A Probabilistic Graphical Model Based Data Compression Architecture for Gaussian Sources

by

Wai Lok Lai


Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2016

© 2016 Massachusetts Institute of Technology. All rights reserved.

Signature redacted

Author

Signature redacted

Certified by

Signature redacted

Accepted by

Signature redacted

Gregory W. Wornell
Sumitomo Professor of Engineering
Thesis Supervisor

Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee
DISCLAIMER NOTICE

a: The pagination in this thesis reflects how it was delivered to the Institute Archives and Special Collections.

b: The images contained in this document are of the best quality available.
A Probabilistic Graphical Model Based Data Compression Architecture for Gaussian Sources

by

Wai Lok Lai

Submitted to the Department of Electrical Engineering and Computer Science on September 6, 2016, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

Data is compressible because of inherent redundancies in the data, mathematically expressed as correlation structures. A data compression algorithm uses the knowledge of these structures to map the original data to a different encoding. The two aspects of data compression, source modeling, i.e. using knowledge about the source, and coding, i.e. assigning an output sequence of symbols to each output, are not inherently related, but most existing algorithms mix the two and treat the two as one.

This work builds on recent research on model-code separation compression architectures to extend this concept into the domain of lossy compression of continuous sources, in particular, Gaussian sources. To our knowledge, this is the first attempt with using sparse linear coding and discrete-continuous hybrid graphical model decoding for compressing continuous sources.

With the flexibility afforded by the modularity of the architecture, we show that the proposed system is free from many inadequacies of existing algorithms, at the same time achieving competitive compression rates. Moreover, the modularity allows for many architectural extensions, with capabilities unimaginable for existing algorithms, including refining of source model after compression, robustness to data corruption, seamless interface with source model parameter learning, and joint homomorphic encryption-compression.

This work, meant to be an exploration in a new direction in data compression, is at the intersection of Electrical Engineering and Computer Science, tying together the disciplines of information theory, digital communication, data compression, machine learning, and cryptography.

Thesis Supervisor: Gregory W. Wornell
Title: Sumitomo Professor of Engineering
Acknowledgments

I would like to express my gratitude for the many people without whose help, direct or indirect, I would not have been able to complete this work.

First, I would like to thank Prof. Gregory Wornell for introducing me to the world of probabilistic inference through the class sequence 6.437-6.438, from which I learned to apply and refine my understanding of probability to the problem of statistical inference. Prof. Wornell has been extremely patient when progress was slow, pointing me to useful resources and shedding light to problems in a different way. I feel fortunate to have the opportunity to be under his guidance, to have drawn from the breadth of his knowledge.

I would also like to thank Joshua Lee, who worked with me extensively on this project. The initial idea and the original MATLAB implementation of a low-density quantizer was due to him. Throughout this project, he has given me many crash courses in various topics in Electrical Engineering, which, coming from a Mathematical and Computer Science background, I lack, especially for a project fundamentally tied to concepts in Electrical Engineering.

This work has been inspired by Dr. Ying-Zong Huang, whose 2015 PhD dissertation details the idea and the implementation of a model-code separation architecture for compressing discrete sources. Ying-Zong communicated to Joshua and I valuable experiences he had with his system, which form the foundation of this work.

During the exploratory phase of this work, I had the chance to speak with Dr. Ulugbek Kamilov, who researched in a similar topic of message-passing de-quantization with respect to compressed sensing. This conversation impacted the design of the quantization structure presented in this work.

Next, I would like to express my appreciation for Prof. Katrina LaCurts and Shreya Saxena, who first introduced me to the concept of channel coding, linear codes, and the Lempel-Ziv algorithm. This foundational knowledge was very useful for joining the dots between channel coding and data compression.

I am thankful for the SIA community, especially for the support of Tricia and Giovanni, who helped me with various issues during my time at SIA.

With the mathematically heavy nature of this work, I would like to thank Martin Butzen, William Carroll, Jim Owens, and Joseph Maggio for endowing me with a mathematical and a probabilistic understanding, and without whose help coming to, let alone graduating from, MIT would have been a mere dream.

Finally, I reserve my utmost gratitude for my parents, for being my first teachers and impressing on me the thirst for knowledge, the love of mathematical reasoning, and the determination to pursue my deepest dreams.
# Contents

Abstract  
Acknowledgments  
Table of Contents  
  List of Figures  
  List of Tables  
  List of Algorithms  

1 Introduction  
  1.1 Motivation  
  1.2 Data Compression Overview  
    1.2.1 Structural Challenges  
    1.2.2 Examples of Joint Design  
    1.2.3 Rigidity of Joint Design  
  1.3 Previous Work  
  1.4 Contributions  
  1.5 Thesis Organization  
  1.6 Notation  

2 Background  
  2.1 Gaussian Distributions  
    2.1.1 Univariate Gaussian Random Variables  
    2.1.2 Jointly Gaussian Random Variables  
      2.1.2.1 Properties of Joint Gaussians  
  2.2 Coding Theory  
    2.2.1 Entropy and Coding  
    2.2.2 Linear Codes  
    2.2.3 LDPC Matrices for Compression  
    2.2.4 Rate-Distortion Theory  
  2.3 Probabilistic Graphical Models  
    2.3.1 Undirected Graphs  
      2.3.1.1 Pairwise Undirected Graphs  
    2.3.2 Factor Graphs  
    2.3.3 Gibbs Sampling  
    2.3.4 Marginalization and Belief Propagation  
      2.3.4.1 Maximum a Posteriori and Marginalization  
      2.3.4.2 Belief Propagation  


### 3 Compressor Architecture Overview

3.1 Architecture Components
- 3.1.1 Source Model
- 3.1.2 Quantizer
- 3.1.3 Translator
- 3.1.4 Binary Code

3.2 Encoder

3.3 Decoder
- 3.3.1 Source Subgraph
- 3.3.2 Quantizer Subgraph
- 3.3.3 Translator Subgraph
- 3.3.4 Code Subgraph
- 3.3.5 Decoding Algorithm

3.4 Doping Symbols

3.5 Rate-Distortion Optimization
- 3.5.1 Rate Selection

3.6 Complexity
- 3.6.1 Encode
  - 3.6.1.1 Time Complexity of Encode
  - 3.6.1.2 Space Complexity of Encode
- 3.6.2 Decode
  - 3.6.2.1 Time Complexity of Decode
  - 3.6.2.2 Space Complexity of Decode

3.7 Summary

### 4 Code and Translator Structure

4.1 Code Parameters
- 4.1.1 Row Weight and Column Weight
- 4.1.2 Four Cycles in LDPC

4.2 Translator Selection
- 4.2.1 Standard Binary Code
- 4.2.2 Gray Code

4.3 Doping
- 4.3.1 Random Doping
- 4.3.2 Sample Doping
- 4.3.3 Lattice Doping
- 4.3.4 Lattice Doping Extensions

4.4 Decoding Mechanics
- 4.4.1 Message Passing for Code Subgraph
- 4.4.2 Message Passing for Translator Subgraph

4.5 Summary
5 Quantizer Structure
5.1 Quantizer Selection ................................................. 61
  5.1.1 Uniform Scalar Quantizer ........................................ 61
  5.1.2 Low Density Hashing Quantizer ................................. 62
    5.1.2.1 Tuning of $m$ and the Sub-1-Bit Regime .................... 62
    5.1.2.2 Practical Concerns and Implications ....................... 63
5.2 Decoding Mechanics ............................................ 63
  5.2.1 The $\mathcal{F}^Q$ to $\mathcal{S}$ Messages ....................... 64
    5.2.1.1 Gaussian Approximations for $\mathcal{F}^Q$ to $\mathcal{S}$ Messages 65
  5.2.2 The $\mathcal{S}$ to $\mathcal{F}^Q$ Messages ........................ 67
  5.2.3 The $\mathcal{F}^Q$ to $\mathcal{U}$ Messages ......................... 67
5.3 Summary .................................................. 68

6 Source Modeling ............................................. 71
6.1 Graphical Model Representation of Source Models ............... 71
  6.1.1 Gaussian iid Source ...................................... 71
  6.1.2 Gauss-Markov Source ..................................... 72
6.2 Decoding Mechanics ........................................ 73
  6.2.1 Gaussian Potentials ...................................... 73
  6.2.2 Gaussian Message Passing ................................. 74
    6.2.2.1 Initialization ....................................... 75
  6.2.3 Marginalization and Convergence ........................ 75
6.3 Summary ................................................ 76

7 Compression Performance ..................................... 77
7.1 Experimental Setup ......................................... 77
7.2 Gaussian i.i.d. Sources .................................... 78
  7.2.1 Rate-Distortion Bound .................................. 78
  7.2.2 Known Lower Bounds for Classes of Algorithms .......... 78
    7.2.2.1 Lloyd-Max Algorithm .................................. 79
    7.2.2.2 Entropy-Coded Uniform Scalar Quantizer ................. 79
  7.2.3 Results ........................................... 80
7.3 Gauss-Markov Sources .................................... 80
  7.3.1 Rate-Distortion Bound .................................. 81
  7.3.2 Known Lower Bounds for Classes of Algorithms .......... 81
    7.3.2.1 Entropy-Coded Uniform Scalar Quantizer ................. 81
    7.3.2.2 Differential Pulse Code Modulation ..................... 82
  7.3.3 Results ........................................... 82
7.4 Summary ................................................ 82

8 Discussion and Analysis ..................................... 85
8.1 LDPC Code Optimization .................................... 85
  8.1.1 Four Cycles ........................................ 85
  8.1.2 Column Weights ....................................... 85
8.2 Doping Optimization ...................................... 86
List of Figures

1.1 Given a probability distribution and a quantization level \( \ell \) (\( \ell = 4 \) in this example), the Lloyd-Max algorithm provides a set of boundary values \( b_i \) which separate the real line \( \mathbb{R} \) into \( \ell \) regions \( R_i \), also called bins. If a data point falls in bin \( R_i \), it is stored as \( i \) at compression time, and reconstructed as \( a_i \) at decompression time. 

1.3 Most Domain II and Domain III systems process the data to extract compression gains before using an entropy coder to remove any remaining, memoryless structures.

1.4 Using JPEG on digitally created images is a model mismatch. The left image represent the original image, stored in the PNG format, and the right image is one compressed by JPEG. While natural photos decorrelate well in the DCT frequency domain, for digitally created images with sharp edges, this model assumption fails, leading to JPEG’s infamous wavy pattern.

1.5 The block diagram showing information flow in our proposed architecture for lossy compression on continuous sources. Note that by design, the Encode procedure is model-free, i.e. does not require information about the underlying source, while the Decode procedure applies its knowledge about the source model to reconstruct the data.

2.36 An undirected graph \( \mathcal{G} \) is defined by its variable nodes \( V \) and their associated node potentials \( \phi_i(s_i) \), and its edges \( E \) and their associated edge potentials \( \psi_{ij}(s_i, s_j) \).

2.42 A factor graph \( \mathcal{G} \) is defined by its variable nodes \( V \), its factor nodes \( \mathcal{F} \) and their associated factors \( f_\alpha(z_{N_a}) \), and its edges \( E \).

3.9 The Encode procedure takes in a source sequence \( s^n \) as an input, and outputs \( x^{kb} \) as the compressed bit stream. Encode contains three components: Quantize, Translate, and Code.

3.19 The Decode procedure takes in the compressed sequence \( x^{kb} \) as an input, and outputs \( \hat{s}^n \) as the reconstruction. Decode also needs the code, translator, and quantizer components, as well as the source model, to reconstruct the original source sequence.

