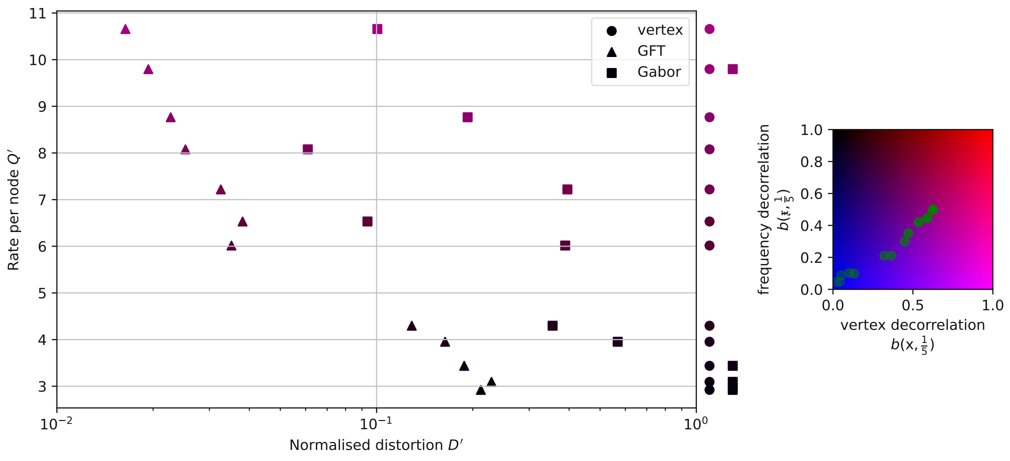


Figure 5.3: Rate-distortion comparison for varying $\tau_{\mathfrak{a}}$ using the parameters from Table 5.3 on random geometric graphs. The rate-distortion function yields $D' \in [1.45 \cdot 10^{-9}, 1.87 \cdot 10^{-9}]$.

domain correlation. However, even in low-rate scenarios, the direct comparison with Fig. 5.1b shows that a $\tau_{\mathfrak{a}}$ that is four times higher yields better GFT compression results. A reason for this finding could lie in faster decreasing frequency powers. Thus, it is sufficient to represent the signal with less GFT coefficients at a higher rate.

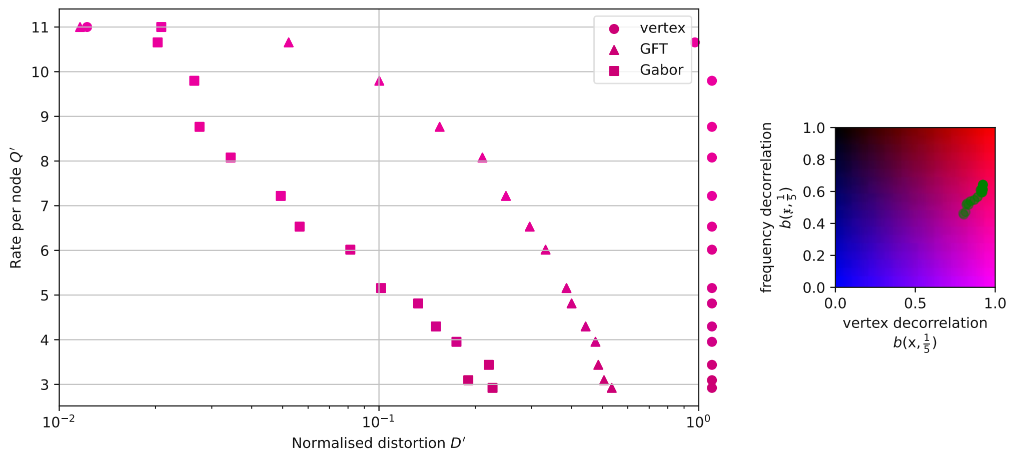
The GGT features the most uncorrelated coefficients throughout the sweep. It has 2 % fewer large correlation coefficients than the vertex domain on average, but suffers from less sparse coefficients. Due to a larger $\tau_{\mathfrak{a}}$, we set $L > J$, i.e., we sampled at more frequency than vertex shifts. As the correlation in the frequency domain is unexpectedly high too, dense sampling in this domain leads to insufficient results.

However, as we chose a low bandwidth for $\mathbf{m}$ due to a large $\tau_{\mathfrak{a}}$, we obtain a more predictable behaviour of the vertex-domain correlation (cf. Section 3.4.1). This leads to suboptimal but not destructive results in the GGT, compared to the results depicted in Fig. 5.1b.

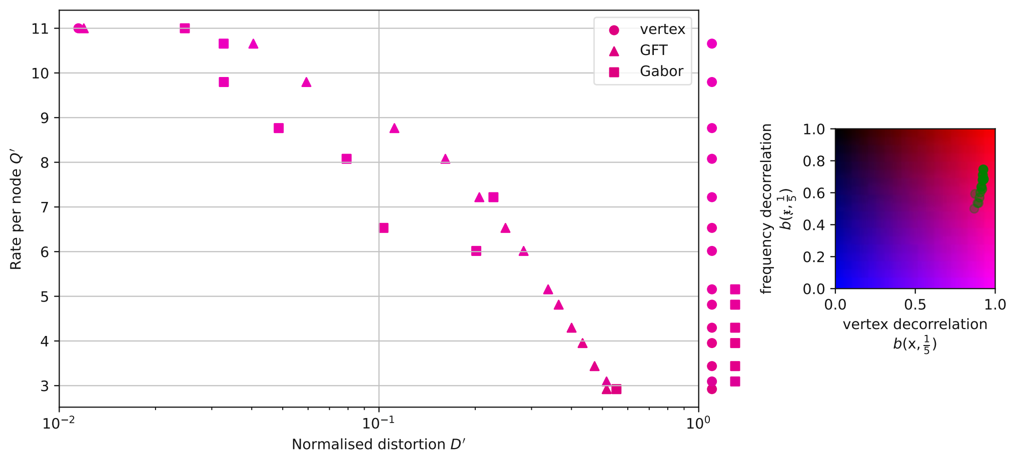
5.1.2 Weakly correlated signals

Table 5.4 lists the parameters for a less correlated but still high-rate scenario. The goal was to decrease vertex-domain correlation with a small $\tau_{\mathfrak{a}}$[4], while increasing

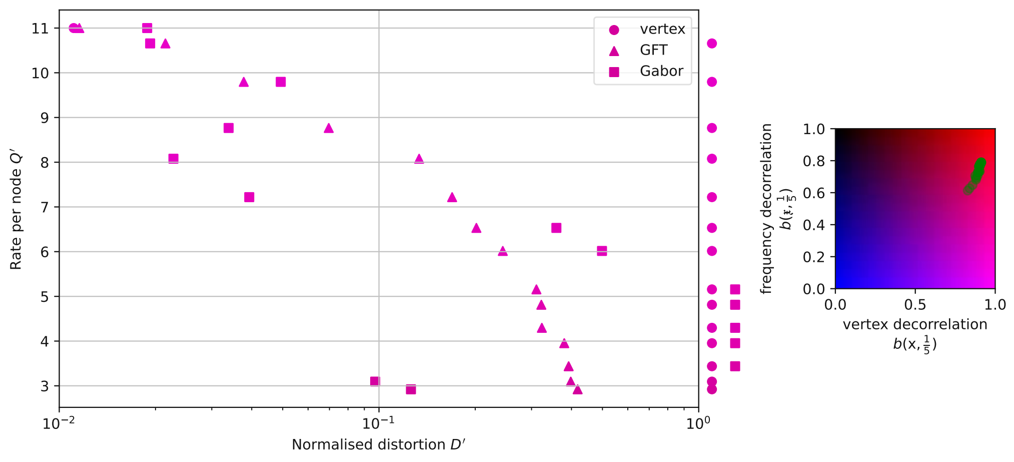
[4]This leads to higher values for k_{rms}.



(a) Perturbed random regular graphs



(b) Random geometric graphs



(c) Barabási–Albert graphs

Figure 5.4: Rate-distortion comparison for varying K using the parameters from Table 5.4 for the three different graph models. The rate-distortion function yields $D' \in [1.2 \cdot 10^{-8}, 6.7 \cdot 10^{-8}]$.

Parameter	**Value**
graph size, N	64
nominal bandwidth, K	$[16, 64]$
average node degree	3
perturbed regular edges	16.6 %
vertex-correlation filter rate, $\tau_{\mathfrak{a}}$	0.035
GGT oversampling, $M_{\mathfrak{r}_{\mathbf{g}}} - N$	$0.85N$
bits per vertex sample, Q	11

Table 5.4: Parameters for scenario in Section 5.1.2.

Graph model	$b\left(\mathfrak{r}_{\mathbf{g},\mathcal{S}}, \frac{1}{5}\right)$
Perturbed random regular	82 %
Random geometric	87 %
Barabási–Albert	76 %

Table 5.5: Percentage of small cross-correlation Gabor coefficients $b\left(\mathfrak{r}_{\mathbf{g},\mathcal{S}}, \frac{1}{5}\right)$, averaged over nominal bandwidth K. Parameters of this scenario are listed in Table 5.4.

frequency-domain correlation with a high bandwidth of $\mathbf{m}$. As indicated by the colour map in Fig. 5.4, correlation in *both* domains is reduced instead, compared to the simulation described in Section 5.1.1. The level of correlation in the vertex domain remains lower than in the frequency domain, however. Thus we chose $J > L$ again, which yielded samples with correlation similarly low as vertex-domain samples, as shown in Table 5.5.

Note that with a smaller $\tau_{\mathfrak{a}}$, higher frequencies now contribute noticeably to k_{rms} and thus to a reduction in vertex-domain correlation. The colour map in Fig. 5.4a shows best that the runs at higher bandwidths, depicted as the less transparent green dots, are less correlated.

Unlike in Section 5.1.1, GGT compression performs the best in mid- to high-bandwidth scenarios for all three graph models. In scenarios with low nominal bandwidth, only perturbed random regular graphs, which have the lowest coherence out of the graph models used, benefit from GGT compression. This coincides with the region where the GGT coefficients are decreasingly sparse, as shown in Fig. 5.5. Once a drop in k_{eff} happens (for reasons unclear to the author), the GGT compression shows better performance than the other two techniques for all graph models.

Since generating frequency-domain correlation with $\mathbf{m}$ works better for low-coherence

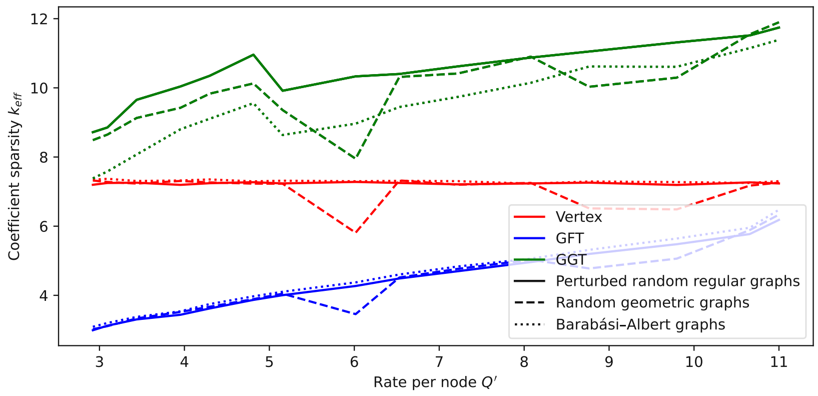


Figure 5.5: Coefficient sparsity k_{eff} versus rate for various transforms using parameters in Table 5.4.

graphs, high-coherence graphs gain less advantage of GGT compression over GFT compression again (see Fig. 5.4c). As frequency-domain samples become less correlated with increasing bandwidth, the performance of GFT compression improves. For $K = N$, when both vertex- and GFT-domain compression cover the entire bandwidth, they slightly outperform the GGT-based approach.

Like in the simulations of Section 5.1.1, vertex-domain correlation only works for $K = N$, as the signal is not truly bandlimited. The rate-distortion function shows that the theoretical limits of vector quantisations is orders of magnitude away from our results at $D' \approx 10^{-8}$.

Despite this weakly correlated, high-rate scenario, the simulation results in Fig. 5.6 show that the GGT compression is susceptible to mismatched L, J. Compared to the simulations of Fig. 5.4b, L was increased while J was decreased to still obtain $LJ = M$; the remaining parameters remained unchanged. In this setting, GGT compression only works in selected examples and performs worse than the other presented techniques.

5.2 Class II processes

We realised the processes for the following simulation with the method described in Section 3.4.2, where an upper band matrix $\mathfrak{B}$ is used to create a band GFT covariance matrix $\mathbf{C}_{\mathfrak{r}}$. With this technique, the effective bandwidth of the unquantised signal equals the nominal bandwidth K.