3.20 The joint graph \( \mathcal{G} \) has four subgraphs: source \( \mathcal{C} \), quantizer \( Q \), translator \( T \), and code \( \mathcal{K} \), with factor nodes within each of the latter three components. There are four sets of variable nodes: source \( S \), quantized bins \( U \), translated bits \( Z \), and hashed bits \( X \). The relationships between the variables are embedded within the graph \( \mathcal{G} \).
3.34 In a rateless system, the compressed bits \( x \) is streamed to the decoder, until the decoder acknowledges that there is enough information for successful decoding.

4.2 Sample doping, where all bits corresponding to whole samples are doped. The shaded nodes represent the bits doped.

4.3 Lattice doping, where the same bit of each sample is doped. The shaded nodes represent the bits doped.

4.4 Multiple lattice doping, in this example the first and the third bits are doped. The shaded nodes represent the bits doped.

4.11 Message passing on the code subgraph: the orange arrow is a \( Z \) to \( T^M \) message, the blue arrow is a \( T^M \) to \( Z \) message, and the red arrow is a \( Z \) to \( T^F \) message. Note that the information about the compressed bit stream comes into the system only through the red messages.

4.23 Message passing on the translator subgraph: the red arrow is a \( F \) to \( Z \) message, and the blue arrow is a \( F \) to \( U \) message.

5.9 Message passing on the quantizer subgraph: the red arrow is a \( S \) to \( Q \) message, the blue arrow is a \( Q \) to \( S \) message, the brown arrow is a \( U \) to \( Q \) message, the orange arrow is a \( Q \) to \( U \) message.

6.3 Graphical Model for the Gaussian iid source, which only has node potentials \( \phi_i^C \).

6.14 Graphical Model for the Guass-Markov source, which has node potentials \( \phi_i^C \) and edge potentials \( \psi_{ij}^C \) in the structure of a Markov chain.

6.23 Belief propagation on the source subgraph \( C \). The source messages \( m_{S_i,S_j}^{(\tau)} \) (orange arrow) is computed as the integral over \( s_i \) of the product of the node potential (red arrow), neighboring edge potentials (blue arrows) except that from node \( S_j \), and incoming messages (green arrow).

7.12 Performance of our algorithm on Gaussian iid sources against the rate distortion bound, Lloyd-Max, and EPCM. The \( y \)-axis is the rate and the \( x \)-axis is the SQNR.

7.25 Performance of our algorithm on Gauss-Markov models against the rate distortion bound, ECUS, and DPCM. The \( y \)-axis is the rate and the \( x \)-axis is the SQNR.

9.5 Augmented source model to handle source models with known structures but unknown parameters.

9.7 The block diagram showing information flow in the encrypted compression architecture proposed by Johnson et al. Note that there are two channels: the public one for the compressed ciphertext, and the private one for the one-time pad. An eavesdropper who only has access to the compressed ciphertext will not be able to decompress and decrypt the data.
9.8 The decoding graphs for our system modified for quantized sources encrypted with a discrete one-time pad (left) and for sources encrypted with an additive Gaussian one-time pad (right). Note that the secret key $k$ enters the system through factor nodes (purple) corresponding to how it encrypts the data. 

9.11 A block diagram for a homomorphic encryption-compression scheme, achieved by incorporating Zhou's integer vector homomorphic encryption scheme into our architecture.

9.16 The decoding graphs for our system modified for Zhou's integer vector homomorphic encryption scheme. Note that the secret key $K$ enters the system through factor nodes $F^X$ (purple).
List of Tables

1.2 Classification of existing data compression algorithms into four categories, distinguished by their model specificity and reconstruction accuracy. In this thesis, we compare the performance of our architecture mainly to Domain III systems, which are shaded above. . . . . . . . . 19
List of Algorithms

2.43 Gibbs Sampling .................................................. 35
2.47 Belief Propagation on Factor Graphs .......................... 37
2.49 Belief Propagation on Pairwise Undirected Graphs ....... 37
3.17 Model-free Encoder ............................................... 42
3.29 Decoding Algorithm: Belief Propagation on Joint Graph 9 .. 47
Introduction

Data compression is a problem of fundamental importance at the intersection of electrical engineering and computer science. From the celebrated universal Lempel-Ziv algorithm which powers the ZIP compressor, to data specific compressors like JPEG, compression is essential to data transmission and storage.

Given the decades of research into data compression, it may be surprising to hear that existing algorithms have significant fundamental and theoretical inadequacies that inherently constraint its continued development. In this thesis, we will explore a new architecture for lossy compression that addresses these inadequacies while at the same time giving competitive performance compared to existing algorithms.

1.1 Motivation

To begin, let us consider an illustrative example. Suppose in a scientific experiment, 500 particles are fired one by one from a particle gun, and their terminal positions are thought to follow a Gaussian distribution with a known mean. The position of each particle is thought to be independent from one another, and standard deviation of the Gaussian distribution is thought to be 1.

Given that a Gaussian distribution is continuous that can take on any possible real value, it is impossible to consistently represent the positions of the particles to arbitrary precision with a finite number of bits. Therefore, some distortion is inevitably introduced between the actual position and the represented position.

One way to store the positions of the particles will be to store them as decimals, with precision up to a certain number of significant figures. However, with the number of bits we need to represent the truncated decimal numbers, we can do better by noting that in a Gaussian distribution, values closer to the mean are more likely to occur, and we can improve the expected distortion by recording values closer to the mean with a higher precision at the expense of the values far from the mean, rather than the uniform $10^{-x}$ precision implicitly imposed by a decimal representation or the $2^{-x}$ imposed by a floating point number.
The well known Lloyd-Max algorithm [16] gives the optimum expected distortion for a given fixed number of bits used to represent a single value, provided that the distribution is known [16]. The Lloyd-Max algorithm provides a set of boundary values on the real line $\mathbb{R}$, and representative values for each bin defined by the boundaries, where the representative values are used for reconstruction. Figure 1.1 illustrates the Lloyd-Max algorithm.

![Figure 1.1](image.png)

Given a probability distribution and a quantization level $\ell$ ($\ell = 4$ in this example), the Lloyd-Max algorithm provides a set of boundary values $b_i$ which separate the real line $\mathbb{R}$ into $\ell$ regions $R_i$, also called bins. If a data point falls in bin $R_i$, it is stored as $i$ at compression time, and reconstructed as $a_i$ at decompression time.

Given the optimum guarantee, it makes sense to use the Lloyd-Max algorithm to store the observed positions of our particles. However, if we were to find out later that the underlying Gaussian distribution of the positions is better modeled with a standard deviation of 2, the optimum guarantee breaks down. Moreover, if it turns out that the positions of the particles are not independent at all, that they can be modeled as a Markov chain with each position correlated to its immediate neighbors, we will be stuck with an extremely inefficient and imprecise way of representing the data, for we only have access to the represented values, not the original values. It is not clear how one would modify the existing structure to accommodate this new discovery, if possible at all. If luckily we still have access to the original position data, we will have to redesign a new way of representing and storing the values. If we do not, then not much can be done to improve the representation.

This example illustrates the core concept of rate-distortion trade-off, in which more bits are necessary for less distortion. In addition, it is an instance of the very common design paradigm of using specific assumptions about the data (the source model) to optimize for the compression rate. It also exemplifies how dependent the compression representation is on the source model, that even a small change in the parameters will lead to significant incompatibilities. In this respect, most existing practical compression architectures suffer from this very same problem faced by our example.

### 1.2 Data Compression Overview

To better discuss existing compression architectures, let us first categorize them into four categories.
Table 1.2: Classification of existing data compression algorithms into four categories, distinguished by their model specificity and reconstruction accuracy. In this thesis, we compare the performance of our architecture mainly to Domain III systems, which are shaded above.

In Table 1.2, the columns refer to whether the algorithm requires the form of the data to be specified at the time when the algorithm is designed.

- Model specific compression algorithms are designed around knowledge about the data type they are built to compress. For example, JPEG is designed with knowledge that the data being compressed is a 2-dimensional array of color values, and MPEG is designed with knowledge that the data is a set of frames of images.

- Model agnostic compression algorithms, on the other hand, do not use prior knowledge about the source; instead, it learns the source model from the data at compression time. For example, Lempel-Ziv does not make any assumptions about the format of the data: at compress time it dynamically builds a dictionary of repeating symbols to extract one-dimensional patterns from the data itself.

The rows refer to the accuracy of the reconstruction compared to the original source.

- Lossless compression expects a perfect reconstruction of the original data. For example, an unzipped ZIP folder contains the exact same bit sequence as the original files. It is usually applied to digital content, e.g., documents, texts, which cannot afford any loss of original content.

- Lossy compression, on the other hand, allows for distortions in the reconstruction. For example, JPEG files pixelate when reconstructed. It is usually applied to analog content, e.g., music, pictures, videos, that may be too costly and unnecessary, or even impossible, to digitally represent perfectly.

The study of information theory and digital communication, which began with Claude Shannon’s groundbreaking research in 1948 [20], directly gave rise to Domain I systems. While not used practically in and of themselves, Domain I systems form the basis of most practical compression systems, which lie in Domain II or Domain III with this categorization. In particular, most Domain II systems are extensions of Domain I systems with adaptivity, and most Domain III systems are extensions of Domain I systems with quantization.
As noted by Huang [14], the task of coding is thus very similar with most existing systems, for the paradigm of most systems is to pre-process the data to achieve compression with knowledge of the structure before using an entropy coder to clean up any remaining gains, illustrated by Figure 1.3. Inherent to this paradigm, however, is the fundamental problem of entanglement, as we shall now discuss.

### 1.2.1 Structural Challenges

As illustrated in Figure 1.3, the Encode algorithm for existing architectures can be broken down into a data specific Process component and an entropy coding Code component. Knowledge of the source model is required to design both Process and Code. One immediate concern is the lack of distinction of the role of either components: it is not clear what one does that the other does not, and it is hard to draw a line between the two, for both are involved in source modeling, ie. using knowledge of the source, and coding, ie. assigning a compressed output for a given data input.

With this definition of source modeling and coding, we can see that most existing systems can be labeled as joint model-code architectures, which consider source modeling and coding together as a whole when designing the algorithm. Such algorithms, by their nature, entangle model and code in both the Process and Code components. This entanglement of model and code is usually justified by the opportunities for clever optimizations in the compression rate. However, this inherently constraints the power of the compression algorithm in terms of readiness for change and in terms of generalization to other (perhaps related) data sources.

### 1.2.2 Examples of Joint Design

An example for a Domain III system is the JPEG compressor [21], which involves Discrete Cosine Transform (DCT, the real portion of the Discrete Fourier Transform), quantization, and entropy coding. The model assumption is that still images decorrelate well in the DCT frequency domain. The quantization step discards the high frequency data, since the human eye is insensitive to high frequency components. Finally, the symbols are then coded with a Huffman code. On the decoding side, the decoder reverses this procedure by first decoding the Huffman code, then piecing the data back together in the frequency domain, before applying the inverse DCT to reconstruct the image.
On the other hand, the popular ZIP compressor, a representative of a Domain II system, also follows this design pattern. The DEFLATE algorithm [6] which powers the ZIP compressor involves using the Lempel-Ziv compressor followed by Huffman coding. While ZIP is a universal compressor that does not require an explicit source model at compression time, the Lempel-Ziv component implicitly assumes a 1-dimensional pattern in the data source, and will only work well practically when this implicit model is satisfied. While it can still achieve some compression for non-1-dimensional data like photos, its performance is much better for sources like plain text, which satisfies its implicit model. In the DEFLATE algorithm, Lempel-Ziv learns and adapts to the data source at compression time by building a dynamic dictionary to represent repeating symbols, and the residual entropy is squeezed out by Huffman coding.

The final step of both JPEG and ZIP is Huffman coding, a Domain I system, which is being embedded within the larger compressor to squeeze out any remaining compression, with the majority of the compression rate comes from the pre-processing components.

1.2.3 Rigidity of Joint Design

We now illustrate the design rigidity of the joint model-code paradigm with JPEG as an example. While natural photographs decorrelate well in the DCT frequency domain, it was later discovered that they decorrelate better in the Wavelet domain, a decision that has been incorporated into the newer JPEG2000 standard. This improvement of knowledge of the source model, however, cannot be incorporated into the existing JPEG algorithm as the DCT and the Wavelet Transform are not compatible. An refinement of the source model has thus led to an entirely new algorithm, JPEG2000, but it sees a low adoption rate since it is fundamentally incompatible with the original JPEG standard.

Furthermore, JPEG was originally intended for natural photographs, as suggested by its full name (Joint Photographic Experts Group). However, over time, as new types of images became more common and available, its function has been overloaded with users compressing artificial graphics with JPEG (Figure 1.4). Artificial graphics have a very different underlying model from natural photographs, but JPEG, even JPEG2000, has no way to handle this source model assumption violation, which tends to occur when new types of data becomes available and there is not an algorithm outstanding to compress it. The result of this model mismatch is the infamous JPEG pixelation, where the wavy patterns of the Discrete Cosine basis become noticeable. To solve this problem, an entirely new algorithm mutually incompatible with JPEG will need to be designed, and in general, with the joint model-code paradigm, a new algorithm will need to be designed whenever a new class of data emerges.

The rigidity of joint model-code design motivates us to consider a compression architecture with model-code separation, where there is a clear separation between using knowledge about the data source (ie. source modeling) and assigning output symbols (ie. coding).
Hello

Figure 1.4: Using JPEG on digitally created images is a model mismatch. The left image represents the original image, stored in the PNG format, and the right represent one compressed by JPEG. While natural photos decorrelate well in the DCT frequency domain, for digitally created images with sharp edges, this model assumption fails, leading to JPEG’s infamous wavy pattern.

1.3 Previous Work

With the object of this thesis being designing and exploring a data compression algorithm, most relevant to our architecture would be work from the information theory and digital communications community. Shannon’s landmark 1948 paper [20] laid the foundation for both digital communication and information theory. In particular, the branch of information theory called rate-distortion theory investigates lossy compression and its fundamental limits, a well-developed theory that we describe in Section 2.2.4 which we shall reference frequently in this thesis.

Following Shannon’s developments, Gallager developed Low-Density Parity-Check (LDPC) codes in his doctoral dissertation in 1960 [7], a type of linear code that will form the backbone of our compression algorithm. LDPC codes are discussed in more details in Section 2.2.3. LDPC code was soon forgotten as its computational complexity for exact decoding was too high for computer systems in the 1960s, and its development remained dormant until 1996.

During this time, Pearl introduced Belief Propagation (BP) in 1982 [17]. Also known as sum-product message passing, BP is an inferential algorithm for marginalization on probabilistic graphical models, as we shall introduce in Section 2.3.4.2. While originally developed for tree graphs, over the years, the machine learning community has refined the tool as an approximation algorithm for general graphs, and as we shall see, approximate BP is a tool key to decoding LDPC codes efficiently.

More recently, in 2003 Caire et al. researched in using LDPC codes for lossless compression of memoryless discrete sources [4], using belief propagation and doping (discussed in Section 3.4) as the decoding mechanism. The researchers extended the algorithm to a universal lossless compressor by pre-processing the data using the Burrows-Wheeler transform [3], which helps the algorithm adapt to and decorrelate the data source at compression time. This pre-processing does not achieve any compression, but allows LDPC codes to compress the processed data more efficiently.

This universal lossless compressor, by its nature, does not allow for incorporation of source model information. More relevant to our work would be Garcia-Frias and Zhong’s 2003 paper [8], which experimented with compressing binary Markov chains with LDPC codes, the first instance of some conception of a model-code separation architecture. Then, Schonberg et al.’s 2006 paper [18] discussed the idea of a source model (in the sense of knowledge about the source, as we defined in Section 1.2.1) and experimented with a 2D Markov model on bi-level images encrypted with a one-time pad.

LDPC codes for lossy compression for binary sources was discussed in 2007 by Braunstein et al. [2], who used LDPC codes over \( \mathbb{Z}_p \) (instead of the established
binary LDPC codes) to achieve results close to the rate-distortion bound for binary Bern \( \left( \frac{1}{2} \right) \) sources.

The recent work of Huang and Wornell in 2014 [13] brings together these concepts by proposing a general compression algorithm with model-code separation architecture for discrete sources. The core component is a lossless binary compressor powered by LDPC encoding and BP decoding, with a flexible architecture to allow for lossless non-binary discrete sources (by incorporating a translator component) and for lossy binary compression (by incorporating a quantizer component).

1.4 Contributions

In this thesis, we expand on the work of Huang and Wornell [13, 14], who proposed a model-code separation architecture for data compression that addressed the concerns raised in Section 1.2.1 for discrete systems. We shall apply the fundamental concept of model-code separation and belief-propagation decoding proposed by Huang, while exploring the results of extending the concept into the continuous domain. In particular, we shall focus our attention to compress Gaussian sources, which are known to be the most difficult to compress due to the fact that the Gaussian distribution is the maximum entropy distribution for any given mean and variance. Figure 1.5 illustrates our proposed architecture. To our knowledge, a lossy compressor for continuous sources based on sparse codes has not been attempted.

![Diagram](image)

Figure 1.5: The block diagram showing information flow in our proposed architecture for lossy compression on continuous sources. Note that by design, the Encode procedure is model-free, i.e. does not require information about the underlying source, while the Decode procedure applies its knowledge about the source model to reconstruct the data.

In the following chapters, we present the details of this model-code separation architecture on Gaussian sources. We note the modularity of the model component, which operates in the continuous domain, and the code component, which operates in the discrete domain, whereby the two components are connected by a quantizer component, which bridges the continuous and the discrete domains into one structure. To our knowledge, this is the first attempt in using continuous-discrete hybrid Belief Propagation in the context of lossy compression. Next, We discuss the benefits afforded by its architectural flexibility, emphasizing the component-wise optimizations that can be done independently of other components, such as refining the source model or searching for better codes. Finally, we compare its performance with the
theoretical lower bound (ie. the Rate-Distortion Bound) and the bounds of other known classes of algorithms, showing that our architecture provides competitive performance against existing algorithms, without the rigidity and inadequacies discussed in Section 1.2.1.

1.5 Thesis Organization

The thesis is organized as follows:

In Chapter 2 (Background), we first introduce concepts and tools, including Gaussian distributions, linear coding, and probabilistic graphical models, all of which will be essential to understanding this work.

Next, in Chapter 3 (Compressor Architecture Overview), we give a self-contained overview of our proposed compression architecture, pointing out the different modular components of the architecture, including coding, quantization, and source modeling.

Then, in Chapters 4 (Code and Translator Structure), 5 (Quantizer Structure), and 6 (Source Modeling), we discuss the details of the code, quantizer, and source model components of the architecture respectively.

After presenting the algorithm, in Chapter 7 (Compression Performance) we discuss our performance in light of the theoretical lower bounds as well as the bounds of other algorithms.

In Chapter 8 (Discussion and Analysis) we analyze our design choices and how those decisions affect performance.

In Chapter 9 (Realistic Applications and Extensions), we propose extensions to our system that will have practical impacts.

Finally, in Chapter 10 (Conclusion), we summarize our work and point out potential ideas that call for further research.

1.6 Notation

While there are many commonly used sets of notations within the different communities of information theory, digital communication, and machine learning, for the sake of unity, clarity, and conciseness, we have chosen the following set of notation for this thesis:

We use lower case $x$ to denote a scalar or vector variable. If necessary, we clarify the dimension of a vector with superscripts $x^n$ or with $x \in \mathbb{R}^n$.

We use upper case $A$ to denote a matrix. We clarify its dimension with $A \in \mathbb{R}^{m \times n}$.

We use upright $x$ to denote a constant, either a scalar or a vector.

For vectors and matrices, we use subscripts $x_i$ to denote the $i^{th}$ entry of the vector $x$ or $A_{ij}$ to denote the $(i, j)^{th}$ entry of the matrix $A$.

For iterative algorithms, we use super script $m_i^{(T)}$ to denote the value of the variable $m_i$ at iteration $T$. 
We use italicized sans serif $x$ to denote a random variable, with dimensionality clarified as necessary.

We use boldface $\mathbf{N}(\cdot)$ to denote a distribution.

We use script $\mathcal{H}$ to denote a set.

We use $\mathbb{R}$ to denote the real numbers, $\mathbb{R}_+$ to denote the positive real numbers, and $\mathbb{Z}_n$ to denote the integers mod $n$.

We use italics $f(\cdot)$ to denote a function. In particular, we use $\mathcal{N}(x; \mu, \sigma^2)$ to denote the probability density function (pdf) of a Gaussian with mean $\mu$ and variance $\sigma^2$, which is a function in $x$.

We use double strike $\mathbb{H}(\cdot)$ to denote a statistical function of random variables. With $\mathbb{E}(\cdot)$ denoting the expected value of random variable, subscripts $\mathbb{E}_k(\cdot)$ clarify the variable with respect to which the expectation is taken.

We use calligraphic $\mathcal{G} = (\mathcal{V}; \mathcal{E})$ to denote a graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$, and $\mathcal{G} = (\mathcal{V}_1, \ldots, \mathcal{V}_n; \mathcal{E})$ to denote an $n$-partite graph. We use $\mathcal{N}_i^{\mathcal{V}_k}$ to denote the set of vertices in $\mathcal{V}_k$ that are neighbors of vertex $i$. The super script will be dropped when unambiguous.

We use sans serif $\text{Decode}$ to denote a procedure.

We use monospace $\text{Lloyd-Max}$ to denote an established system or scheme.
Background

To design an architecture capable of compressing a sequence drawn from a continuous model into a finite number of bits, we unavoidably face a three-fold challenge: source modeling, quantization, and encoding. In this chapter, we give an overview of the tools and methodologies required for such a system before we present our compression architecture.

Section 2.1 reviews the definition and the properties of a multivariate Gaussian distribution, the class of sources that is the focus of this thesis. Section 2.2 presents classical coding theory as a method to reframe the data compression problem, as well as presenting an overview of the quantization landscape, which includes methodologies and baselines to which we compare our architecture. Section 2.3 introduces probabilistic graphical models, the main tool that we use to construct the architecture.

2.1 Gaussian Distributions

2.1.1 Univariate Gaussian Random Variables

The following is the familiar definition of a single variable Gaussian (normal) distribution:

**Definition 2.1. (Univariate Gaussian).** A random variable $s \in \mathbb{R}$ is Gaussian if its probability density function (pdf) $p_s(\cdot)$ can be written as

$$p_s(s) \propto \exp \left\{ -\frac{1}{2\sigma^2} (s - \mu)^2 \right\}$$

(2.2)

for some $\mu \in \mathbb{R}$, known as the mean, and $\sigma^2 \in \mathbb{R}_+$, known as the variance.