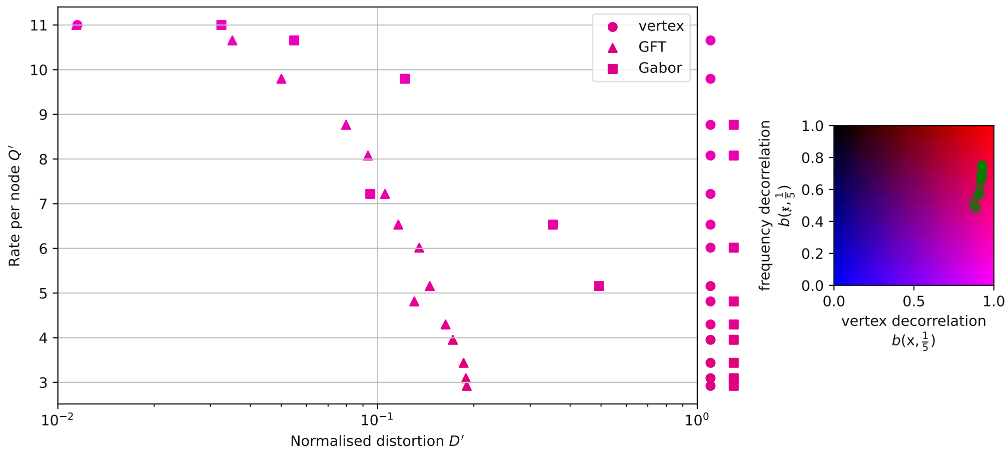


Figure 5.6: Rate per node Q' versus normalised distortion D' for mismatched GGT sampling grid ratio on random geometric graphs. The rate-distortion function yields $D' \in [1.2 \cdot 10^{-8}, 5.6 \cdot 10^{-8}]$.

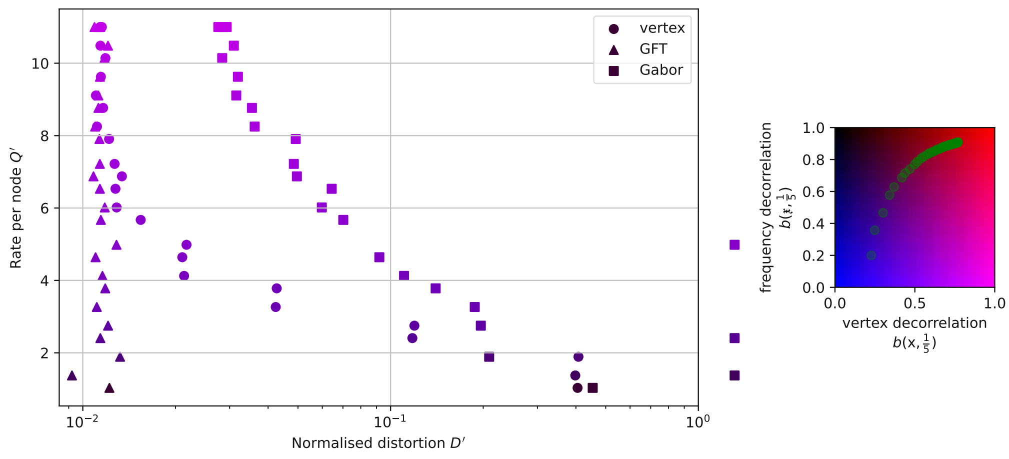
5.2.1 Bandwidth sweep

In this section, we analyse simulation results for processes with increasing bandwidth K. We checked the correlation in both frequency and vertex domain as a function of K and the number of super- and subdiagonals B with the other parameters listed in Table 5.6. In order to get a broad sweep over correlation, we set $B = 3$.

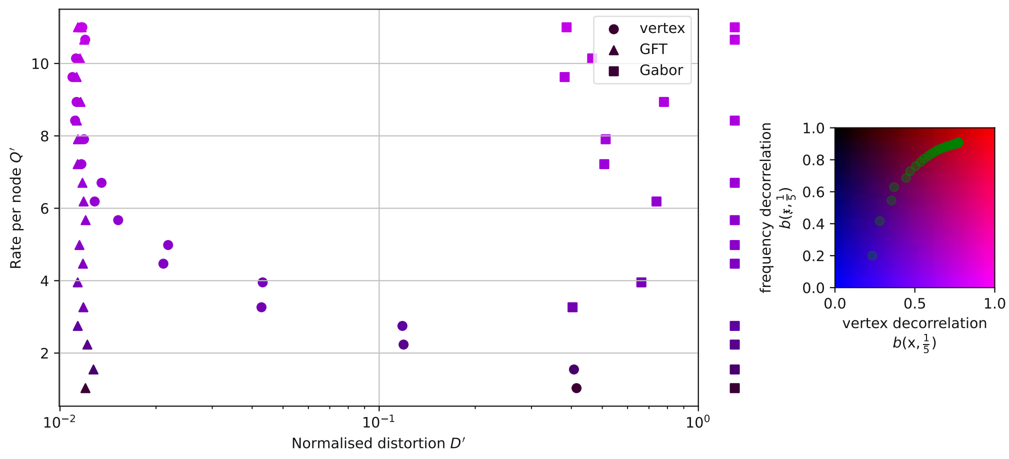
In this scenario, frequency-domain correlation is decreasing faster than vertex-correlation with increasing bandwidth K, as shown by the colour map of Fig. 5.7; perhaps be-

Parameter	**Value**
graph size, N	64
nominal bandwidth, K	$[6, 64]$
average node degree	3
perturbed regular edges	16.6 %
process generation method	upper band matrix $\mathfrak{B}$
vertex-correlation filter rate, $\tau_{\mathfrak{a}}$	0
Number of subdiagonals B	3
Subdiagonal factor ζ	1
GGT oversampling, $M_{\mathfrak{r}_\mathfrak{g}} - N$	$0.85N$
bits per vertex sample, Q	11

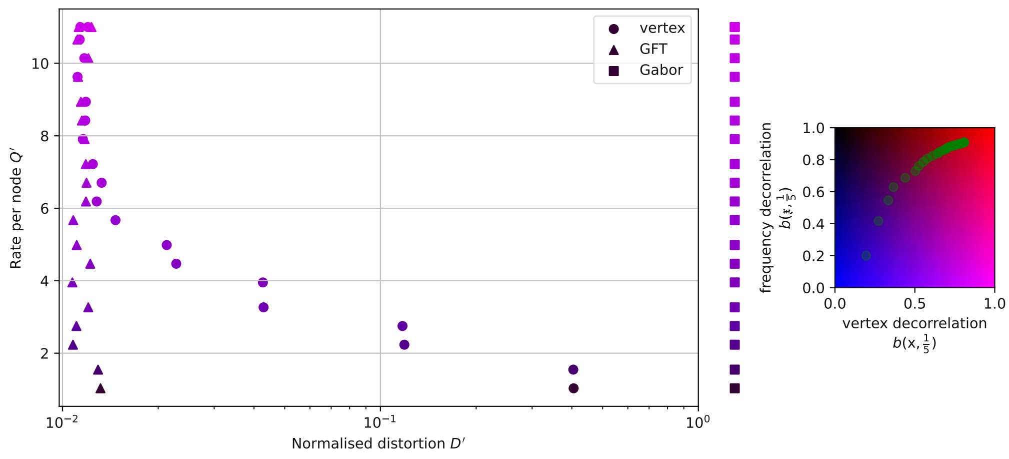
Table 5.6: Parameters for scenario in Section 5.2.1.



(a) Perturbed random regular graphs



(b) Random geometric graphs



(c) Barabási–Albert graphs

Figure 5.7: Rate-distortion comparison for varying K using the parameters from Table 5.6 for the three different graph models. The rate-distortion function yields $D' \in [1.6 \cdot 10^{-8}, 1.1 \cdot 10^{-7}]$.

cause the super- and subdiagonals are also clipped for a bandlimited process. Surprisingly, even at $K = N$, we were unable to obtain fully uncorrelated vertex-domain samples with $B \neq 0$, despite $k_{\text{rms}} = k_{\text{rms,max}} = \sqrt{\frac{(N+1)(2N+1)}{6}}$.

In a first run, we used $\mathbf{U}_{K,\mathcal{S}}$ for vertex-domain compression as described in Section 4.1. Since this led to total destruction (i.e., $D' > 1$ as described in Section 5.1) for this technique, we went for extreme instead of moderate oversampling for the remainder of this section: We dismissed the bandlimited approach and used $\mathbf{U}_{N,\mathcal{S}} = \mathbf{U}$ despite $K < N$ for the remainder of this section[5]. Thus, every signal value is sampled, albeit at a lower rate since the bit budget remains identical (cf. (5.1)).

This fixes vertex-domain compression as shown in Fig. 5.7, at the cost of now having to sample at every node. For $K > \frac{N}{2}$, the performance of the vertex-domain approach approximately matches the GFT approach, which yields the best results.

The normalised distortion of the GFT compression remains almost constant throughout the bandwidth sweep and graph model variation at $D'_{\text{GFT}} \approx 10^{-2}$. As the effective bandwidth equals the nominal bandwidth here, GFT compression always encompasses the full spectrum. Due to $\tau_{\mathfrak{a}} = 0$, the process yields ideal lowpass signals. With Q extra bits that the GFT compression can allocate per additional frequency[6], this frequency (that carries the same power) is sampled at the same bit rate. Like in other scenarios, we observed many runs where the distortion for both vertex and GFT compression was orders of magnitude lower than the mean that is depicted in Fig. 5.7. However, a significant number of outliers that likely suffer from an ill-adjusted quantiser support lowered the mean performance.

Similar to Section 5.1.1, the GGT compression only yields meaningful results for perturbed random regular graphs in Fig. 5.7a, i.e., in a low-coherence setting. The GGT-based approach, however, never outperforms the other techniques here, not even in the case of high bandwidths (and thus high bit budgets). Fig. 5.8 shows that, while the vertex and GFT coefficients become equally sparse in this region, k_{eff} is more than 50 % higher for GGT coefficients.

Graph model	$b\left(\mathfrak{x}_{\mathbf{g},\mathcal{S}}, \frac{1}{5}\right)$
Perturbed random regular	53 %
Random geometric	55 %
Barabási–Albert	42 %

Table 5.7: Percentage of small cross-correlation Gabor coefficients $b\left(\mathfrak{x}_{\mathbf{g},\mathcal{S}}, \frac{1}{5}\right)$, averaged over nominal bandwidth K. Parameters of this scenario are listed in Table 5.6.

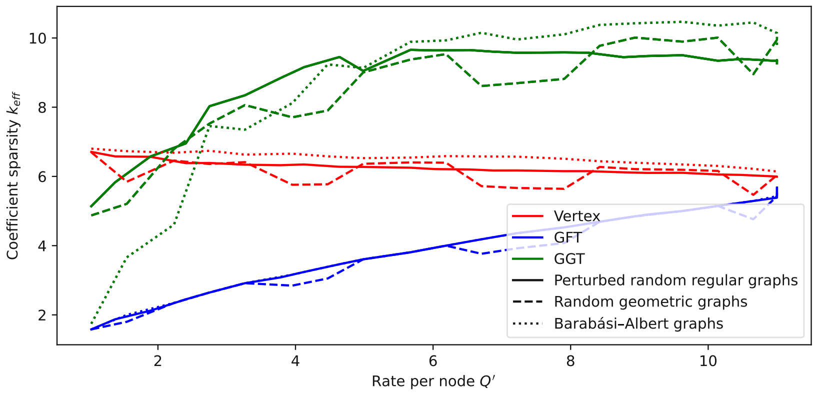


Figure 5.8: Coefficient sparsity k_{eff} versus rate for various transforms using parameters in Table 5.6.

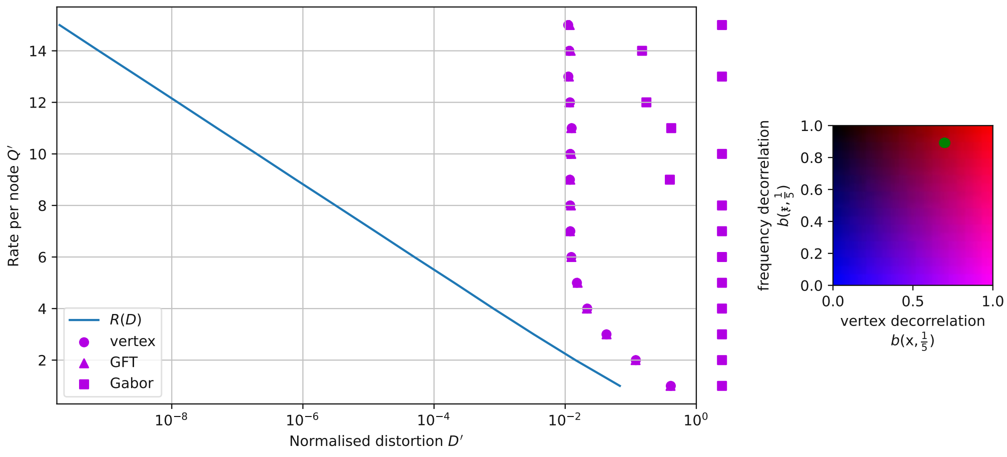
5.2.2 Rate sweep

For the simulations depicted in Fig. 5.9, we kept the bandwidth constant at $K = N$ and chose different sweeping parameter, namely Q, the number of bits per vertex sample. We limit our discussion to random geometric graphs. Since the process parameters stay constant, so does the correlation in the vertex and frequency domain, which is equal to the correlation at $K = N$ in the simulation of Section 5.2.1. The GGT correlation can be found in Table 5.9.