Following standard notation, we write $s \sim \mathcal{N}(\mu, \sigma^2)$ to denote that $s$ is a Gaussian random variable with the associated parameters. In particular, we call $\mathcal{N}(0, 1)$ the standard normal distribution. We write $p_s(s) = \mathcal{N}(s; \mu, \sigma^2)$ to denote the probability density function itself.
2.1.2 Jointly Gaussian Random Variables

**Definition 2.3. (Multivariate Gaussian).** A random vector \( s^n \) is jointly Gaussian if its joint distribution \( p_{s^n}(\cdot) \) can be written as

\[
p_{s^n}(s^n) \propto \exp \left\{ -\frac{1}{2} (s - \mu)^T \Sigma^{-1} (s - \mu) \right\}
\]

(2.4)

for some \( \mu \in \mathbb{R}^n \), known as the mean vector, and some positive definite matrix \( \Sigma \in \mathbb{R}^{n \times n} \), known as the covariance matrix.

Through a change of variables, we can obtain an alternate parametrization, known as the information form, of the jointly Gaussian distribution. Letting \( \Lambda = \Sigma^{-1} \) and \( \eta = \Lambda \mu \), a simple substitution gives

\[
p_{s}(s) \propto \exp \left\{ -\frac{1}{2} s^T \Lambda s + \eta^T s \right\}
\]

(2.5)

where \( \eta \) is known as the potential vector and \( \Lambda \) is known as the precision matrix.

Following standard notation, we write \( s^n \sim \mathcal{N}(\mu, \Sigma) \) or \( s^n \sim \mathcal{N}^{-1}(\eta, \Lambda) \) to denote that \( s \) is jointly Gaussian with the corresponding parameters. In particular, we call \( \mathcal{N}(0, I) \) the iid standard normal distribution. We write \( p_{s^n}(x^n) = \mathcal{N}(s^n; \mu, \Sigma) \) to denote the joint probability density function.

**2.1.2.1 Properties of Joint Gaussians**

We now present four important facts that are central to our development of Gaussian random variables.

**Fact 2.6.** A random vector \( s \) is jointly Gaussian if and only if it can be expressed as a linear combination of independent and identically distributed (iid) univariate standard normal variables, ie. there exists some matrix \( A \) and some offset vector \( b \) such that \( s = Aw + b \), where \( w \sim \mathcal{N}(0, I) \) is the iid standard normal distribution.

**Fact 2.7.** A random vector \( s \) is jointly Gaussian if and only if every linear combination of the elements of \( s \) is a univariate Gaussian variable, ie. for any constant vector \( a \), there exists scalars \( \mu \in \mathbb{R} \) and \( \sigma^2 \in \mathbb{R}_+ \) such that \( a^T s \sim \mathcal{N}(\mu, \sigma^2) \).

**Fact 2.8. (Marginalization of Gaussians).** Any marginalization of a jointly Gaussian distribution is also a jointly Gaussian distribution.

**Fact 2.9. (Conditionals of Gaussians).** The conditional distribution of any subset of a set of jointly Gaussian random variable, conditioned on any other subset of the set, is also jointly Gaussian.

As a more concrete formula, which we shall use in later parts of this thesis, we present two additional facts:
Fact 2.10. (Marginalization of Joint Gaussians in Information Form). If \( s \) is a set of jointly Gaussian variables with mutually exclusive subsets \( s_1 \) and \( s_2 \) such that
\[
s = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \sim N^{-1} \left( \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix}, \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22} \end{bmatrix} \right)
\] (2.11)
then
\[
s_1 \sim N^{-1}(\eta', \Lambda')
\] (2.12)
where
\[
\eta' = \eta_1 - \Lambda_{12} \Lambda_{22}^{-1} \eta_2
\] (2.13)
\[
\Lambda' = \Lambda_{11} - \Lambda_{12} \Lambda_{22}^{-1} \Lambda_{21}
\] (2.14)

Fact 2.15. (Pointwise Products of Gaussian pdf’s). Consider a set of \( n \) Gaussian pdf’s \( N^{-1}(s; \eta_i, \Lambda_i) \) for \( i \in \{1..n\} \). Their pointwise product is
\[
\prod_{i=1}^{n} N^{-1}(s; \eta_i, \Lambda_i) \propto N^{-1}(s; \eta', \Lambda')
\] (2.16)
where
\[
\eta' = \sum_{i=1}^{n} \eta_i
\] (2.17)
\[
\Lambda' = \sum_{i=1}^{n} \Lambda_i
\] (2.18)

2.2 Coding Theory

The study of data processing and compression began with Claude Shannon, the pioneer of the field of communication and information theory. In his original work, he investigated two types of compression systems: lossless compression (also known as source coding theory), described in Section 2.2, and lossy compression (also known as rate-distortion theory), described in Section 2.2.4.

In both systems, a probabilistic description of the source \( s \), known as a source model, needs to be known. The source model represents prior knowledge of the data being compressed, assuming that each instance \( s^n \) of the class of data \( s^n \) is drawn independently from the known joint probability density function \( p_{s^n}(s^n) \).

2.2.1 Entropy and Coding

Entropy \( H(s) \) is the measure of the average information transmitted through a message \( s \). Entropy, thus, serves as a natural lower bound to the compression rate if a
message were to be recovered without loss. We first present a fundamental theorem of coding theory:

**Theorem 2.19. (Shannon’s Source Coding Theorem).** For an iid source $s^n$, as $n \to \infty$,

(a) There exists a uniquely decodable code that can compress instances $s^n$ into less than $n \cdot H(s)$ bits without any loss of information, and

(b) There does not exist a uniquely decodable code that can compress instances $s^n$ into less than $n \cdot H(s)$ bits without any loss of information,

where the value

$$H(s) := \lim_{n \to \infty} \frac{1}{n} H(s^n) = \lim_{n \to \infty} \frac{1}{n} \mathbb{E}_{s^n}(-\log_2(p_{s^n}(s^n)))$$

(2.20)

is known as the entropy rate of $s$.

### 2.2.2 Linear Codes

Shannon’s development of information theory makes it clear that he treated communication and compression as closely related problems, perhaps even duals of the same problem. After Shannon’s original publication, many others have discovered that random sparse linear codes are amongst the best codes for digital communication, while the compression side of the problem has not seen much development along these lines.

**Definition 2.21. (Linear Code).** A linear code is a code that encodes a length $(n-k)$ source sequence into a length $n$ codeword by using $k$ extra parity check bits to correct for potential corruption of codewords when sent over a channel. In particular, for a source alphabet $\mathcal{S}^{n-k} = \mathbb{Z}_q^{n-k}$, a linear code defined by a linear transform

$$L : \mathcal{S}^{n-k} \to \mathcal{S}^n$$

(2.22)

Such a linear transform can be characterized by a *generator matrix* $G \in \mathbb{Z}_q^{(n-k) \times n}$ with

$$L(s) = G^T s$$

(2.23)

or equivalently, by the *parity check matrix* $H \in \mathbb{Z}_q^{k \times n}$ with

$$HG^T = 0$$

(2.24)

Linear codes are widely used as *error correcting codes* because of the following well known fact on its optimality:

**Fact 2.25.** Linear codes over finite field alphabets $\mathbb{Z}_q$ achieve capacity over symmetric discrete memoryless channels (DMC).
Linear encoding, as described above, amounts to multiplication by $G^T$, which has time complexity $O(n(n-k))$. Decoding, on the other hand, requires some more work. Letting $\mathcal{L} = \{v : v = G^T u\}$ with $\{u\}$ being the set of all source sequences, we note that the set of received codewords, with channel noise $w$, would be

$$\mathcal{L} + w = \{y : y = G^T u + w\} = \{y : Hy = Hw\}$$

Hence, decoding amounts to determining the maximum likelihood (ML) $\hat{w}$ that satisfies $Hy = H\hat{w}$, which has time complexity $O(n|\mathcal{F}|^k)$, too large to be efficiently decoded.

Low-density parity-check (LDPC) codes are codes with parity check matrices that have insignificant row and column weights, negligible compared to the growth of $n$. With iterative local decoding, for which good performance has been proven, decoding has time complexity $O(k|\mathcal{F}|^\rho \tau^*)$, where $\rho$ is the maximum row weight and $\tau^*$ is the number of iterations till convergence, with $\tau^* = O(1)$ for fixed rates. With $\rho$ set to a constant, LDPC codes have encoding complexity of $O(n)$ and decoding complexity $O(n)$.

### 2.2.3 LDPC Matrices for Compression

As previously noted, there is a strong duality between communication and compression. Identifying the channel noise in communication with the source distribution in compression, we get the dual of Fact 2.25:

Fact 2.27. Linear codes over finite field alphabets $\mathbb{Z}_q$ achieve entropy in lossless compression of discrete memoryless sources (DMS).

In this dual problem, compression amounts to a linear projection by the parity-check matrix $H \in \mathbb{Z}_q^{k \times n}$, with compressed data $x \in \mathcal{F}^k$ computed as

$$x = H z$$

for source sequence $z \in \mathcal{F}^n$. Compression thus has time complexity $O(nk)$.

As in Section 2.2.2, computing the maximum likelihood (ML) $\hat{x}$ for a general parity check matrix $H$ will have time complexity $O(n|\mathcal{F}|^k)$. However, if we resort to LDPC matrices with iterative local decoding, we can bring the decoding complexity down to $O(k|\mathcal{F}|^\rho \tau^*)$, while encoding will be $O(\rho k)$. It is important to note that the time complexity of both operations are thus both $O(n)$, linear in the length of the source sequence.

While Fact 2.27 only applies to discrete memoryless sources, the same result for general sources including sources with memory and non-stationary sources. Although it does not immediately follow that LDPC codes and ML decoding achieve the theoretical bound, we take this result with an optimistic note that recent research on much larger classes of sources with LDPC encoding yields mostly positive results.
2.2.4 Rate-Distortion Theory

When the size of the alphabet is too large, e.g. infinite, or that too many bits are required to represent each possible value distinctly, we may be willing to sacrifice some accuracy in return for a much smaller compressed output. Rate-Distortion Theory, the study of this trade-off between rate and distortion, is central to our development of a lossy compression architecture.

Lossy compression, unlike lossless compression, is a many-to-one compression method, which means its reconstruction of the source will be imperfect. To judge the goodness of a reconstruction \( \hat{s} \) of the original sequence \( s \), therefore, a measure of distortion \( \Delta(\hat{s}, s) \) is needed.

For discrete sources of a finite alphabet, a natural distortion measure is the Hamming Distortion:

\[
\Delta_H(s^n, s^n) := \frac{1}{n} \sum_{i=1}^{n} 1_{\hat{s}_i \neq s_i}
\]  

and for continuous sources, the common choice is the Mean Squared Error:

\[
\Delta_{\text{MSE}}(\hat{s}^n, s^n) := \frac{1}{n} \sum_{i=1}^{n} (\hat{s}_i - s_i)^2
\]

The problem of lossy data compression can thus be formulated as the following: For a source \( s \) with alphabet \( \mathcal{S} \), and given a distortion value \( \delta \), find the lowest achievable compression rate \( r \) such that the distortion between \( s \in \mathcal{S}^n \) and \( \hat{s} \in \mathcal{S}^n \) is at most \( \delta \), where \( \hat{s} \) is reconstructed from a bit stream \( x \in \mathbb{Z}_2^n \).

The rate-distortion function \( R(\delta, s) \) provides a lower bound on the compression rate \( r \) as a function of \( \delta \).

**Definition 2.31. (Rate-Distortion Function).** The rate-distortion function, also known as the rate-distortion curve, is defined to be

\[
R(\delta; s) := \lim_{n \to \infty} \frac{1}{n} R(\delta; s^n) = \lim_{n \to \infty} \frac{1}{n} \left( \inf_{\mathcal{G}} I(\hat{s}^n; s^n) \right)
\]

where \( \mathcal{G} := \{ p_{\hat{s}^n} : \mathbb{E}_{s^n}(\Delta(\hat{s}^n, s^n)) < \delta \} \) represents the set of reconstruction methods that has expected distortion smaller than \( \delta \), with \( I(\cdot; \cdot) \) denoting the mutual information.