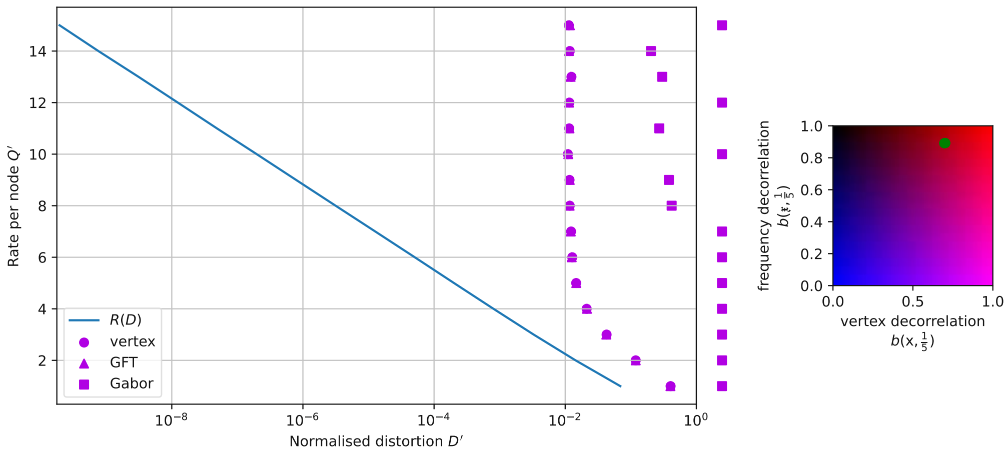
We conducted the simulation three times with different strategies for calculating L, J, i.e., the number of sampled frequencies and nodes for the GGT. In the first scenario, we set $L' = \frac{N}{B}$ while J' grows logarithmically with k_{rms}. The second scenario corresponds to linear decay of L' over B and linear growth of J' over k_{rms}. In the third scenario, we chose L' and J' as their means of the first two scenarios.

[5]Note that substantial, but not extreme oversampling, e.g. $M_{\mathbf{x}} = \min(2K, N)$ yielded promising preliminary results too.

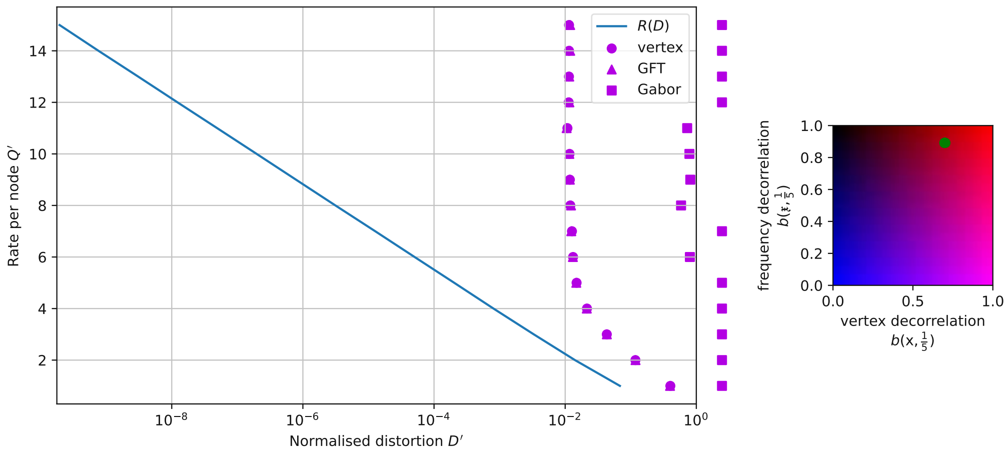
[6]This disregards the non-linear growth of R for $K \lesssim N$, see 5.1.



(a) Nonlinear L, J



(b) Linear L, J



(c) Tradeoff

Figure 5.9: Rate-distortion comparison for varying K using the parameters from Table 5.8.

Parameter	**Value**
graph size, N	64
nominal bandwidth, K	64
graph model	Random geometric
average node degree	3
perturbed regular edges	16.6 %
process generation method	upper band matrix $\mathfrak{B}$
vertex-correlation filter rate, $\tau_{\mathfrak{a}}$	0
Number of subdiagonals B	4
Subdiagonal factor ζ	1
GGT oversampling, $M_{\mathfrak{x}_\mathbf{g}} - N$	$0.85N$
bits per vertex sample, Q	$[1, 16]$

Table 5.8: Parameters for scenario in Section 5.2.2.

L, J	$b\left(\mathfrak{x}_{\mathbf{g},\mathcal{S}}, \frac{1}{5}\right)$
Nonlinear	66 %
Linear	71 %
Tradeoff	70 %

Table 5.9: Percentage of small cross-correlation Gabor coefficients $b\left(\mathfrak{x}_{\mathbf{g},\mathcal{S}}, \frac{1}{5}\right)$, averaged over nominal bandwidth K. Parameters of this scenario are listed in Table 5.8.

Since $L'J' \neq M_{\mathfrak{x}_\mathbf{g}}$, we correct the values to

$$L = \left\lceil \sqrt{N \frac{L'}{J'}} \right\rceil, \qquad J = \left\lceil \sqrt{N \frac{J'}{L'}} \right\rceil,$$

which results in $LJ \gtrsim N$ samples. As shown in the boxplot in Fig. 5.10, we observe the lowest $\kappa\left(\mathbf{\Gamma}_{\mathcal{S}}\right)$, i.e., the condition number of the GGT sampling matrix for the linear choice of L and J. The other simulation parameters remained similar to previous simulations (see Table 5.8).

The distortion induced by vertex and GFT compression is almost identical, as shown in Fig. 5.9. This is in accordance with similar levels of correlation and coefficient sparsity in the two domains. Unlike in the high-rate scenarios described so far, this scenario shows that coming close to the theoretical boundary of the rate-distortion function is possible, at least for vertex- and GFT-domain compression in low-rate settings. Of note is that both techniques converge soon towards $D' \approx 10^{-2}$, with no

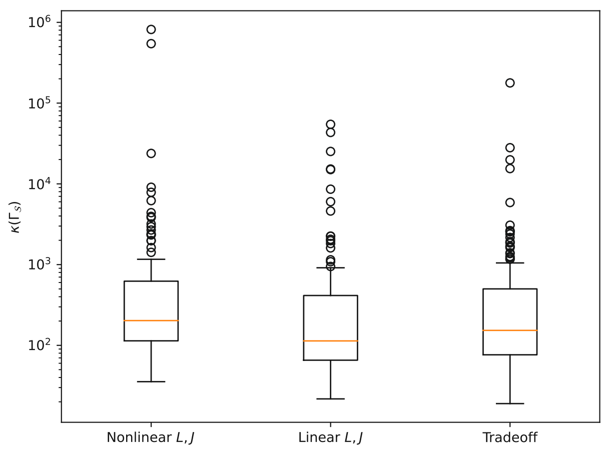


Figure 5.10: Boxplot of $\kappa\left(\mathbf{\Gamma}_{\mathcal{S}}\right)$, the condition number of the GGT sampling matrix, for the different choices of L and J with parameters in Table 5.8.

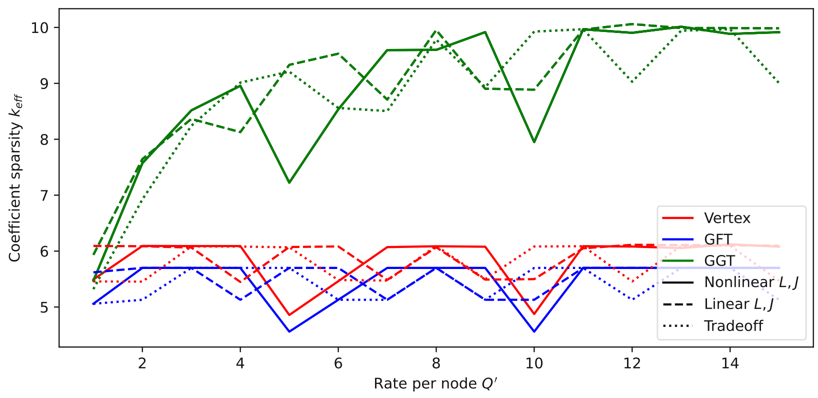


Figure 5.11: Coefficient sparsity k_{eff} versus rate for various transforms using parameters in Table 5.8.

need in spending $R > 400$ bit in this scenario. Again, outliers mainly contribute to this limit since the majority of distortion results is substantially lower than $D' \approx 10^{-2}$. For higher rates, better adjustment of the quantiser to the process would be needed.

Only at this threshold, the GGT compression (inconsistently) yields results $D' < 1$. Again, the sampled GGT coefficients are considerably less sparse then those of the other two techniques (see Fig. 5.11). At least in this setting with $\zeta = \zeta_{\max}$, no diagonal dominance and $\tau_{\mathfrak{a}} = 0$, the choice of L, J cannot improve the GGT results.

6 Conclusion and outlook

6.1 Conclusion

In this thesis, we presented, analysed and tested various compression techniques for signals on graphs. Since quantisation quality depends on the correlation of samples, we applied various graph transforms to achieve approximately uncorrelated samples in the respective domain; however without the computationally expensive Karhunen-Loève transform. In our simulations, we measured the distortion caused by uniform quantisation, as well as the correlation and sparsity of the transform coefficients in the respective domain. We tested the compression techniques on perturbed random regular graphs, random geometric graphs and Barabási–Albert graphs. Since stationary processes are already perfectly uncorrelated in the frequency domain, we designed nonstationary processes by filtering and modulating white noise as well as with banded frequency correlation matrices.

We investigated a graph signal compression technique proposed in literature that intends to alternately optimise the number of quantiser levels and a sampling filter. We found that the method suffers from numerical instability and high computational complexity. Hence, we did not include it in our presented simulations.

As a first, straightforward method, we sampled the graph signal in the vertex domain and quantised each sample at an identical rate. Recovering the original signal then builds on the idea of bandlimited signal reconstruction. However, numerical results show that it is not sufficient to set the number of vertex-domain samples only slightly higher than the bandwidth; this causes total destruction of the signal when the signal is reconstructed after quantisation. Instead, significant oversampling is necessary to yield meaningful and competitive results, given a process with low vertex-domain correlation.

Our second approach was to perform source coding in the graph frequency domain. We allocated our bit budget incrementally to the graph Fourier coefficients according to maximal reduction in expected quantisation distortion. Due to sparse GFT coefficients, naturally more so for bandlimited processes, the GFT-domain compression compensates potentially high levels of correlation to robustly yield acceptable quantisation results for the signal models considered.

Of note is that compression in either domain approaches the theoretical boundary of the rate-distortion function only in scenarios with a low number of bits per graph node. In many simulated scenarios, however, a significant amount of outliers lead to a mean distortion that is orders of magnitude higher than the distortion in the majority of runs. These outliers are caused by imperfect quantisation that require a more precise adjustment of the quantiser, its support in particular.

For significantly nonstationary processes, we proposed the GGT, motivated by the Gabor transform in classical signal processing. By sampling a WFT according to the (moderate) correlation width in time and frequency, the classical Gabor transform represents the signal with approximately uncorrelated samples while being more adjustable regarding the level of correlation in either domain.

For a generalisation of the Gabor transform on graphs, we used the WGFT that was proposed in literature and matched the sampling grid to the expected correlation in vertex domain and frequency domain. However, the WGFT does not have group structure and is not energy preserving. Thus, it is impossible to find a shifted window matrix for both Gabor analysis and Gabor synthesis. Instead, for the latter, we had to opt for the computationally expensive pseudoinverse of the Gabor analysis shifted window matrix.

Apart from these undesirable properties of the WGFT with regard to the GGT, the GFT did not conserve certain properties of the DFT that facilitate the generation of nonstationary processes. This resulted in poorly predicted correlation levels, in a delicate choice of the Gabor sampling grid and the window for the WGFT and thus in a rather fragile GGT transform.

We used an incremental bit allocation for the GGT, similar to the bit allocation for GFT-domain compression. In ‘mild’ compression scenarios, i.e., weakly correlated processes on low-coherence graphs and at high bit rates, the GGT yielded the best results when tuned accordingly. Due to a lack of coefficient sparsity, sensitive tuning and undesirable WGFT properties, the GGT lacks performance in many scenarios, especially for low bit budgets.

6.2 Outlook

Apart from the above-mentioned quantiser adjustments, future work could include different generation methods for nonstationary processes that control correlation in vertex domain and frequency domain in a more reliable fashion by considering properties of the GFT. These novel processes could then be used for a broad range of simulations of the introduced compression techniques.

Furthermore, the performance of the GGT could perhaps be improved by a more accurately matched GGT sampling grid. Selecting the grid based on the actual correlation (instead of process parameters) and based on a joint optimisation of the sampling grid and the WGFT window both appear promising.

Based on our findings, a more fundamental approach would be to formulate a new transform that features more advantageous properties than the above-described. All compression techniques require the GFT matrix of the entire graph, which is computationally expensive since the basis vectors do not have finite support. Furthermore, as the WGFT frames are not tight [5, Section 6.7.2], a novel transform that features such frames could lead to more precise results at a lower computational complexity.

A Python scripts

The following scripts were created by the author. Note that the parts where figures are created are not included. Instead, this appendix focuses on data processing and the calculations required for the analyses performed in this thesis. The software products required to run these Python scripts are listed in Tab. A.1. Note that many fundamental graph signal processing techniques presented in Section 3, as well as methods to plot signals on graphs have been implemented in the Python package PyGSP. Necessary fixes of existing implementations are covered in our fork of this package that can be found under `https://github.com/SP-TUW/pygsp`. On the following pages, we listed Python scripts that we wrote for methods not covered in PyGSP.

Product	Version
Python	3.10
NumPy	1.23.5
SciPy	1.13.0
Scikit-learn	1.1.3
PyGSP	own fork
CVXPY	1.4.1

Table A.1: Required modules/packages for the Python scripts of this appendix

A.1 Simulations

Listing A.1: Main script for simulations in Chapter 5

```python
import numpy as np
from pygsp import graphs
from gsp_decorr_package.utils import is_pos_semi_def, is_diag_domin, perturb, uniform_quantizer, \
    golden_section_search, rate_distortion
from gsp_decorr_package.processes import gft_band, innovations
from gsp_decorr_package.ggt import wft_atoms, sampling_idx_ggt
from gsp_decorr_package.sampling import sampling_ggt, reconstruct_ggt, sampling_vertex, sampling_gft, \
    rate_gft, rate_ggt, reconstruct_vertex
```

```python

if __name__ == "__main__":
    rng = np.random.default_rng()

    N = int(np.sqrt(64)) ** 2  # number of nodes
    degree = 3
    perturbed_edges_percentage = 1 / 6
    # gtype = "Sensor"  # coherence [0.7, 0.85], outlier at 0.9
    # gtype = "RegularPerturbed"        # coherence [0.45, 0.76]
    gtype = "Barabasi-Albert"  # coherence [0.81, 0.96]

    process = "gft_band"
    # process = "filter"

    K = int(1 * N)
    # for process = 'gft_band'
    # number of additional diagonals on each side of the main diagonal (tridiagonal matrix: B=1)
    B = 3  # (2 * K) // 5
    diag_domin = False

    # additional diagonal i will be multiplied by factor**i
    zeta = 1
    # rate of the exponential decay of the power per frequency
    tau_hat_signal = 0.00  # 0.07       #np.log(10 ** -14) / (-4 * K)  # 0.17       #
    # upper limit at np.log(10**-16)/(-4*K), otherwise cov_x_frak is rank-deficient
    assert tau_hat_signal <= np.log(10 ** -16) / (-4 * K)
    # correlation coefficient of the Gabor covariance matrix should be less than 1/theta
    theta = 5
    # high beta corresponds to linear decay of L over B and linear growth of J over k_rms (cf. Sec. 5.2.2)
    beta = 0.3

    Q = 11  # number of bits/sampled node
    # level of oversampling for GGT, i.e., M_x_g/N, must be >1 (e.g., 1.1)
    alpha = 1.85  # 1.2 seems to low, 1.9 better
    # sampling_ggt_mode = 'regular'
    # sampling_ggt_mode = 'idx_fq_rework'
    # sampling_ggt_mode = 'idx_vertex_rework'
    sampling_ggt_mode = 'idx_fq_vertex_rework'
    eps = 0.05
    iterations = 15
    reshape_order = 'C'

    eta = 2  # controls support of uniform quantizer and thus overloading probability (for eta=2 5% for ~N)
    cond_max = np.inf

    graph_realisations = 10
    signal_realisations = 5 * N

    for K in np.linspace(6, 64, num=20):  # for Q in range(1, 16):
        K = int(K)
        # size of vertex sampling set: M=|S|, necessary condition for signal recovery: P>=K
        # maximum size of joint q+s sampling set
        M = int(np.min((K * 1.1, N)))  #
        # oversample in vertex domain? set K_vertex>K and/or M_vertex>M
        K_vertex = N
        M_vertex = N
        R = M * Q
        print(f'N={N}, K={K}, Q={R}')
        Q_prime = R / N  # effective number of bits/graph node, gen. not int

        fail_count = np.zeros(2, dtype=int)
        signal_energy = np.zeros(shape=(graph_realisations, signal_realisations))
        condition_number = np.zeros_like(signal_energy)
        distortion = np.zeros(shape=(3, graph_realisations, signal_realisations))
        b = np.zeros(shape=(3, graph_realisations))
        coherence = np.zeros(shape=graph_realisations)
        RD = np.zeros(shape=graph_realisations)
        D = np.zeros(shape=graph_realisations)
        kappa_Gamma = np.zeros(shape=graph_realisations)
        k_eff = np.zeros_like(b)
        connected_graph_idx = []
        for i in range(graph_realisations):
            # generate graphs
            match gtype:
                case "Sensor":
                    G = graphs.Sensor(N=N, k=degree, distributed=True)  # generate sensor graph
                    p = None
                case "RegularPerturbed":
                    G = graphs.RandomRegular(N=N, k=degree)  # generate regular graph
                    # generate a perturbation regular graph
                    G = perturb(G, perturbed_edges_percentage=perturbed_edges_percentage, rng=rng)
                    p = None
                case "Barabasi-Albert":
                    G = graphs.BarabasiAlbert(N=N)
                    p = G.d / np.sum(G.d)
                case _:
                    # default case
                    raise ValueError("Graph model not covered yet!")
            if not G.is_connected():
                signal_energy[i] = np.nan
                condition_number[i] = np.nan
                distortion[:, i, :] = np.nan
                b[:, i] = np.nan
                coherence[i] = np.nan
                RD[i] = np.nan
                D[i] = np.nan
                kappa_Gamma[i] = np.nan
                k_eff[:, i] = np.nan
                continue
            else:
                connected_graph_idx.append(i)

            G.compute_fourier_basis()  # compute G.U
            # generate process
            match process:
                case "gft_band":
                    cov_x, L, J = gft_band(G=G, rng=rng, K=K, B=B, tau_hat_signal=tau_hat_signal, zeta=zeta,
                                           diag_domin=diag_domin, beta=beta)
                case "filter":
                    cov_x, L, J = innovations(G=G, K=K, beta=beta, tau_a=tau_hat_signal)
                case _:
                    # default case
                    raise ValueError("Signal model not covered yet!")

            assert is_pos_semi_def(cov_x)

            sigma2_eq = np.linalg.eig(cov_x)[0]
            if np.max(np.abs(np.imag(sigma2_eq))) < 1e-9:
                sigma2_eq = np.real(sigma2_eq)
            sigma2_eq = sigma2_eq[sigma2_eq > 1e-15]
            Xi_l0 = 1e-10
            Xi_r0 = np.max(sigma2_eq)

            # find mu that yields the rate from the RD function that is closest to R
            Xi_opt, __ = golden_section_search(rate_distortion, Xi_l0, Xi_r0, 1e-10, 80, 2, sigma2_eq, R)
            # get the rate distortion pair from the RD function
            D[i], RD[i], __ = rate_distortion(Xi_opt, sigma2_eq, R)

            # generate signal
            x = rng.multivariate_normal(mean=np.full(G.N, fill_value=0), cov=cov_x, size=signal_realisations)
            signal_energy[i] = np.sum(x ** 2, axis=-1)

            cov_x_frak = G.U.T @ cov_x @ G.U
            assert is_pos_semi_def(cov_x_frak)
            coherence[i] = G.coherence

            # GFT DOMAIN
            R_i_gft = rate_gft(cov_x_frak=cov_x_frak[:K, :K], R=R, eta=eta)
            idx_gft = np.where(R_i_gft)[0]

            # GGT DOMAIN
            # generate grid
            i_m_breve, k_m_caron = sampling_idx_ggt(G=G, alpha=alpha, J=J, L=L, mode=sampling_ggt_mode)
            # Sampling using Gabor analysis
```

```python
        # optimise window parameters to grid
        tau_g_hat_l0 = 0.05  # 0.02
        tau_g_hat_r0 = 40

        tau_g_hat_opt, kappa_Gamma[i] = golden_section_search(wft_atoms,
                                                              tau_g_hat_l0,
                                                              tau_g_hat_r0,
                                                              eps,
                                                              iterations,
                                                              1,  # we are interested in 2nd argument of wft_atoms
                                                              G, i_m_breve, k_m_caron,
                                                              reshape_order)  # remaining wft_atoms arguments
        # calculate optimised, sampled WFT atoms
        Gamma_S, __ = wft_atoms(tau_g_hat=tau_g_hat_opt, G=G, i_m_breve=i_m_breve, k_m_caron=k_m_caron,
                                reshape_order=reshape_order)
        # calculate covariance matrix of Gabor coefficients
        cov_x_frak_g_S = Gamma_S.T @ cov_x @ Gamma_S
        R_i_ggt = rate_ggt(cov_x_frak_g_S=cov_x_frak_g_S, R=R, eta=eta)
        idx_ggt = np.where(R_i_ggt)[0]

        # check for valid all cov: percentage where C_kl**2 < C_kk*C_ll for k!=l
        __, b[0, i] = is_diag_domin(cov_x, factor=theta)
        k_eff[0, i] = np.sqrt(np.sum(np.arange(cov_x.shape[0]) * np.sort(np.diag(cov_x))) / np.sum(np.diag(cov_x)))

        __, b[1, i] = is_diag_domin(cov_x_frak[np.ix_(idx_gft, idx_gft)], factor=theta)
        k_eff[1, i] = np.sqrt(
            np.sum(np.arange(idx_gft.shape[0]) * np.sort(np.diag(cov_x_frak[np.ix_(idx_gft, idx_gft)]))) / np.sum(
                np.diag(cov_x_frak[np.ix_(idx_gft, idx_gft)])))
        __, b[2, i] = is_diag_domin(cov_x_frak_g_S[np.ix_(idx_ggt, idx_ggt)], factor=theta)
        k_eff[2, i] = np.sqrt(
            np.sum(np.arange(idx_ggt.shape[0]) * np.sort(np.diag(cov_x_frak_g_S[np.ix_(idx_ggt, idx_ggt)]))) /
            np.sum(np.diag(cov_x_frak_g_S[np.ix_(idx_ggt, idx_ggt)])))

        for j in range(signal_realisations):
            # sampling methods
            # VERTEX DOMAIN
            x_S, S_vertex, condition_number[i, j] = sampling_vertex(G=G, x=x[j], rng=rng, K=K_vertex, M=M_vertex,
                                                                    p=p, condition_number_threshold=cond_max,
                                                                    ignore_rank=True)
            R_i_vertex = np.zeros_like(x_S, dtype=int)
            R_i_vertex[S_vertex] = int(R / M_vertex)

            # GFT DOMAIN
            x_frak_S = sampling_gft(G=G, x=x[j], R_i_gft=R_i_gft)
            # GGT DOMAIN
            x_frak_g_S = sampling_ggt(x[j], Gamma_S)

            # quantisation: uniform quantisation of y_s
            # VERTEX DOMAIN
            x_hat_S, __, __ = uniform_quantizer(x_S,
                                                R_i_vertex,
                                                cov_x,
                                                eta,
                                                np.array([True, True, False]))

            # GFT DOMAIN
            x_frak = G.gft(x[j])
            x_frak_hat = np.zeros_like(x_frak_S)
            x_frak_hat[:K], __, __ = uniform_quantizer(x_frak[:K], R_i_gft, cov_x_frak[:K, :K], eta,
                                                       np.array([True, False, False]))

            # GGT DOMAIN
            x_frak_g_S_hat, __, __ = uniform_quantizer(x_frak_g_S, R_i_ggt, cov_x_frak_g_S, eta,
                                                       np.array([True, False, False]))

            # Reconstruction
            x_hat_vertex = reconstruct_vertex(G=G, K=K_vertex, S_vertex=S_vertex, x_s=x_hat_S)
            x_hat_gft = G.igft(x_frak_hat)
            x_ggt_synth = reconstruct_ggt(Gamma_S=Gamma_S, x_frak_g=x_frak_g_S)
            x_hat_ggt = reconstruct_ggt(Gamma_S=Gamma_S, x_frak_g=x_frak_g_S_hat)

            # distortion calc
            distortion[0, i, j] = np.sum((x[j] - x_hat_vertex) ** 2)
            distortion[1, i, j] = np.sum((x[j] - x_hat_gft) ** 2)
            distortion[2, i, j] = np.sum((x[j] - x_hat_ggt) ** 2)

    # distortion mean and variance for every technique
    mean_distortion = np.nanmean(distortion, axis=(1, 2)) / np.nanmean(signal_energy)
    var_distortion = np.nanvar(distortion, axis=(1, 2)) / (np.nanmean(signal_energy) ** 2)

    mean_uncorr = np.nanmean(b, axis=-1)
    mean_k_eff = np.nanmean(k_eff, axis=-1)
```