Therefore, the rate-distortion function serves as the natural bound with which we compare lossy compression algorithms. In particular, we will focus on using the Mean Squared Error (MSE) as our distortion measure. Given the magnitude of the MSE error, it is customary to represent the error in decibels, i.e. on the logarithmic scale, a value known as the *signal to quantization noise ratio* (SQNR). The conversion between \( \delta_{\text{MSE}} \) distortion and \( \text{SQNR}_{dB} \) is defined to be

\[
\text{SQNR}_{dB} = -10 \cdot \log_{10}(\delta_{\text{MSE}})
\]
2.3 Probabilistic Graphical Models

Graphs provide a natural representation of the relationships between random variables. Recent developments within the machine learning community have led to standardization of many tools for statistical inference, in particular, probabilistic graphical models for the task of inference. Such models provide a compact representation of large sets of random variables with sparse correlation. While graphical models are primarily used for parameter estimation within the context of machine learning, this tool, with an appropriate set-up, is also suited to solve the problem of data compression.

2.3.1 Undirected Graphs

Definition 2.34. (Undirected Graphical Models). An undirected graphical model $G = (V; E)$ of a set of random variables $s$, also known as a Markov random field on $s$, consists of a set of vertices $V$, with each vertex $V_i$ representing a random variable $s_i$, and a set of edges $E$ connecting the vertices. For any mutually exclusive subsets $A$, $B$, $C \subset V$, the edges $E$ satisfy the following conditional independence property:

$$s_A \perp s_B \mid s_C$$

(2.35)

holds (ie. $s_A$ is conditionally independent of $s_B$ given $s_C$) whenever there does not exist a path in $E$ from any vertex in $A$ to any vertex in $B$ that does not pass through a vertex in $C$.

While it is not clear at first glance whether a graph satisfying this property can be constructed, the following asserts that such a graph exists for any given joint distribution:

Fact 2.37. (Universality of Undirected Graphs). Any strictly positive distribution $p_{s^n}(s^n)$ over an alphabet $\mathcal{S}^n$ can be represented as an undirected graph $G = (V; E)$, with $V$ representing the set of variables $s$ and the set of edges $E$ such that $p_{s^n}(s^n)$ can be factored over the maximal cliques $C$ of $G$ as

$$p_{s^n}(s^n) \propto \prod_{C \in \text{MaxClq}(G)} \psi_C(s_C)$$

(2.38)
where $\psi_{c}(\cdot)$ are non-negative functions known as potential functions. In addition, by the Hammersley-Clifford theorem [11], this factorization satisfies the conditional independence property in 2.35 when applied on the graph $\mathcal{G}$.

### 2.3.1.1 Pairwise Undirected Graphs

A subset of undirected graphical models, known as pairwise models, is a less powerful class of models, representing distributions where cliques are of size at most 2, i.e., it suffices to take the product in Equation 2.38 only over the edges. For this class of models, while the factorization in Equation 2.38 is complete, in practice it is more convenient to consider potentials on edges and assign unary potential functions to variable nodes, resulting in the alternate factorization of

$$p_{\mathbf{s}}(\mathbf{s}^{n}) \propto \prod_{i \in \mathcal{V}} \phi_{i}(s_{i}) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(s_{i}, s_{j})$$

for pairwise undirected graphs.

Due to the factorization structure of joint Gaussians, however, it can be proven that all Gaussian graphical models can be expressed as a pairwise undirected graph, meaning that we do not lose representation power in using pairwise undirected graphs for any arbitrary Gaussian distribution. Therefore, for the rest of this thesis, we shall assume the use of pairwise models on Gaussian distributions.

### 2.3.2 Factor Graphs

Undirected graphs can be consolidated to provide a more effective graphical representation than factorization over maximal cliques, which are hard to handle algorithmically and hard to decouple conceptually. Factor graphs provide an alternative representation of the same probability distribution on the same set of random variables.

**Definition 2.40. (Factor Graphs).** A factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{F}; \mathcal{E})$ of a set of random variables $\mathcal{Z}^{n}$ is a bipartite graph with edges $\mathcal{E}$ joining variable nodes $\mathcal{V}$ and factor nodes $\mathcal{F}$. A factor graph represents a factorization of

$$p_{\mathbf{z}}(\mathbf{z}^{n}) \propto \prod_{a \in \mathcal{F}} f_{a}(z_{\mathcal{N}_{a}^{\mathcal{V}}})$$

where $f_{a}(\cdot)$ are non-negative functions known as factors, and $\mathcal{N}_{a}^{\mathcal{V}}$ represents the neighbors of factor node $a$ in the variable nodes $\mathcal{V}$.

The universality of factor graphs in representing arbitrary joint distributions can be seen by applying Fact 2.37. Given an undirected graph, we can construct a factor representing the same distribution by replacing maximal cliques with factor nodes over the same variable nodes.
A graphical model represents the independence structure of a set of random variables. A particularly important task would be to sample from a distribution given its graphical model representation. The undirected nature of Markov random fields lends itself to a well-known sampling algorithm known as Gibbs sampling, as we describe now.

We assume that we have access to a single-element sampler $\text{Samp}(p_s)$ that can draw a single-element sample $s_i \in S$ according to the distribution $p_s(\cdot)$ in $O(|S|)$ time. We first initialize $s^{(0)}$ to some (potentially deterministic) value, and initialize $\sigma$ to be a permutation of $\{1 \ldots n\}$ as the order we update corresponding index of the sample. We then perform the following updates:

**Algorithm 2.43 Gibbs Sampling**

1: procedure GibbsSampler($p_{s^n}$) 
2: $\sigma \leftarrow \text{Perm}(n)$ \hspace{1cm} $\triangleright$ a permutation of $\{1 \ldots n\} $
3: $s^{(0)} \leftarrow 0$ \hspace{1cm} $\triangleright$ initialization
4: for $\tau \leftarrow 1 \ldots \tau^*$:
5: for $i \leftarrow \sigma(1) \ldots \sigma(n)$ : \hspace{1cm} $\triangleright$ sample every element of the vector
6: $s^{(\tau)}_i \leftarrow \text{Samp} \left( p_{s^n}(\cdot, s^{(\tau-1)}_i) \right)$
7: return $s^{(\tau^*)}$

For an undirected graphical model, we can express the expression on the right hand side of line 2.43.6 explicitly as

\[
s^{(\tau)}_i \leftarrow \text{Samp} \left( p_{s^n}(\cdot, s^{(\tau-1)}_i) \right) := \text{Samp} \left( \prod_{c \in \text{MaxClique}(S)} \psi_c(\cdot, s^{(\tau-1)}_{c \setminus i}) \right) \quad (2.44)
\]

in terms of the parameters of the undirected graph $S$.

After enough iterations, i.e. for a large enough $\tau^*$, we observe asymptotically independent samples, under moderate conditions on $p_{s^n}$. The Gibbs sampler has a time complexity of $O(n|S|\tau^*)$. 

37
2.3.4 Marginalization and Belief Propagation

2.3.4.1 Maximum a Posteriori and Marginalization

When we are faced with the task of inference, a frequent approach would be to represent underlying variables \( z \) and observed variables \( x \) as a graphical model, given its ease of representing conditional relationships between random variables. Thus, the relations among the underlying variables \( z \) represent a prior belief, which by having access to the values of the observed variables \( x \), we seek to compute a posterior belief on the underlying variables.

In mathematical terms, the task of inference is to find the maximum a posteriori (MAP) distribution, i.e., compute

\[
\hat{z} := \arg \max_z p_{z|x}(z \mid x) = \arg \max_z p_{z,x}(z, x)
\]

where the equality is due to \( p_{z,x}(z, x) = p_{z|x}(z \mid x) \cdot p_x(x) \). Hence, Equation 2.45 reduces the posterior computation to a marginalization of \( z \), i.e., computing the unconditional distribution \( p_{z,x} \) given the observation \( x \), which as we shall see is an operation central to our decompression system.

2.3.4.2 Belief Propagation

Given the importance of marginalization, we present the celebrated sum-product algorithm, an iterative algorithm that can efficiently compute the marginals on a graphical model representation of random variables. The sum-product algorithm, also known as belief propagation or message passing, works well for both undirected graphs and factor graphs.

For factor graphs, we assume a factorization structure as introduced in 2.41 of

\[
p_{z^n}(z^n) \propto \prod_{a \in F} f_a(z_{N^a})
\]

We let \( m_{ia}^{(t)}(\cdot) \) denote the message from variable node \( i \in V \) to factor node \( a \in F \) and \( m_{at}^{(\tau)}(\cdot) \) denote the message from \( a \) to \( i \) at iteration \( \tau \), noting that each message is a single variable function. Then, the sum product algorithm is defined as follows:

The expression in 2.47.7 is called the variable to factor update, the expression in 2.47.8 is called the factor to variable update, and the expression in 2.47.10 is called the total belief.

While a similar algorithm exists for general undirected graphs, we present the version that focuses on pairwise undirected graphs, as described in Section 2.3.1.1. The factor graph representation of a pairwise undirected graph will have factor nodes that are connected to at most 2 variables. Thus, the product in 2.47.8 will only involve one term. Thus, assuming the factorization in Equation 2.39 of

\[
p_{z^n}(z^n) \propto \prod_{i \in V} \phi_i(z_i) \prod_{(ij) \in E} \psi_{ij}(z_i, z_j)
\]
Algorithm 2.47 Belief Propagation on Factor Graphs

1: procedure FactorBP(\mathcal{V}, \mathcal{F}, \mathcal{E})
2: \hspace{1em} for (i, a) \in \mathcal{E} : \hspace{2em} \triangleright \text{initialization}
3: \hspace{2em} m_{ia}^{(0)}(z_i) \leftarrow 1
4: \hspace{2em} m_{ai}^{(0)}(z_i) \leftarrow 1
5: \hspace{1em} for \tau \leftarrow 1 .. \tau^*:
6: \hspace{2em} for (i, a) \in \mathcal{E} :
7: \hspace{3em} m_{ia}^{(\tau)}(z_i) \leftarrow \prod_{b \in \mathcal{F}_i \setminus \{a\}} m_{bi}^{(\tau-1)}(z_i) \hspace{1em} \triangleright \text{variable to factor messages}
8: \hspace{3em} m_{ai}^{(\tau)}(z_i) \leftarrow \sum_{z_{\mathcal{V}_i \setminus \{i\}}} \left( f_a(z_{\mathcal{V}_i}) \prod_{j \in \mathcal{V}_i \setminus \{i\}} m_{ja}^{(\tau-1)}(z_j) \right) \hspace{1em} \triangleright \text{fac. to var. mes.}
9: \hspace{2em} for i \in \mathcal{V} :
10: \hspace{3em} \hat{p}_{zi}(z_i) \leftarrow \prod_{a \in \mathcal{F}_i} m_{ai}^{(\tau^*)}(z_i) \hspace{1em} \triangleright \text{compute total belief, ie. marginals}
11: \hspace{1em} \text{return } \hat{p}_{z^n}

and denoting $m_{ij}^{(\tau)}(\cdot)$ as the message from node $i$ to node $j$, we can derive the message passing equations for pairwise undirected graphs as:

Algorithm 2.49 Belief Propagation on Pairwise Undirected Graphs

1: procedure PairwiseUndirectedBP(\mathcal{V}, \mathcal{E})
2: \hspace{1em} for (i, j) \in \mathcal{E} : \hspace{2em} \triangleright \text{initialization}
3: \hspace{2em} m_{ij}^{(0)}(z_j) \leftarrow 1
4: \hspace{1em} for \tau \leftarrow 1 .. \tau^*:
5: \hspace{2em} for (i, j) \in \mathcal{E} :
6: \hspace{3em} m_{ij}^{(\tau)}(z_j) \leftarrow \sum_{z_i} \left( \phi_i(z_i) \psi_{ij}(z_i, z_j) \prod_{k \in \mathcal{V}_i \setminus \{j\}} m_{ki}^{(\tau-1)}(z_i) \right) \hspace{1em} \triangleright \text{messages}
7: \hspace{2em} for i \in \mathcal{V} :
8: \hspace{3em} \hat{p}_{zi}(z_i) \leftarrow \phi_i(z_i) \prod_{j \in \mathcal{V}_i} m_{ji}^{(\tau^*)}(z_i) \hspace{1em} \triangleright \text{marginalization}
9: \hspace{1em} \text{return } \hat{p}_{z^n}

In both versions of the belief propagation, if any of the variables are continuous variables, we replace the sum with an integral of the corresponding variable.

As [17] has demonstrated, the iterative sum-product algorithm converges to the true belief (marginalization) within $D$ iterations on a tree graph, where $D$ is the diameter of the tree. The properties of running iterative sum-product on loopy graphs, known as \textit{loopy belief propagation}, is an open problem and is under active research in the machine learning community, but loopy belief propagation nonetheless converges.
reliably in most cases.

For either versions, the time complexity of each iteration of the algorithm will be linear in the product of the number of edges and the number of computation done at each summation. Therefore, for discrete variables, the time complexity will be $O(\tau^*|E| |\mathcal{E}|^2)$. $\tau^*$, the time to convergence, is $O(1)$ if the algorithm converges, although not necessarily the correct solution. Therefore, for an alphabet of a fixed size, the algorithm will be linear in the size of the input.

## 2.4 Summary

We introduced important concepts and tools that are building blocks for the data compression scheme presented in this thesis, including Gaussian distributions (Section 2.1), the class of distributions we seek to compress; coding theory for lossless and lossy schemes (Section 2.2), the framework with which we structure our compression scheme; and finally probabilistic graphical models (Section 2.3), the main tool we use for the decompressing algorithm.

With LDPC coding from telecommunications and the sum-product algorithm from machine learning, we are now ready to present our compression architecture.
Compressor Architecture Overview

In this chapter, we present the overview of the compressor architecture, which we will elaborate in details in later chapters. We first introduce the components of the system in Section 3.1, then describe the encoding scheme using these components in Section 3.2, before elaborating on the decoding algorithm in Section 3.3. We then discuss some practical concerns about decoding in Section 3.4 and about rate optimization in Section 3.5. Finally, we analyze the time and space complexity of the encoding and decoding algorithms and the communication costs in Section 3.6.

3.1 Architecture Components

As we described in Chapter 1.4, the primary focus of this thesis is to construct a lossy compression architecture that compress a sequence of Gaussian variables $s$ into a bit stream $x$. To achieve such a task, we require the following inputs to the algorithm: (i) a source model to represent the underlying correlation structure of the data (Section 3.1.1), (ii) a quantizer that maps a real value to a discrete value (Section 3.1.2), and (iii) a code that compresses the discrete values into a binary bit stream, which can be sub-divided into a translator that converts a set of discrete values into a binary representation (Section 3.1.3), and a binary code that compresses the translated binary representation into a shorter bit stream (Section 3.1.4).

3.1.1 Source Model

Let $n$ denote the length of the source sequence. The source model is the joint probability distribution $p_s$ from which $s^n$ is drawn, represented in the form of a probabilistic graphical model, introduced in Section 2.3. For ease of discussion, we represent it as a pairwise undirected graphical model (Section 2.3.1.1), noting that pairwise undirected models also has universal representation power, for all Gaussian sources can
be written with a factorization structure of

\[ p_s(s^n) \propto \prod_{i \in \mathcal{V}} \phi_i(s_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(s_i, s_j) \]  \hfill (3.1)

### 3.1.2 Quantizer

Essential to any lossy compression algorithm is quantization, the procedure of mapping a length-\( n \) source sequence \( s^n \) drawn from a large alphabet \( \mathcal{S}^n \), in our case \( \mathbb{R}^n \) which is uncountably large, into a length-\( m \) quantized sequence \( u^m \) from a smaller set \( \mathcal{U}^m \). In our case, we let the cardinality of the set to be \( |\mathcal{U}| := \beta = 2^b \), a power of 2, such that each \( u \) can be represented with \( b \) bits.

For our system, we draw the quantizer \( q(\cdot) \) from a collection \( \mathcal{Q} (n, m) \), such that each \( q \) is a function satisfying

\[ q : \mathcal{S}^n \to \mathcal{U}^m \]  \hfill (3.2)

which we write as

\[ u^m = q(s^n) \]  \hfill (3.3)

### 3.1.3 Translator

To represent a set of discrete values with 0s and 1s, we need a one-to-one mapping \( t_b \) that maps

\[ t_b : \mathcal{U} \to \mathbb{Z}_2^b \]  \hfill (3.4)

where \( \mathbb{Z}_2 \) represents the finite field of 2. As a slight abuse of notation, we allow \( t \) to operate on vectors \( u^m \), ie.

\[ z^{mb} = t_b(u^m) \]  \hfill (3.5)

with \( z^{mb} \) denoting the length-\( mb \) output vector of 0s and 1s. The subscript \( b \) of \( t_b \) will be dropped when unambiguous.

### 3.1.4 Binary Code

The last component of the algorithm would be the binary code, for which we drop the “binary” prefix for the remainder of the thesis when unambiguous. The code is a function \( h(\cdot) \) that maps length-\( mb \) bit stream into a shorter length-\( kb \) bit stream, an operation commonly known as hashing.

For our system, we draw the hashing functions \( h(\cdot) \) from a collection \( \mathcal{H} (mb, kb) \), such that each \( h \) is a function satisfying

\[ h : \mathbb{Z}_2^{mb} \to \mathbb{Z}_2^{kb} \]  \hfill (3.6)
which we write as

\[ x^{kb} = h(z^{mb}) \quad (3.7) \]

### 3.2 Encoder

The encoder, as we introduced in Section 1.4, is model-free, meaning that the encoding operation does not depend on the source model. By choosing suitable parameters \( m \) and \( k \), described above in Section 3.1, we achieve a *nominal compression rate* of

\[ r_{\text{code}} := \frac{kb}{n} \quad (3.8) \]

bits per sample.

![Diagram of Encode process](image)

**Figure 3.9:** The Encode procedure takes in a source sequence \( s^n \) as an input, and outputs \( x^{kb} \) as the compressed bit stream. Encode contains three components: Quantize, Translate, and Code.

For our particular encoder, we let the set of quantizers to be

\[ \mathcal{Q}(n, m) = \{ q : q(s^n) = q_{us}(Qs^n + Q_0) \} \quad (3.10) \]

where the set of \( Q \)'s and \( Q_0 \)'s we use are chosen from a subset of

\[ \{ Q : Q \in \mathbb{R}^{m \times n}, \max \deg(Q) \leq \xi \} \quad (3.11) \]
\[ \{ Q_0 : Q_0 \in \mathbb{R}^m \} \quad (3.12) \]

and where \( Qs^n \) denotes a matrix multiplication by a matrix \( Q \) and \( q_{us}(\cdot) \) represents the uniform scalar quantizer, defined to be

\[ q_{us}(x) = [x] \quad (3.13) \]

where we take the integer part each element in the vector.

For the translator \( t_b(\cdot) \), we shall focus on the \( b \)-bit gray code, which has the special
property that the encoding of adjacent numbers being off by exactly one bit. We write

\[ \text{tb}(u) = \text{vec}(\{\text{Gray}_b(u_i)\}_{i=1}^m) \] (3.14)

For the code, we focus on low-density parity-check (LDPC) codes, a class of linear codes introduced in Section 2.2.3, such that

\[ \mathcal{H}(mb, kb) = \{ h : h(z^{mb}) = Hz^{mb}; H \in \mathbb{Z}_2^{kb \times mb}, \max \deg(H) \leq \rho \} \] (3.15)

where the set of \( H \)'s we use are chosen from a subset of

\[ \{ H : H \in \mathbb{Z}_2^{kb \times mb}, \max \deg(H) \leq \rho, \min \deg(H) \geq 1 \} \] (3.16)

We will discuss each of these choices in the following chapters. Note that this is one particular incarnation of the architecture: the architecture that we described in the previous sections are general and different components can be switched out, given its modular nature. Different choices of quantizer, translator, and code will result in difference in the implementation details of the Encode and Decode algorithms, but the general structure of the pseudocode remains the same.

With our choice of quantizer, translator, and code, the Encode function is thus a very light-weight model-free encoder, described in pseudocode as follows:

Algorithm 3.17 Model-Free Encoder

1: **procedure** Encode\( (s^n; \mathbb{Q}, \mathcal{H}) \)
2: \( Q, Q_0 \leftarrow \text{Rand}(\mathbb{Q}(n, m)) \)  \( \triangleright \) randomly select quantization matrix
3: \( H \leftarrow \text{Rand}(\mathcal{H}(mb, kb)) \)  \( \triangleright \) randomly select LDPC matrix
4: \( u^m \leftarrow [Qs^n + Q_0] \)  \( \triangleright \) quantization
5: \( z^{mb} \leftarrow \text{vec}(\{\text{Gray}_b(u_i)\}_{i=1}^m) \)  \( \triangleright \) translation
6: \( x^{kb} \leftarrow Hz^{mb} \)  \( \triangleright \) hashing
7: **return** \( x^{kb} \)

The output of the algorithm, the length-\( kb \) vector \( x \), is the compressed bit stream, which the decoder (as we describe below in Section 3.3) decompresses to recover a source sequence \( \hat{s}^n \) that minimizes the expected value of the distortion measure \( E_x(\Delta(\hat{s}^n, s^n)) \).

## 3.3 Decoder

Given the irreversible nature of our lossy compression architecture, multiple source sequences \( s^n \) will map to the same compressed bit stream \( x^{kb} \). Using minimum squared error (MSE, Equation 2.30) as our distortion measure, our goal in the decoding step is to find the \( \hat{s}^n \) that minimize the expected distortion

\[ E_x(\Delta_{\text{MSE}}(\hat{s}^n, s^n)) = E_x(\frac{1}{n} \sum_{i=1}^n (\hat{s}_i - s_i)^2) \] (3.18)
which satisfies the constraints imposed by the input, the compressed vector $x^{kb}$.

![Decoding Diagram](image1)

**Figure 3.19:** The Decode procedure takes in the compressed sequence $x^{kb}$ as an input, and outputs $\hat{s}^n$ as the reconstruction. Decode also needs the code, translator, and quantizer components, as well as the source model, to reconstruct the original source sequence.

The decoder needs the constraints between $x^{kb}$ and $s^n$, as well as the relationship among the elements of $s^n$. This can be achieved by having an undirected graphical representation of $p_{s^n}$ and the factor graph representations of $q(\cdot)$, $t(\cdot)$, and $h(\cdot)$. We then combine these subgraphs into a joint graph, on which we run graphical inference to infer the estimate $\hat{s}^n$ that minimizes the MSE distortion.

![Joint Graph Diagram](image2)

**Figure 3.20:** The joint graph $\mathcal{G}$ has four subgraphs: source $\mathcal{C}$, quantizer $\mathcal{Q}$, translator $\mathcal{T}$, and code $\mathcal{H}$, with factor nodes within each of the latter three components. There are four sets of variable nodes: source $s$, quantized bins $u$, translated bits $z$, and hashed bits $x$. The relationships between the variables are embedded within the graph $\mathcal{G}$. 