A.2 Nonstationary processes

Listing A.2: Generation of nonstationary processes (cf. Section 3.4)

```
import numpy as np
from gsp_decorr_package.utils import band_matrix
from pygsp import graphs


def gft_band(G: graphs.Graph,
             rng: np.random.Generator,
             K: int,
             B: int,
             tau_hat_signal: float,
             zeta: float,
             diag_domin: bool,
             beta: float):
    # (non)stationary graph process - 3.4.2 Class II Processes: cov_x_frak = band matrix
    # stationary for B = 0 (cov_x_frak = diag), nonstationary otherwise
    if diag_domin:
        # create first sub/superdiagonal
        diag_1 = np.pad(np.exp(-tau_hat_signal * np.arange(K - 1)), (0, G.N - K), 'constant', constant_values=0)
        v_up = []
        # epsilon
        diag_0 = np.pad(np.abs(rng.normal(0, 1 / 4 * diag_1, G.N - 1)), (0, 1), 'constant', constant_values=0)
        diag_0[K - 1] = np.abs(rng.normal(0, 1 / 4 * diag_1[K - 2]))
        for j in range(0, B):
            # compute sub/superdiagonals (factor*multiplied_diagonal_entries)
            diag_j = zeta ** j * np.pad(np.exp(-tau_hat_signal * np.arange(K - 1 - j)),
                                        (0, G.N - K), 'constant', constant_values=0)
            # add sub/superdiagonal to list of diagonals
            v_up.append(diag_j)
            # add sub/superdiagonal entries to main diagonal
            diag_0 += np.pad(diag_j, (j + 1, 0), 'constant', constant_values=0)
            diag_0 += np.pad(diag_j, (0, j + 1), 'constant', constant_values=0)
        # make sure DC component is the largest
        diag_0[0] = 1.1 * np.max(diag_0)
        # add main diagonal to list of diagonals
        v_final = [diag_0] + v_up

        # create band matrix
        cov_x_frak = band_matrix(v_up=v_final, one_sided=False)
    else:
        # set diagonal of frequency covariance matrix
        diag_0 = np.pad(np.exp(-tau_hat_signal / 2 * np.arange(K)), (0, G.N - K), 'constant', constant_values=0)
        '''
        # frequency power in code of Li2022
        diag_0 = np.sqrt(np.pad(np.array([35, 3.00, 2.94, 1.62, 0.88, 0.83, 0.72, 0.63, 0.52, 0.47, 0.45, 0.4,
     0.37,
                                            0.34, 0.33, 0.3, 0.29, 0.28, 0.26, 0.25]),
                                  (0, G.N - K), 'constant', constant_values=0))
        '''
        # initialize list of diagonals
        v_up = [diag_0]
        for j in range(1, B + 1):
            # add non-main diagonals (factor*multiplied_diagonal_entries)
            v_up.append(zeta ** j * diag_0[:-j])
            # v_up.append(zeta ** j * np.diag(np.einsum('i,j->ij', diag_0, diag_0), k=j))
        # create upper band matrix
        B = band_matrix(v_up=v_up, one_sided=True)
        cov_x_frak = B @ B.T

    # IGFT of frequency covariance matrix
    cov_x = G.U @ cov_x_frak @ G.U.T

    # calculate L_prime based on number of sub/superdiagonals, high beta corresponds to linear decay
    L_prime = int(np.round((1 - beta) * np.ceil(G.N / (B + 1)) + beta * np.ceil(G.N + (-1 + 1 / G.N) * B)))

    k_rms = np.sqrt(np.sum(np.diag(cov_x_frak) * np.arange(1, G.N + 1) ** 2) / np.sum(np.diag(cov_x_frak)))
    len_vertex_lin = 1 / (np.sqrt((G.N + 1) * (2 * G.N + 1) / 6) - 1) * (
            (G.N - 1) * k_rms + np.sqrt((G.N + 1) * (2 * G.N + 1) / 6) - G.N)
    len_vertex_log = (1 - G.N) / np.log((1 / (np.sqrt((G.N + 1) * ((2 * G.N + 1) / 6))))) * np.log(
        np.exp(-np.log(np.sqrt((G.N + 1) * ((2 * G.N + 1) / 6))) / (1 - G.N)) * k_rms)
    J_prime = int(np.round(beta * len_vertex_lin + (1 - beta) * len_vertex_log))

    L = int(np.ceil(np.sqrt(G.N * L_prime / J_prime)))
    J = int(np.ceil(np.sqrt(G.N * J_prime / L_prime)))

    return cov_x, L, J


def innovations(G: graphs.Graph,
                K: int,
                beta: float,
                tau_a: float):
    # nonstationary graph process - 3.4.1 Class I Processes: filtered white noise
    a_frak = np.zeros(G.N)
    a_frak[:K] = np.exp(-tau_a / 2 * np.arange(K))

    k_rms_a = np.sqrt(np.sum(a_frak ** 2 * np.arange(1, G.N + 1) ** 2) / np.sum(a_frak ** 2))
    len_vertex_lin = 1 / (np.sqrt((G.N + 1) * (2 * G.N + 1) / 6) - 1) * (
            (G.N - 1) * k_rms_a + np.sqrt((G.N + 1) * (2 * G.N + 1) / 6) - G.N)
    len_vertex_log = (1 - G.N) / np.log((1 / (np.sqrt((G.N + 1) * ((2 * G.N + 1) / 6))))) * np.log(
        np.exp(-np.log(np.sqrt((G.N + 1) * ((2 * G.N + 1) / 6))) / (1 - G.N)) * k_rms_a)
    # choose J, high beta corresponds to linear growth with k_rms_a
    J = int(np.round(beta * len_vertex_lin + (1 - beta) * len_vertex_log))

    L = int(np.ceil(G.N / J))
    # bandwidth of m
    m_K = int(np.round(J))
    # spectrum of m
    M_nu = np.zeros(G.N)
    M_nu[:m_K] = np.ones(m_K)
    m = G.igft(M_nu)
    m = m * np.sqrt(G.N) / np.linalg.norm(m)
    # final vertex covariance matrix
    cov_x = np.diag(m) @ G.U @ np.diag(a_frak ** 2) @ G.U.T @ np.diag(m).T

    return cov_x, L, J
```

A.3 Compression techniques

Listing A.3: Scripts for compression techniques described in Chapter 4

```python
import numpy as np
from scipy.linalg import sqrtm
import cvxpy as cp
from pygsp import graphs
from .utils import uniform_quantizer
from scipy.special import binom


def sampling_vertex(G: graphs.Graph,
                    x: np.ndarray,
                    rng: np.random.Generator,
                    K: int,
                    M: int,
                    p: np.ndarray | None,
                    condition_number_threshold: float,
                    ignore_rank: bool) -> tuple[np.ndarray, np.ndarray[int], float]:
    for tries in range(int(binom(G.N, M)) + 1):
        # choose sampling set, i.e, sampling indices [0,N), for BarabasiAlbert: higher probability for higher
     degree
        S_vertex = rng.choice(G.N, size=M, replace=False, p=p)
        # extract bandlimited, sampled GFT matrix
        U_K_sampled = G.U[S_vertex, :K]
        if ignore_rank:
            break
        if np.linalg.matrix_rank(U_K_sampled) >= K and np.linalg.cond(U_K_sampled) < condition_number_threshold
     :
            # if U_K_sampled's rank is sufficient and the condition number is not too high, take this GFT
     matrix
            break

    assert tries != binom(G.N, M)  # too may tries
    # construct unquantized, sampled signal by taking the original signal only at the sampling indices
    x_s = np.zeros_like(x)
    x_s[S_vertex] = x[S_vertex]  # set samples

    return x_s, S_vertex, np.linalg.cond(U_K_sampled)


def reconstruct_vertex(G: graphs.Graph, K: int, S_vertex: np.ndarray[int], x_s: np.ndarray):
    # bandlimited reconstruction according to (3.7)
    return G.U[:, :K] @ np.linalg.lstsq(G.U[S_vertex, :K], x_s[S_vertex], rcond=None)[0]


def bit_allocation_mse(G: graphs.Graph, Z: float, M_max: int, beta: np.ndarray, cov_x_frak: np.ndarray, K: int,
                       eta: float) -> np.ndarray[int]:
    # [Li2022, algorithm 2] - modified to allocate actual bits, not levels
    # initialize
    R_i = np.zeros(G.N)
    R = np.log2(Z)
    possible_idx = np.arange(G.N)

    # precompute Laplacian polynomial filter F(Lambda) (matrix_power not necessary since Lambda_i is diagonal)
    Lambda_i = np.diag(G.e[:K])[None, ...] ** np.arange(beta.shape[0])[:, None, None]
    Lambda_i[0] = np.identity(K)
    F_Lambda = np.einsum('ijk,i->jk', Lambda_i, beta)

    # precompute auxiliary matrices
    H = G.U[:, :K] @ F_Lambda @ np.linalg.matrix_power(cov_x_frak, 2)[:K, :K] @ F_Lambda @ G.U[:, :K].T
    X = G.U[:, :K] @ F_Lambda @ cov_x_frak[:K, :K] @ F_Lambda @ G.U[:, :K].T

    cov_x = G.U @ cov_x_frak @ G.U.T  # compute covariance matrix in vertex domain [Li2022, eq. (2)]
    F = G.U[:, :K] @ F_Lambda @ G.U[:, :K].T  # compute graph filter matrix from F(Lambda)
    # support of uniform quantizer according to [Li2022, Def. 4], 5% overloading probability for ~N
    gamma = eta * np.sqrt(np.diagonal(F @ cov_x @ F.T))  # row selection follows in while-loop
    f_tmp = np.zeros_like(R_i, dtype=float)  # initialization for approx. gradient vector [Li2022, eq. (26)]
    while np.sum(R_i) < R:  # step 1
```

```python
        sampling_idx = np.where(R_i)[0]  # step 7, equivalent to np.where(R_i > 0)[0]
        if sampling_idx.shape[0] == M_max:  # step 4
            possible_idx = sampling_idx  # step 5

        # step 2
        for j in possible_idx:
            # increase each R_i by 1 separately -> adapt sampling_idx, compute temporary cov_q, f
            R_i_tmp = R_i.copy()  # R_i_tmp has the role of f({2**(R_i^(k) + 1})
            R_i_tmp[j] += 1
            # sampling_idx_tmp=sampling_idx if sampling_idx.shape[0] == P_max
            sampling_idx_tmp = np.where(R_i_tmp)[0]
            cov_q_tmp = np.diag((gamma[sampling_idx_tmp] / np.power(2, R_i_tmp[sampling_idx_tmp],
                                                                   dtype=float)) ** 2)  # [Li2022, eq. (10)]
            # like adaptive ofdm with constant throughput (incremental BER)
            f_tmp[j] = np.trace(H[np.ix_(sampling_idx_tmp, sampling_idx_tmp)] @ np.linalg.inv(
                X[np.ix_(sampling_idx_tmp, sampling_idx_tmp)] + cov_q_tmp))

        # step 3
        R_i[np.argmax(f_tmp)] += 1  # increase resolution of sample with largest gradient
    return R_i  # step 11


def level_allocation_mse(G: graphs.Graph, Z: float, M_max: int, beta: np.ndarray, cov_x_frak: np.ndarray, K:
     int,
                         eta: float) -> np.ndarray[int]:
    # [Li2022, algorithm 2] - vanilla, allocates quantisation levels, not actual bits
    # initialize
    Z_i = np.ones(G.N)
    sampling_idx = np.where(Z_i > 1)[0]
    possible_idx = np.arange(G.N)

    # precompute Laplacian polynomial filter F(Lambda) (matrix_power not necessary since Lambda_i is diagonal)
    Lambda_i = np.diag(G.e[:K])[None, ...] ** np.arange(beta.shape[0])[:, None, None]
    Lambda_i[0] = np.identity(K)
    F_Lambda = np.einsum('ijk,i->jk', Lambda_i, beta)

    # precompute auxiliary matrices
    H = G.U[:, :K] @ F_Lambda @ np.linalg.matrix_power(cov_x_frak, 2)[:K, :K] @ F_Lambda @ G.U[:, :K].T
    X = G.U[:, :K] @ F_Lambda @ cov_x_frak[:K, :K] @ F_Lambda @ G.U[:, :K].T
    cov_x = G.U @ cov_x_frak @ G.U.T  # [Li2022, eq. (2)]
    F = G.U[:, :K] @ F_Lambda @ G.U[:, :K].T
    # support of uniform quantizer according to [Li2022, Def. 4], 5% overloading probability for ~N
    gamma = eta * np.sqrt(np.diagonal(F @ cov_x @ F.T))  # row selection follows in while-loop
    f = 0  # initialization for objective [Li2022, eq. (25)], since sampling_idx is empty
    q = np.zeros_like(Z_i, dtype=float)  # initialization for approx. gradient vector [Li2022, eq. (26)]
    while np.sum(np.log2(Z_i)) < np.log2(Z):  # step 1
        # step 2
        for j in possible_idx:
            # increase each Z_i by 1 separately -> adapt sampling_idx, compute temporary cov_q, f
            tmp = Z_i.copy()  # tmp has the role of f({M_tilde_i^(k) + 1_i=j})
            tmp[j] += 1
            sampling_idx_tmp = np.where(tmp > 1)[0]
            cov_q_tmp = np.diag((gamma[sampling_idx_tmp] / tmp[sampling_idx_tmp]) ** 2)  # [Li2022, eq. (10)]
            f_tmp = np.trace(H[np.ix_(sampling_idx_tmp, sampling_idx_tmp)] @ np.linalg.inv(
                X[np.ix_(sampling_idx_tmp, sampling_idx_tmp)] + cov_q_tmp))
            # calculate approx. gradient vector [Li2022, eq. (26)]
            q[j] = f_tmp - f
        # step 3
        Z_i[np.argmax(q)] += 1  # increase resolution of sample with largest gradient
        if sampling_idx.shape[0] == M_max:  # step 4
            possible_idx = sampling_idx  # step 5
        else:  # step 6
            sampling_idx = np.where(Z_i > 1)[0]  # step 7
            # only recalculate matrices if sampling_idx might have been modified
            cov_q = np.diag((gamma[sampling_idx] / Z_i[sampling_idx]) ** 2)  # [Li2022, eq. (10)]
            f = np.trace(
                H[np.ix_(sampling_idx, sampling_idx)] @ np.linalg.inv(X[np.ix_(sampling_idx, sampling_idx)] +
     cov_q))
    # step 11, do we have to return sampling_idx as well? sampling_idx = np.where(Z_i > 1)[0]
    return np.log2(Z_i).astype(np.uint64)