45
3.3.1 Source Subgraph

The source subgraph $C$ contains prior knowledge of the source $p_s(\cdot)$. If $p_s(\cdot)$ is available in its algebraic form, then we can construct $C$ directly. If the prior knowledge is available through an undirected graphical model, we can transform it into a factor graph through the procedure described in Section 2.3.2, which has a factorization structure of

$$p_s(s^n) \propto \prod_{a \in C} f_a^c(s_{N^a})$$

(3.21)

In the special case that the source is a pairwise model, we can represent it with a pairwise undirected graphical model (Section 2.3.1.1) with factorization

$$p_s(s^n) \propto \prod_{i \in S} \phi_i^c(s_i) \prod_{(i,j) \in \mathcal{E}} \psi_{ij}^c(s_i, s_j)$$

(3.22)

which simplifies the decoding algorithm.

The source subgraph $C$ thus has variable nodes $S$, and we use subscripts $S_i$ to denote the $i^{th}$ node.

3.3.2 Quantizer Subgraph

The quantization procedure contains two steps: linear transform by matrix $Q \in \mathcal{Q}(n,m)$ and uniform scalar quantization $q_{un}(\cdot)$. The relationship between the source sequence $s^n$ and the quantized sequence $u^m$, also called the bin sequence, can be described as a factor graph. Given the linear transform $Q$, we note that each row of $Q$ corresponds to an output of the quantizer. Thus, we construct the factor graph $Q$ such that a quantizer output node $a$ is connected to source node $i$ if and only if $Q_{ai} \neq 0$. This $Q$ graph is called the quantizer subgraph.

In particular, the function $f_a^Q(\cdot)$ associated with each quantizer factor node is an indicator function that evaluates to 1 if and only if the quantizer constraints are met, i.e., the quantized source sequence satisfies the constraints imposed by $u^m$. Thus, the function $f_a^Q(\cdot)$ has the form

$$f_a^Q(s, u) = 1 \left\{ u_a = \left[ \sum_{i=1}^{n} Q_{ai} s_i + Q_{0a} \right] \right\}$$

(3.23)

This gives the unnormalized distribution over the quantizer subgraph, which is also called the quantizer constraint function:

$$c^Q(s^n, u^m) := \prod_{a=1}^{m} f_a^Q(s_{N^a}, u_a) = \prod_{j^Q \in \mathcal{F}_Q^Q} 1 \left\{ u_a = \left[ \sum_{i \in N^a} Q_{ai} s_i + Q_{0a} \right] \right\}$$

(3.24)

which evaluates to 1 if $s^n$ satisfies all constraints imposed by the bin sequence $u^m$, and
0 otherwise. Note that the sums in Equations 3.23 and 3.24 are equivalent because \( Q_{ai} = 0 \) for \( i \notin N_a^Q \).

The quantizer subgraph \( Q \) thus has variable nodes \( S \) and \( U \), and factor nodes \( \mathcal{F}^Q \).

### 3.3.3 Translator Subgraph

As described in Section 3.1.3, the translator \( t(\cdot) \) is a one-to-one mapping from \( U^m \) to \( Z_2^{mb} \). We can describe the relationship the bin sequence \( u^m \) and the translated sequence \( z^{mb} \) with a factor graph with \( m \) factor nodes representing the bin sequence and \( mb \) variables nodes representing the translated sequence, where a translated variable node \( i \) is connected to the bin factor node from which it is translated. We thus obtain the following factorization:

\[
c^{\mathcal{F}}(u^m, z^{mb}) := \prod_{a=1}^{m} f_a^{\mathcal{F}}(u_a, z_{N_a^T}) = \prod_{\mathcal{F}_a^{\mathcal{F}} \in \mathcal{F}} 1 \{ u_a = t^{-1}(z_{N_a^T}) \}
\]  

(3.25)

where \( t^{-1}(\cdot) \) is the inverse translator, the inverse of \( t(\cdot) \), which is uniquely defined due to the one-to-one nature of \( t(\cdot) \).

The translator subgraph \( \mathcal{F} \) thus has variable nodes \( U \) and \( Z \), and factor nodes \( \mathcal{F}^{\mathcal{F}} \).

### 3.3.4 Code Subgraph

The coding matrix \( H \) enforces the parity check bit constraints of

\[
x_a = \sum_{i=1}^{mb} H_{ai} z_i
\]  

(3.26)

where \( a \) is the index of the parity check bit, \( i \) is the index of the translated bits, with the addition done over \( Z_2 \). We can represent this relationship with a bipartite graph \( \mathcal{H} \), such that a parity check factor node \( a \) is connected to translated node \( i \) if and only if \( H_{ai} = 1 \). This \( \mathcal{H} \) graph is called the code subgraph. With this notation, we can rewrite the sum as

\[
x_a = \sum_{i \in N_a^{\mathcal{H}}} z_i
\]  

(3.27)

where \( N_a^{\mathcal{H}} \) represents the neighbors of \( a \) in the \( \mathcal{H} \) graph. With this constraint, we can assign a probability mass function over all possible translated sequences \( z^{mb} \), with sequences that satisfy the parity check constraints imposed by \( x \) having equal probabilities and those that do not satisfy the constraints having a probability of 0. The unnormalized distribution, which is also the hash constraint function, is thus

\[
c^{\mathcal{H}}(z^{mb}, x^{kb}) := \prod_{a=1}^{kb} f_a^{\mathcal{H}}(z_{N_a^{\mathcal{H}}}, x_a) = \prod_{\mathcal{F}_a^{\mathcal{F}} \in \mathcal{F}^{\mathcal{H}}} 1 \left\{ x_a = \sum_{i \in N_a^{\mathcal{H}}} z_i \right\}
\]  

(3.28)

47
where $1\{\cdot\}$ represents the indicator function. $c(z^{mb}, x^{kb})$ evaluates to 1 if $z^{mb}$ satisfies all constraints imposed by $x^{kb}$, and 0 otherwise.

The code subgraph $\mathcal{K}$ thus has variable nodes $Z$ and $X$, and factor nodes $\mathcal{F}^\mathcal{K}$.

### 3.3.5 Decoding Algorithm

With the subgraphs defined above, we are now ready to present the decoding algorithm. Let $\mathcal{G} = \mathcal{C} \cup \mathcal{Q} \cup \mathcal{F} \cup \mathcal{K}$ be the joint graph, with $\mathcal{C}$ as the source subgraph, $\mathcal{Q}$ as the quantizer subgraph, $\mathcal{F}$ as the translator subgraph, and $\mathcal{K}$ as the code subgraph. With this joint graph, the decoding algorithm is straight forward: we run loopy belief propagation, as introduced in Section 2.3.4.2, on the joint graph $\mathcal{G}$ until convergence. The overview of the belief propagation algorithm is listed in Algorithm 3.29, and the details of the message passing equations will be explained in the following chapters.

### 3.4 Doping Symbols

With the naive loopy belief propagation algorithm (Algorithm 3.29) presented in Section 3.3.5, depending on the source model, the trivial initial messages $m_{\mathcal{G}, \mathcal{G}'}^{(0)}(\cdot)$ may itself be a stable point of convergence of the algorithm, which may cause decoding to fail. Given this observation, the encoder may sometimes need to select a subset $\mathcal{D}$ of the unhashed $z^{mb}$ bits to send directly to the decoder for initialization. These set of bits $z_{q}$, known as the **dope bits**, are interpreted by the decoder as deterministic messages from the $Z_{q}$ nodes to the $F^{\mathcal{F}}$ and the $F^{\mathcal{K}}$ factor nodes:

\[
m^{(r)}_{z_i, z_{q}}(\cdot) = 1\{z_i = z_{i}\} \quad \forall i \in \mathcal{D}, \forall a \in N_{i}^{\mathcal{F}} \tag{3.30}
\]

\[
m^{(r)}_{z_i, z_{q}}(\cdot) = 1\{z_i = z_{i}\} \quad \forall i \in \mathcal{D}, \forall a \in N_{i}^{\mathcal{K}} \tag{3.31}
\]

which overrides the messages in Lines 3.29.16 and 3.29.18 of the loopy belief propagation algorithm.

This process of sending unhashed $z$ bits to the decoder, interpreted as deterministic messages, is known as **doping**, as introduced by [4]. The dope bits provide an anchor for the decoding process, and depending on the source model, only a fraction of the unhashed $z$ bits need to be doped. The **dope rate**, which is defined to be

\[
r_{\text{dope}} := \frac{|\mathcal{D}|}{n} \tag{3.32}
\]

can be tuned to optimize for the total rate, which is defined to be

\[
r := r_{\text{code}} + r_{\text{dope}} = \frac{kb + |\mathcal{D}|}{n} \tag{3.33}
\]

As we shall see in Chapter 4, doping is intricately tied to the choice of translators and codes. We shall discuss doping in more detail along with the discussion of code structure.
Algorithm 3.29 Decoding Algorithm: Belief Propagation on Joint Graph $\mathcal{G}$

1: procedure Decode($x, C, Q, T, \mathcal{H}$) 
2:   for $\tau \leftarrow 1 \ldots \tau^*$:  
3:     for $(S_i, S_j) \in \mathcal{E}_S$:  
4:       \[ m_{S_i, S_j}^{(\tau)}(s_j) \leftarrow \int_{s_i} \left( \phi_i^e(s_i) \psi_{ij}^e(s_i, s_j) \prod_{k \in \mathcal{N}_i^e \setminus \{j\}} m_{S_k, S_i}^{(\tau-1)}(s_i) \prod_{a \in \mathcal{N}_i^Q} m_{S_a, S_i}^{(\tau-1)}(s_i) \right) \]  
6:     for $(S_i, \mathcal{T}_a) \in \mathcal{E}_S, \mathcal{T}_a$:  
7:       \[ m_{S_i, \mathcal{T}_a}^{(\tau)}(s_i) \leftarrow \prod_{j \in \mathcal{N}_i^c} m_{S_j, S_i}^{(\tau-1)}(s_i) \prod_{a \in \mathcal{N}_i^Q} m_{S_a, S_i}^{(\tau-1)}(s_i) \]  
8:     for $(\mathcal{T}_a, U_a) \in \mathcal{E}_T, U_a$:  
9:       \[ m_{\mathcal{T}_a, U_a}^{(\tau)}(u_a) \leftarrow \int_{s_{\mathcal{T}_a}^Q \setminus \{i\}} \left( f_a^Q(s_{\mathcal{T}_a}^Q, u_a) \prod_{i \in \mathcal{N}_a^Q} m_{S_i, \mathcal{T}_a}^{(\tau-1)}(s_i) \right) \]  
10:   for $(U_a, \mathcal{T}_a) \in \mathcal{E}_U, \mathcal{T}_a$:  
11:     \[ m_{U_a, \mathcal{T}_a}^{(\tau)}(u_a) \leftarrow \prod_{S_i \in \mathcal{E}_S} m_{U_a, S_i}^{(\tau-1)}(u_a) \]  
12:   for $(\mathcal{T}_a, Z_i) \in \mathcal{E}_T, Z_i$:  
13:     \[ m_{\mathcal{T}_a, Z_i}^{(\tau)}(z_i) \leftarrow \sum_{\mathcal{Z}_a^\mathcal{N} \setminus \{i\}} \left( f_a^T(u_a, z_{\mathcal{N}_a}^\mathcal{T}) \prod_{i \in \mathcal{N}_a^Z} m_{Z_i, \mathcal{T}_a}^{(\tau-1)}(z_i) \right) \]  
14:   for $(\mathcal{Z}_i, \mathcal{T}_a) \in \mathcal{E}_Z, \mathcal{T}_a$:  
15:     \[ m_{\mathcal{Z}_i, \mathcal{T}_a}^{(\tau)}(z_i) \leftarrow \prod_{a \in \mathcal{N}_i^Z} m_{\mathcal{Z}_i, \mathcal{T}_a}^{(\tau-1)}(z_i) \]  
16:   for $(\mathcal{Z}_i, \mathcal{Z}_a^\mathcal{N}) \in \mathcal{E}_Z, \mathcal{Z}_a^\mathcal{N}$:  
17:     \[ m_{\mathcal{Z}_i, \mathcal{Z}_a^\mathcal{N}}^{(\tau)}(z_i) \leftarrow \prod_{a \in \mathcal{N}_i^Z \setminus \{\mathcal{N}_a\}} \left( f_a^Z(z_{\mathcal{N}_a}^\mathcal{Z}, x_a) \cdot \mathbb{1}\{x_a = x_a\} \prod_{j \in \mathcal{N}_a^Z \setminus \{i\}} m_{Z_j, \mathcal{Z}_a^\mathcal{N}}^{(\tau-1)}(z_j) \right) \]  
18:   for $S_i \in S$:  
19:     \[ \hat{p}_{S_i}(s_i) \leftarrow \prod_{a \in \mathcal{N}_i^Q} m_{S_i, S_a}^{(\tau*)}(s_i) \prod_{j \in \mathcal{N}_i^c} m_{S_j, S_i}^{(\tau*)}(s_i) \]  
20: return $\mathbb{E}_S(s^n)$
3.5 Rate-Distortion Optimization