```

```python
def filter_mse(G: graphs.Graph, R_i: np.ndarray[np.uint64], cov_x_frak: np.ndarray, K: int, eta: float,
               solver: str) -> np.ndarray:
    # [Li2022, algorithm 3]
    sampling_idx = np.where(R_i)[0]  # uniquely determined by R_i
    condition_number = np.linalg.cond(G.U[sampling_idx, :K])
    lambda_tilde = np.ones(G.N)  # initialize
    lambda_hat = lambda_tilde ** 2  # [Shen2018, algorithm 1 - step 0]
    T = 25  # maximum number of iterations
    threshold = 0.05  # threshold*np.linalg.norm(lambda_tilde)=eps
    for k in range(T):  # step 1
        # store previous lambda_tilde to compute change
        lambda_tilde_old = lambda_tilde.copy()
        # m_{p,q} for all p and q
        m = (1 + np.identity(sampling_idx.shape[0]) * (2 * eta ** 2) / (
                3 * np.power(2, R_i[sampling_idx], dtype=float) ** 2))
        # 3D array: B in |R**(sampling_idx.shape[0] x sampling_idx.shape[0]) for all i
        B = np.einsum('ij,kj,j,ik->ijk', G.U[sampling_idx, :], G.U[sampling_idx, :], np.diag(cov_x_frak), m)
        # optimize each sampled lambda_tilde separately
        for j in range(sampling_idx.shape[0]):  # step 2
            # step 3
            B_sqrt_inv = np.linalg.inv(sqrtm(B[sampling_idx[j]]))
            # correct numerical inaccuracies of sqrtm
            if 0 <= np.max(np.abs(np.imag(B_sqrt_inv))) < 1e-7:
                B_sqrt_inv = np.real(B_sqrt_inv)

            # split up einsum to exclude sampling_idx[j]
            C = np.einsum('jik,j->ik', B[:sampling_idx[j]], lambda_tilde[:sampling_idx[j]] ** 2) \
                + np.einsum('jik,j->ik', B[sampling_idx[j] + 1:], lambda_tilde[sampling_idx[j] + 1:] ** 2)
            # precompute auxiliary matrices for [Li2022, eq. (29)]
            Lambda_x, U_x = np.linalg.eigh(B_sqrt_inv @ C @ B_sqrt_inv)
            A = U_x.T @ B_sqrt_inv @ G.U[sampling_idx, :] @ cov_x_frak
            # maximum number of iterations for optimizing [Li2022, Lemma 3] at sampling_idx[j]
            T_in = 100
            for k in range(T_in + 1):  # while True:
                if k == T_in:
                    raise RuntimeError(f"Optimization in Lemma 3 reached {T_in} iterations.", condition_number)
                A_modified = A.copy()
                A_modified[:, sampling_idx[j]] = 0
                # convex optimization lambda_hat according to [Shen2018, algorithm 1 - step 3]
                x = cp.Variable(nonneg=True, value=lambda_hat[sampling_idx[j]])

                # eq. (29)
                a_j = A[:, sampling_idx[j]] ** 2 * Lambda_x - (A_modified ** 2 @ lambda_hat)

                # numerator and denominator of maximisation do not have the same sign
                sign_wrong_idx = np.sign(-a_j) != np.sign((lambda_hat[sampling_idx[j]] + Lambda_x))
                # in indices where they do not have the same sign: is numerator~0 and denominator positive?
                zero_case = np.logical_and(np.abs(a_j[sign_wrong_idx]) < 1e-13, (lambda_hat[sampling_idx[j]] +
        Lambda_x)[sign_wrong_idx] >0)

                if np.any(sign_wrong_idx==True):
                    if not np.all(zero_case==True):
                        raise ValueError(f'Optimization in Algorithm 3 failed: shen2018 not applicable, ||sqrt(
        B)||={np.linalg.norm(B[sampling_idx[j]])}.',
                                         condition_number)
                # Lemma 3 formulation of eq. (29) is suitable for optimization according to [Shen2018]
                y_j_opt = np.sqrt(-a_j) / (lambda_hat[sampling_idx[j]] + Lambda_x)
                objective = cp.Maximize(
                    cp.sum(cp.multiply(2 * y_j_opt, cp.sqrt(-a_j)) - cp.multiply(y_j_opt ** 2, (x + Lambda_x)))
        )

                prob = cp.Problem(objective)
                # The optimal objective value is returned by `prob.solve()`, optimal value for x is stored in `
        x.value`
                try:
                    # try solving the optimization problem
                    __ = prob.solve(solver=solver, verbose=True)
                except cp.error.SolverError as e:
                    # if function fails, raise an error with the error message and the corresponding kappa(
        U_K_sampled)
                    raise ValueError(e, condition_number)

                lambda_hat_old = lambda_hat[sampling_idx[j]]

                # prob.value ... minimum value of the objective
                # For maximization problems the optimal value is -inf if infeasible and inf if unbounded.
                print(f'lambda_hat_opt = {x.value}')
                if prob.value == np.inf:  # equivalent to x.value is None
                    # problem is unbounded
                    raise ValueError(f'Problem is unbounded: lambda_hat[{sampling_idx[j]}] optimized to nan',
                                     condition_number)
                elif x.value is None:
                    raise ValueError(f'Other error: lambda_hat[{sampling_idx[j]}] optimized to nan', condition_number)

                lambda_hat[sampling_idx[j]] = x.value

                # correct numerical inaccuracies of the cvxpy
                if -1e-10 < lambda_hat[sampling_idx[j]] < 0:
                    lambda_hat[sampling_idx[j]] = 0
                if np.linalg.norm(lambda_hat[sampling_idx[j]] - lambda_hat_old) <= threshold * np.linalg.norm(
                        lambda_hat_old):
                    # if optimized lambda_hat[sampling_idx[j]] changes only little from one iteration to another, stop
                    break
            lambda_tilde = np.sqrt(lambda_hat)
        if np.linalg.norm(lambda_tilde - lambda_tilde_old) <= threshold * np.linalg.norm(lambda_tilde_old):  # step 5
            # if lambda_tilde changes only little from one iteration to another, stop
            break  # step 6
    return lambda_tilde  # step 9


def sampling_idx_mse(G: graphs.Graph, Z: float, M_max: int, P: int, cov_x_frak: np.ndarray, eta: float, K: int,
                     solver: str, bit_allocation: bool) -> \
        tuple[np.ndarray, np.ndarray[int]]:
    # [Li2022, algorithm 4]
    beta = np.zeros(P)  # initializing coefficients the Laplacian polynomial filter F(lambda)
    beta[0] = 1  # equivalent to F_Lambda = np.identity(G.K) -> F=I
    F = np.identity(G.N)  # equivalent to G.U @ F_Lambda @ G.U[:, :K].T
    threshold = 0.05  # threshold*np.linalg.norm(F)=eps
    T = 25  # maximum number of iterations
    for k in range(T + 1):  # while True:
        if k == T:
            sampling_idx = np.where(R_i)[0]  # uniquely determined by R_i
            condition_number = np.linalg.cond(G.U[sampling_idx, :K])
            raise RuntimeError(f"Algorithm 4 reached {T} iterations", condition_number)
        F_old = F
        if bit_allocation:
            R_i = bit_allocation_mse(G, Z, M_max, beta, cov_x_frak, K, eta)  # step 2 modified
        else:
            R_i = level_allocation_mse(G, Z, M_max, beta, cov_x_frak, K, eta)  # step 2 vanilla
        lambda_tilde = filter_mse(G, R_i, cov_x_frak, K, eta, solver)  # step 3
        F = G.U @ np.diag(lambda_tilde) @ G.U.T  # algorithm 3, step 10

        if np.linalg.norm(F_old - F) <= threshold * np.linalg.norm(F_old):  # step 5
            break
        else:
            # if another iteration is needed, approximate Laplacian polynomial filter coefficients for F(lambda) from
            # algorithm 3
            # since F(lambda) and Lambda**i are diagonal matrices, we can express them as vectors and apply LS
            Lambda_i_diagonal = (G.e[None, ...] ** np.arange(beta.shape[0])[:, None]).T
            beta = np.linalg.lstsq(Lambda_i_diagonal, lambda_tilde, rcond=None)[0]
    return F, R_i  # step 6


def sampling_mse(F: np.ndarray, x: np.ndarray, R_i_mse: np.ndarray[int]):
    # sampling according to [Li2022]
    # deduce sampling set from the bit allocation
    sampling_idx = np.where(R_i_mse)[0]
    # initialize filtered signal
    y_s = np.zeros_like(x)
    # filter vertex-domain signal with calculated Laplacian polynomial filter
    y_s[sampling_idx] = F[sampling_idx, :] @ x
    return y_s


def reconstruct_mse(G: graphs.Graph, F: np.ndarray, cov_x: np.ndarray, cov_x_frak: np.ndarray, cov_q: np.ndarray,
                    K: int, sampling_idx: np.ndarray[int], y_s: np.ndarray):
    # reconstruction according to [Li2022]
    Gamma_opt = cov_x_frak[:K, :K] @ np.linalg.inv(np.diag(np.diag(cov_x_frak)[:K])) @ G.U[:, :K].T  # for cov_obs=0
    Phi_opt = Gamma_opt @ cov_x @ F[sampling_idx, :].T \
              @ np.linalg.inv(
        F[sampling_idx, :] @ cov_x @ F[sampling_idx, :].T + cov_q[np.ix_(sampling_idx, sampling_idx)])
    return G.U[:, :K] @ Phi_opt @ y_s[sampling_idx]


def rate_gft(cov_x_frak: np.ndarray[float], R: int, eta: float) -> np.ndarray[int]:
    # find bandwidth using the length of the signal in the GFT domain
    K = cov_x_frak.shape[0]
    # initialize the bit allocation to zero
    R_i = np.zeros(K, dtype=int)

    # start bit allocation process again if the bit limit has not been reached
    while np.sum(R_i) < R:
        # current sampling indices
        sampling_idx = np.where(R_i)[0]
        # number of samples
        num_samples = sampling_idx.shape[0]
        # Less power is assigned to higher frequencies, i.e. cov_x_frak is monotonically decreasing. Thus, the next bit can
        # belong to either an already sampled or the next unsampled frequency, so there are num_samples + 1 options.
        num_possible_samples = np.min((num_samples + 1, K))
        # initialize array with expected quantization error
        dist_q_exp = np.zeros(num_possible_samples)
        for i in range(num_possible_samples):
            # repeat these steps for all possible frequencies
            # copy the status quo of bit allocation
            R_i_tmp = R_i.copy()
            # assign a temporary bit to the current frequency
            R_i_tmp[i] += 1
            # calculate only the expected quantization distortion if this extra bit was assigned to this frequency
            __, dist_q_exp[i], __ = uniform_quantizer(None, R_i_tmp, cov_x_frak, eta, np.array([False, True, False]))
        # permanently assign bit to frequency with the lowest quantization error
        R_i[np.argmin(dist_q_exp)] += 1

    # return complete bit allocation
    return R_i


def sampling_gft(G: graphs.Graph, x: np.ndarray, R_i_gft: np.ndarray[int]):
    # sampling in the gft domain
    sampling_idx = np.where(R_i_gft)[0]
    x_frak = G.gft(x)
    y_s = np.zeros_like(x_frak)
    y_s[sampling_idx] = x_frak[sampling_idx]  # set samples

    return y_s


def sampling_ggt(x: np.ndarray, Gamma_S: np.ndarray):
    # Gabor analysis
    return Gamma_S.T @ x


def rate_ggt(cov_x_frak_g_S: np.ndarray[float], R: int, eta: float) -> np.ndarray[int]:
    # find M (number of samples) using the shape of the cov matrix in the GGT domain
    M = cov_x_frak_g_S.shape[0]
    # initialize the bit allocation to zero
    R_i = np.zeros(M, dtype=int)

    # start bit allocation process again if the bit limit has not been reached
    while np.sum(R_i) < R:
        # current sampling indices
        sampling_idx = np.where(R_i)[0]
        # initialize array with expected quantization error
        D_Q_exp = np.zeros(M)
        for i in range(M):
            # repeat these steps for all possible frequencies
            # copy the status quo of bit allocation
            R_i_tmp = R_i.copy()
            # assign a temporary bit to the current frequency
            R_i_tmp[i] += 1
            # calculate only the expected quantization distortion if this extra bit was assigned to this
 frequency
            __, D_Q_exp[i], __ = uniform_quantizer(None, R_i_tmp, cov_x_frak_g_S, eta, np.array([False, True,
 False]))
        # permanently assign bit to frequency with the lowest quantization error
        R_i[np.argmin(D_Q_exp)] += 1

    # return complete bit allocation
    return R_i


def reconstruct_ggt(Gamma_S: np.ndarray, x_frak_g: np.ndarray):
    # Gabor synthesis
    return np.linalg.pinv(Gamma_S).T @ x_frak_g.ravel()
```

A.4 Graph Gabor transform

Listing A.4: GGT-related functions (cf. Section 4.4)

```python
import numpy as np
from pygsp import graphs
from .utils import center_vertex_sampling


def wft_atoms(tau_g_hat: float, G: graphs.Graph, i_m_breve: np.ndarray, k_m_caron: np.ndarray, reshape_order:
 str):
    gamma_hat = np.exp(-tau_g_hat * G.e)
    # normalize in frequency domain
    gamma_hat /= np.linalg.norm(gamma_hat)

    # 4d operator for all translations & modulations, [Shuman16, eq.52]
    mt = G.N * np.einsum('nk,il,nl->ikn1', G.U, np.conjugate(G.U), G.U)
    gamma_ik = mt @ gamma_hat
    Gamma_S = np.reshape(gamma_ik[np.ix_(i_m_breve, k_m_caron, np.ones(G.N, dtype=bool))], (-1, G.N),
                         order=reshape_order).T
    cond = np.linalg.cond(Gamma_S.T)

    # all coefficients: sf_ik = np.einsum('n,ikn->ik', x, gamma_ik)  # equivalent to np.inner(x, gamma_ik)
    # sf_ik[np.ix_(i_m_breve, k_m_caron)].ravel() = Gamma_S.T @ x

    # sample and reshape 4d translation+modulation operator to 3d
    '''
    S_sampled = np.reshape(
        mt[np.ix_(idx_vertex, k_m_caron, np.ones(N, dtype=bool), np.ones(N, dtype=bool))],
        (-1, N, N), order=reshape_order)  # suitable name would be mt_sampled_flat
    '''
    # swap axes of operator
    # S_sampled = np.einsum('jnl->njl', S_sampled)
    # at this point: Gamma_S = S_sampled @ gamma_hat

    return Gamma_S, cond


```

```python
def sampling_idx_ggt(G: graphs.Graph,
                     alpha: float,
                     J: int,
                     L: int,
                     mode: str):
    if alpha > 1:
        # In case of oversampling, increase the more densely sampled domain to hit the desired number of
 samples more
        # precisely. If in that case there would be more than N samples in that domain, oversample in the other
 domain.
        if L >= J:
            if int(np.round(alpha * L)) <= G.N:
                L += int(np.ceil((alpha * G.N - (L * J)) / J))
            elif int(np.round(alpha * J)) <= G.N:
                J += int(np.ceil((alpha * G.N - (L * J)) / L))
        else:
            if int(np.round(alpha * J)) <= G.N:
                J += int(np.ceil((alpha * G.N - (L * J)) / L))
            elif int(np.round(alpha * L)) <= G.N:
                L += int(np.ceil((alpha * G.N - (L * J)) / J))

    match mode:
        case 'regular':
            # sample at equidistant indices
            i_m_breve = np.linspace(0, G.N, J, dtype=int, endpoint=False)
            k_m_caron = np.linspace(0, G.N, L, dtype=int, endpoint=False)
        case 'idx_fq_rework':
            # in frequency domain, sample at equidistant frequencies
            i_m_breve = np.linspace(0, G.N, J, dtype=int, endpoint=False)
            k_m_caron = np.argmin(np.abs(G.e[None, ...]
                                         - (G.e[0] + G.e[G.N-1]) / L * np.arange(L)[..., None]), axis=1)
            for i in range(L - 1):
                if k_m_caron[i] == k_m_caron[i + 1]:
                    # if identical fq indices have been selected, increase one index by one
                    k_m_caron[i + 1] += 1
        case 'idx_vertex_rework':
            # in vertex domain, sample at the cluster centers of spectral clustering
            i_m_breve = center_vertex_sampling(G.A, J)
            k_m_caron = np.linspace(0, G.N, L, dtype=int, endpoint=False)
        case 'idx_fq_vertex_rework':
            # apply 'idx_fq_rework' and 'idx_vertex_rework'
            i_m_breve = center_vertex_sampling(G.A, J)
            k_m_caron = np.argmin(np.abs(G.e[None, ...]
                                         - (G.e[0] + G.e[G.N-1]) / L * np.arange(L)[..., None]), axis=1)
            for i in range(L - 1):
                if k_m_caron[i] == k_m_caron[i + 1]:
                    # if identical fq indices have been selected, increase one index by one
                    k_m_caron[i + 1] += 1
        case _:
            # default case
            raise ValueError("Sampling mode is not covered yet.")

    # remove sampling indices > N
    i_m_breve = i_m_breve[i_m_breve < G.N]
    k_m_caron = k_m_caron[k_m_caron < G.N]
    return i_m_breve, k_m_caron
```

A.5 Auxiliary functions

Listing A.5: More generally applicable auxiliary functions

```python
import numpy as np
from scipy.linalg import eigvalsh
from typing import Callable
from sklearn.cluster import SpectralClustering
from scipy import sparse as sps
from pygsp import graphs


def band_matrix(v_up: list[np.ndarray], one_sided: bool = False) -> np.ndarray:
    # create a band matrix with diagonals v_up
    # v_up: [v_diag, v_diag+1=v_diag-1,...]

    # create diagonal matrix
    matrix = np.diag(v_up[0])
    for i in range(1, len(v_up)):
        # check length of non-main diagonal
        assert v_up[i].shape[0] == v_up[0].shape[0] - i
        # set non-main diagonal in upper and lower triangle
        matrix += np.diag(v_up[i], +i)
        if not one_sided:
            matrix += np.diag(v_up[i], -i)

    return matrix


def is_pos_semi_def(x: np.ndarray) -> bool:
    # check positive definiteness of matrix x (complex Hermitian or real symmetric matrix)

    # calculate all eigenvalues for a complex Hermitian or real symmetric matrix
    eigval_x = eigvalsh(x)
    # sort out potential numerical inaccuracies and check if all eigenvalues are >= 0
    if 0 < np.max(np.abs(np.imag(eigval_x))) < 1e-12:
        eigval_x = np.real(eigval_x)
    return np.all(eigval_x >= -1e-12)


def is_diag_domin(x, factor: float = 10.) -> tuple[bool, float]:
    ''''''
    # check if matrix x is diagonally dominant in one sense or another

    '''
    abs_x = np.abs(x)
    # strictest: factor*sum of absolute off-diagonal elements < diagonal element
    bool_array = factor * np.sum(abs_x-np.diag(np.diag(abs_x)), axis=1) < np.diag(abs_x)
    return np.all(bool_array), np.sum(bool_array) / x.shape[0]

    # less strict: factor*absolute off-diagonal element < minimum of respective diagonal elements
    dx = np.diag(abs_x)
    d0 = np.repeat(dx[:, None], dx.shape[0], axis=1)
    d1 = np.repeat(dx[None, :], dx.shape[0], axis=0)
    bool_array = factor * (abs_x - np.diag(np.diag(abs_x))) < np.minimum(d0, d1)
    '''
    # least strict: correlation coefficient < 1/factor
    bool_array = np.abs(np.diag(np.diag(x) ** (-1 / 2)) @ x @ np.diag(np.diag(x) ** (-1 / 2))
                        - np.identity(x.shape[0])) < 1 / factor
    return np.all(bool_array), (np.sum(bool_array) - x.shape[0]) / (x.shape[0] * x.shape[1] - x.shape[0])


def golden_section_search(g: Callable, x_l0: float, x_r0: float, eps: float, iterations: int = 20, arg: int =
    0, *args):
    # nested intervals to find x* with g(x*)<=g(x) for all x
    a = (np.sqrt(5) - 1) / 2
    x_l = x_l0
    x_r = x_r0
    x_lh = a * x_l + (1 - a) * x_r
    x_rh = a * x_r + (1 - a) * x_l
    for p in range(iterations):
        g_xlh = g(x_lh, *args)[arg]
        g_xrh = g(x_rh, *args)[arg]
        if g_xlh > g_xrh:
            x_l = x_lh
            x_lh = x_rh
            x_rh = a * x_r + (1 - a) * x_l
        else:
            x_r = x_rh
            x_rh = x_lh
            x_lh = a * x_l + (1 - a) * x_r
        if np.abs(x_l - x_r) < eps:
            print(f'Golden section search optimum: g(x*)={g_xlh:.2f} at x*={x_lh:.2f} after {p} iterations')
            break
        if p == iterations - 1:
            print(f'Aborted golden section search after {iterations} iterations!')

    return (x_rh + x_lh) / 2, g_xlh


def center_vertex_sampling(A: sps.csr_matrix, J: int):
    if J == A.shape[0]:
        return np.arange(J)

    # Cluster
    sc = SpectralClustering(J, affinity='precomputed', n_init=100)
    sc.fit(A)
    center_idx = np.zeros(J, dtype=int)
    for r in range(J):
        groupr = sc.labels_ == r
        # create adjacency matrix of current cluster
        A_0 = A[np.ix_(groupr, groupr)]
        # initialise boolean matrix
        A_tmp = sps.lil_matrix((np.sum(groupr), np.sum(groupr)), dtype=bool)

        for w in range(1, np.sum(groupr) + 1):
            # which node can be reached from which node in w hops?
            A_tmp[sps.linalg.matrix_power(A_0, w) != 0] = True
            # ignore hopes back to the same node
            A_tmp[np.eye(A_tmp.shape[0], dtype=bool)] = False
            reached_nodes = np.sum(A_tmp, axis=1)
            if np.any(reached_nodes == np.sum(groupr) - 1):
                # if every node is reachable from a certain node, we call it the center node of this cluster
                center_idx[r] = np.where(groupr)[0][np.where(reached_nodes == np.sum(groupr) - 1)[0][0]]
                break
    # sample at the cluster centers
    i_m_breve = center_idx
    return i_m_breve


def perturb(G: graphs.Graph, perturbed_edges_percentage: float, rng: np.random.Generator | None = None):
    # perturb a graph
    assert 0 <= perturbed_edges_percentage <= 1  # check if perturbed_edges_percentage is in valid range
    Wtmp = G.W.toarray()  # save the graph's weight matrix for modification
    perturbations = int(np.round(G.Ne * perturbed_edges_percentage))  # number of created/deleted edges
    if rng is None:
        rng = np.random.default_rng()
    idx1 = rng.integers(G.N, size=perturbations)  # indices of nodes on one side of the perturbed edges
    idx2 = rng.integers(G.N, size=perturbations)  # indices of nodes on the other side of the perturbed edges
    if np.any(idx1 == idx2):  # avoid self-loops by new choice for idx2
        for i in np.where(idx1 == idx2):
            idx2[i] = rng.choice(np.delete(np.arange(G.N), idx1[i]))

    Wtmp[idx1, idx2] = 1 - Wtmp[idx1, idx2]
    Wtmp[idx2, idx1] = Wtmp[idx1, idx2]

    # return graph with modified weight matrix but same coordinates
    return graphs.Graph(sps.csr_matrix(Wtmp))


def uniform_quantizer(s_unquantized: np.ndarray[float] | None,
                      R_i_S: np.ndarray[int],
                      cov_y_S: np.ndarray[float],
                      eta: float,
                      output_on: np.ndarray[bool] = np.array([True, True, True])) -> tuple[
    np.ndarray[float] | None, float | None, np.ndarray[float] | None]:
    # support of uniform quantizer according to [Li2022, Def. 4], 5% overloading probability for ~N
    gamma = eta * np.sqrt(np.diagonal(cov_y_S))
    # step size of uniform quantizer
    delta = 2 * gamma / np.power(2, R_i_S, dtype=float)

    # if the respective output is not needed, it is set to None
    s_quantized = None
    D_Q = None
    cov_q = None

    # only compute variables if output is needed
    if output_on[0]:
        # quantised graph signal (Uniform Quantisation / Mid-Riser Characteristic) according to [Li2022, Def.
 4]
        s_quantized = np.minimum(
            np.maximum(delta * (np.floor(s_unquantized / delta) + 1 / 2), -(gamma - delta / 2)), (gamma - delta
 / 2))
    if output_on[1]:
        # expected quantizer error
        D_Q = np.mean(delta ** 2 / 12)
    if output_on[2]:
        # matrix G according to [Li2022, eq. (10)]
        cov_q = np.diag((gamma / np.power(2, R_i_S, dtype=float)) ** 2)
    return s_quantized, D_Q, cov_q


def rate_distortion(Xi: float, sigma2_eq: np.ndarray, Q: int):
    # calculate rate distortion function for given Xi and power of equivalent, uncorrelated source according to
 (2.1)
    D_i = np.minimum(Xi, sigma2_eq)
    D = np.sum(D_i)
    RD = np.sum(1 / 2 * np.log2(sigma2_eq / D_i))
    # calculate deviation from a given bit budget
    dist = np.linalg.norm(RD - Q) ** 2
    return D, RD, dist
```

Bibliography

[1] G. Leus, A. G. Marques, J. M. Moura, A. Ortega, and D. I. Shuman, "Graph signal processing: History, development, impact, and outlook," *IEEE Signal Processing Magazine*, vol. 40, no. 4, pp. 49–60, 2023.

[2] T. M. Cover, *Elements of information theory*. Hoboken, NJ: Wiley-Interscience, 2nd ed., 2006.

[3] P. Li, N. Shlezinger, H. Zhang, B. Wang, and Y. C. Eldar, "Graph Signal Compression by Joint Quantization and Sampling," *IEEE Trans. Signal Process.*, vol. 70, pp. 4512–4527, 2022.

[4] W. Kozek, "Matched generalized Gabor expansion of nonstationary processes," in *Proc. 27th Asilomar Conf. Signals, Systems and Computers*, (Pacific Grove, CA), 1993.

[5] D. I. Shuman, B. Ricaud, and P. Vandergheynst, "Vertex-frequency analysis on graphs," *Applied and Computational Harmonic Analysis*, vol. 40, no. 2, pp. 260–291, 2016.

[6] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Springer, 1992.

[7] J. W. Demmel, *Applied Numerical Linear Algebra*. SIAM, 1997.

[8] A. Ortega, *Introduction to Graph Signal Processing*. Cambridge University Press, 2022.

[9] T. Tao, "The Fourier Transform," in *The Princeton Companion to Mathematics* (T. Gowers, J. Barrow-Green, and I. Leader, eds.), ch. III.27, pp. 204–208, Princeton University Press, 2010.

[10] D. Gleich, "The MatlabBGL Matlab library." `https://www.cs.purdue.edu/homes/dgleich/packages/matlab_bgl/`.

[11] R. Ramakrishna, H.-T. Wai, and A. Scaglione, “A User Guide to Low-Pass Graph Signal Processing and Its Applications: Tools and Applications,” *IEEE Signal Processing Magazine*, vol. 37, pp. 74–85, Nov. 2020.

[12] A. Papoulis and S. U. Pillai, *Probability, Random Variables, and Stochastic Processes*. New York: McGraw Hill, 4th ed., 2002.

[13] L. Yu, J. Xie, and X. Zheng, “The relationship between graph Fourier transform (GFT) and discrete cosine transform (DCT) for 1D signal and 2D image,” *Signal, Image and Video Processing*, vol. 17, pp. 445–451, May 2022.

[14] K. Shen and W. Yu, “Fractional programming for communication systems–part i: Power control and beamforming,” *IEEE Trans. Signal Process.*, vol. 66, no. 10, pp. 2616–2630, 2018.

[15] E. K. P. Chong, *An introduction to optimization*. New York: John Wiley & Sons, Inc., 2nd ed., 2001.

List of figures

Nomenclature

Abbreviations

DCT	discrete cosine transform
DFT	discrete Fourier transform
GFT	graph Fourier transform
GGT	graph Gabor transform
IDCT	inverse discrete cosine transform
IDFT	inverse discrete Fourier transform
IGFT	inverse graph Fourier transform
MSE	mean square error
pdf	probability density function
PSD	power spectral density
rms	root mean square
SNR	signal-to-noise ratio
WFT	windowed Fourier transform
WGFT	windowed graph Fourier transform

Symbols

$\mathbf{A}$	adjacency matrix
$\mathfrak{a}$	filter for vertex-domain correlation

B	number of super- and subdiagonals
$b(\mathbf{x}, \theta)$	percentage of off-diagonal correlation coefficients smaller than θ
$\mathfrak{B}$	upper band matrix
$\mathbf{C}_{\mathbf{x}}$	covariance matrix
$\mathbf{D}$	degree matrix
$\mathrm{Diag}(\cdots)$	degree matrix
D	(average) distortion
D'	(average) normalised distortion
$d(\mathbf{x}, \mathbf{y})$	distortion function
$D_{\mathcal{Q}}$	expected quantisation distortion
$\deg(m)$	degree
Δ	Laplace operator
δ	quantiser step size
$\boldsymbol{\delta}_i$	Kronecker delta
$\delta_u(t)$	Dirac delta function shifted by u
$\mathcal{E}$	set of edges
$\mathrm{E}\{\cdot\}$	expectation
$E_{\mathbf{x}}$	mean signal energy
$EA(\tau, \nu)$	expected ambiguity function
$\mathcal{F}$	Fourier transform
F	frequency period for continuous time Gabor transform
$f_N(\mathbf{x})$	encoder

$\mathcal{G}$	graph
$\mathbf{g}_{ik}$	WGFT atom
$\gamma_{iT,kF}(t)$	continuous time Gabor analysis window, translated by iT and modulated by kF
$g_{iT,kF}(t)$	continuous time Gabor synthesis window, translated by iT, modulated by kF
$g_N(\cdot)$	decoder
$g_{\tau\xi}(t)$	WFT atom
γ	(one-sided) quantiser support
$\boldsymbol{\gamma}_{i_{\check{m}}k_{\check{m}}}$	graph Gabor analysis window, translated by $i_{\check{m}}$, modulated by $k_{\check{m}}$
$\boldsymbol{\Gamma}_{\mathcal{S}}$	graph Gabor analysis matrix
$\mathbf{H}$	matrix of a linear graph filter
$\mathbf{h}$	filter transfer function
$\odot$	Hadamard product (i.e., component-wise multiplication)
$\mathbf{I}_N$	identity matrix
$\mathrm{I}\{\cdot\}$	indicator function
$\mathtt{I}$	interpolation procedure
$I(\mathsf{x},\mathsf{y})$	mutual information
J	number of sampled nodes for GGT
$j=\sqrt{-1}$	imaginary unit
k_{eff}	coefficient sparsity quantity
k_{rms}	vertex-domain correlation quantity
$\kappa(\cdot)$	condition number

L	number of sampled frequencies for GGT
λ	eigenvalue, graph frequencies
$\mathbf{L}$	Laplacian matrix
$\tilde{\mathbf{L}}$	normalized graph Laplacian
$\mathbb{M}_k\mathbf{x}$	generalised modulation of $\mathbf{x}$ by k
$\mathbf{m}$	filter for frequency-domain correlation
M	number of samples
$\left(\tilde{\mathbb{M}}_\xi x\right)(t)$	modulation of $x(t)$ by ξ
μ	coherence
$\mathcal{N}(\cdot,\cdot)$	Gaussian (normal) distribution
N	number of nodes
$\nabla_\mathcal{G}$	graph gradient
ν_0	frequency-domain correlation width
ω	(angular) frequency
$\mathrm{P}\{\cdot\}$	probability of an event
P	filter order of a Laplacian polymial filter
$\mathbf{p}_\mathfrak{x}$	power spectral density
$\mathbf{P}^\#$	pseudoinverse of $\mathbf{P}$
$\psi_{\mathsf{x}_m\mathsf{x}_n}$	correlation coefficient
$\mathtt{Q}$	quantisation procedure
Q	rate per vertex sample
Q'	rate per graph node

q_w	quantiser decision boundary
$\mathbb{R}$	real numbers
R	rate, bit budget
$R^{(I)}$	information rate
$\mathcal{S}$	sampling set
$\mathtt{S}$	sampling procedure
$\boldsymbol{\mathsf{S}}_{\mathcal{S}}$	GGT shift tensor
$\mathbf{S}_{\mathcal{S}}$	row-selection matrix
σ^2	variance
$\mathbb{T}_i\mathbf{x}$	generalised translation of $\mathbf{x}$ by i
T	time period for continuous time Gabor transform
$\mathcal{T}_2(\mathbf{x})$	normalised smoothness metric
$\left(\tilde{\mathbb{T}}_u x\right)(t)$	time shift of $x(t)$ by u
τ_0	vertex-domain correlation width
θ	threshold for absolute correlation coefficient
$\mathbf{U}$	matrix with eigenvectors as columns, GFT matrix, DFT matrix
$\mathbf{u}$	eigenvector
$\mathbf{U}_K$	first K columns of GFT matrix
$\tilde{\mathbf{u}}_i$	i^{th} row of $\mathbf{U}$
$\mathcal{V}$	set of nodes
$\mathbf{W}$	weight matrix
$\mathfrak{w}, \mathbf{w}$	white noise in frequency and vertex domain

W	number of reproducer values
w	weight function
$\mathbf{x}$	random vector
$\mathbf{x}$	source vector, source signal
$\mathcal{X}$	source alphabet
$\boldsymbol{\mathfrak{x}}$	GFT coefficients
$\mathfrak{x}\left(j\omega\right)$	spectrum of continuous time signal $x\left(t\right)$
$\mathbf{x}_K$	first K GFT coefficients
$\boldsymbol{\mathfrak{x}}_{\mathbf{g}}$	WGFT coefficient for all vertex and frequency shift
$\mathfrak{x}_g\left(\tau, \xi\right)$	WFT coefficient for time shift τ frequency shift ξ
$\hat{\mathbf{x}}$	reproducer vector
$\hat{\mathcal{X}}$	reproduction alphabet
$\mathbf{x}_{\mathcal{S}}$	sampled signal vector
Ξ	Lagrange multiplier for rate-distortion function
ζ	subdiagonal factor

Statement of authorship

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, 2. Juni 2024

Philipp Reingruber