The four subgraphs (source, quantizer, translator, code) of the joint graph $G$ can be divided into two portions, operating in different domains: (i) the continuous portion, which operates in $\mathbb{R}$ on the source subgraph $C$, and (ii) the discrete portion, which operates in $\mathbb{Z}_p$ on the translator subgraph $T$ with $p = 2^b$ and the code subgraph $H$ with $p = 2$.

As noted in Section 1.4, to the best of our knowledge, this work is the first attempt in using continuous-discrete hybrid belief propagation in the context of data compression. With a hybrid joint distribution over the graphical model, the quantizer subgraph $Q$ discretizes the continuous source and acts as a bridge for information flow between the two portions. Hence, in terms of the rate-distortion theory introduced in Section 2.2.4, the choice of the quantizer subgraph $Q$ is tied to the distortion $\delta$ of the reconstruction $\hat{s}^n$, while the choice of the code subgraph $H$ is tied to the rate $r$.

Given this observation, we note that rate and distortion can be optimized somewhat independently by optimizing the quantizer and the code, while still being bound by the rate distortion function (Definition 2.31). This means that for a fixed quantizer $q(\cdot)$, we can optimize the rate by tuning the code $h(\cdot)$.

3.5.1 Rate Selection

Similar to the Huang's system [14], our compression algorithm is a fixed rate system that does not inherently allow for feedback. This means that the encoder requires the compression rate $r$ to be supplied, implied by the size of the doped indices $\Omega$ and input-output ratios of the functions $q(\cdot), t(\cdot), h(\cdot)$. Therefore, we will need some estimate on the entropy $H(z^{mb})$ of the unhashed bits $z^{mb}$, so that the rate $r$ is chosen slightly above $E(z^{mb})$ to ensure correct decoding.

Such an estimate of entropy requires knowledge of the entropy of the source $s$, as well as the structure of the quantizer. However, for each source and each quantizer, the encoder only needs to know of the value of the estimated entropy, and the actual source and quantizer structures are not required. This still maintains the architectural separation between model and code.

If such prior knowledge of the entropy of the source is not available, another way to optimize for rate $r$ is to allow for feedback, whereby the hashed sequence $x^{kb}$ is sent letter by letter, until the decoder acknowledges sufficiency of information for reconstruction. This approach, mentioned by Huang [14], is justified by the conditional independence among the $x^{kb}$ bits conditioned on $z^{mb}$. This means that only the number of bits sent, but not the actual positional identity of the bits, matters for the purpose of decoding. This transforms our scheme into a rateless system. In fact, in a broadcast setting, even feedback in not necessary.

The feedback system above can be modified to serve in a data storage scenario. For a fixed source and quantizer, we can begin with a high rate, e.g. an uncompressed code $h(\cdot)$ and dope $\Omega$ with $kb + |\Omega| = mb$. As we later discover that the decoder can decode at a lower rate, we can truncate $x$ to a shorter length. This is again justified
by the conditional independence among $x_{\text{conditioned on} z}$.

If neither feedback nor source entropy is available, then we can simulate feedback by including the Decode algorithm in the encoder, as illustrated in Figure 3.34. Note that this does not break the model-code separation, for the Encode block is still model free, even though the encoder uses both Encode and Decode for compression. The fundamental advantage of model-code separation is not in designing the encoder and decoder separately, but in designing the coding structure to allow for separate optimization and flexibility.

### 3.6 Complexity

#### 3.6.1 Encode

**3.6.1.1 Time Complexity of Encode**

We first analyze the time complexity of the encoder. Quantization of $s^n$ to $u^m$ takes $O(\xi m)$ time, where $\xi$ is the maximum row weight of the quantization matrix $Q$. Translation of $u^m$ to $z^{mb}$ takes $O(mb)$ time. LDPC coding of $z^{mb}$ to $x^{kb}$ takes $O(\rho kb)$ time, where $\rho$ is the maximum row weight of the LDPC matrix $H$. With $\xi$ and $\rho$ chosen to be constants, the total complexity is thus

$$O(\xi m + mb + \rho kb)$$

If we choose the maximum row weight of the quantizer matrix and the code matrix to be $\xi = O(1)$ and $\rho = O(1)$, with $m = O(n)$ and $b = O(1)$, the time complexity for encoding will be linear in the input size $n$.

**3.6.1.2 Space Complexity of Encode**

For space complexity, we need to store the quantization matrix $Q$ which has $O(\xi m)$ elements, the translator which is a table with $O(2^b b)$ elements, and the LDPC matrix $H$ which has $O(\rho kb)$ elements. Hence, the space complexity on the encoder side is

$$O(\xi m + 2^b b + \rho kb)$$

which is also linear in $n$ with the choice of parameters as above.
3.6.2 Decode

3.6.2.1 Time Complexity of Decode

The time complexity of the decoder is dependent on the number of iterations till convergence $\tau^*$. As noted in Section 2.2.2, LDPC decoding over $\mathbb{Z}_2$ takes linear time. This time bound also holds true for the extended graph $\mathcal{G}$ that we constructed above, as we now show.

For each iteration, messages passing among the source nodes $S$ takes $O(I_e|E_e|)$, where $I_e$ is the time needed to calculate each integral. For univariate Gaussian variables, $I_e$ is $O(1)$ since there are closed form expressions for the Gaussian mean and variance updates. Message passing between the $S$ and the $U$ nodes has time complexity $O(I_q|E_{S,U}|)$, where $I_q$ is the time to evaluate the integral in the quantization messages, which is $O(|U|) \cdot O(1)$ given that there are $|U|$ bins for each $U$ node and that the messages from $S$ to $U$ are Gaussian. Message passing between the $U$ and the $Z$ nodes has time complexity of $O(mb)$, which is the cost of translation. Finally, message passing between $Z$ and $X$ costs $O(pmb)$, which is the LDPC decoding cost we discussed in Section 2.3.4.2. Hence, the total time complexity is

$$\tau^* \cdot (O(I_e|E_e|) + O(I_q|E_{S,U}|) + O(mb) + O(mb))$$

If we choose the maximum row weight of the quantizer matrix and the code matrix to be $\xi = O(1)$ and $\rho = O(1)$ respectively, with $m = O(n)$ and $b = O(1)$, the time complexity for decoding will be linear in the input size $n$, and linear in the source model complexity $|E_e|$, which is at most $O(n^2)$ for any source models.

3.6.2.2 Space Complexity of Decode

The space complexity of the decoder can be determined by analyzing the number of messages being passed in each iteration of message passing. Since a pair of messages is passed for each edge, we simply count the number of edges, noting that each message can be represented in $O(1)$ as we see in the following chapters. The source subgraph has $O(|E_e|)$ edges. The quantizer subgraph has $O(|U|\xi m)$ edges. The translator subgraph has $O(mb)$ edges. The code subgraph has $O(pkb)$ edges.

Hence, the space complexity of the decoder is

$$O(|E_e| + \xi m + mb + \rho mb)$$

which is again linear in the input size $n$ and linear in the source model complexity $|E_e|$, at most $O(n^2)$ for any source models.
3.6.3 Communication Costs

The decoder needs the compressed bit stream $x^{kb}$, the doped bits $z_\varphi$, the source model $p_{x^\varphi}(\cdot)$, the quantizer matrix $Q$, the translator $t(\cdot)$, and the code matrix $H$ for correct decoding.

We first note that the source model $p_{x^\varphi}(\cdot)$ is information available on the decoder side, not information that is sequence dependent. Next, the translator $t(\cdot)$ can be chosen to be a standardized code which does not need to be communicated from the encoder to the decoder. In addition, the randomly generated $Q$ and $H$ matrices can in practice be communicated to the decoder by sending the random seeds used to generate the matrix, which are of length $O(1)$, assuming that the decoder has the same matrix generating algorithm. This leaves the compressed stream $x^{kb}$ and the doped bits $z_\varphi$, which is of length

$$O(kb) + O(|2|) = O(rn)$$

3.7 Summary

In this chapter, we gave an overview to the different components of the compression architecture, namely (i) a source model, (ii) a quantizer function, (iii) a translator function, and (iv) a code function. We then proceeded to give the specific choice of each of the components we described, with linear transform followed by uniform scalar quantization as the quantizer function, gray code as the translator function, and LDPC encoding as the code function. With this specific implementation, we provided a graphical model description, and we presented the loopy belief propagation algorithm with the message passing equations for Decode. We concluded by introducing ideas for rate optimization and analyzing the time complexity of the Encode and the Decode algorithms.

In the following chapters, we shall first present the results of this compression scheme in Chapter 7. Then, we will explore each of the components of the algorithm in more detail and explain the different approximations, optimizations, and design choices of the implementation of the general architecture. Chapter 4 will be devoted to the code and translator components as well as doping, Chapter 5 will be devoted to the choice of quantizers, and Chapter 6 will be devoted to the discussion of source modeling.
Code and Translator Structure

While flexibility and modularity is the main priority of the general architecture of the compression algorithm, practicality in terms of compression rate and computing resources are also important concerns for our algorithm. For the algorithm to be practical, a tuning of the set of codes $\mathcal{H}(mb, kb)$ is indispensable. In this chapter, we shall discuss the details of the choice of codes in Section 4.1, then introduce different translators in Section 4.2, before discussing different doping schemes in Section 4.3. Finally, we present the details of the message passing equations in Section 4.4.

4.1 Code Parameters

In Section 2.2.2, we introduced the concept of linear codes for channel coding, which includes low-density parity-check (LDPC) codes. LDPC codes are of particular interest to us, given that its sparse nature will significantly bring down the coding complexity. In Section 2.2.3, we briefly discussed using LDPC codes for compression, noting the inherent duality of the channel coding problem and the compression problem.

LDPC code optimization is an active field of research itself, which is beyond the scope of this work; nevertheless, in this section, we introduce the parameters to tuning an LDPC code.

4.1.1 Row Weight and Column Weight

An LDPC code has two associated parameters $\varrho(\cdot)$ and $\varrho'(\cdot)$ known as degree distributions. The row degree distribution $\varrho(\cdot)$ is a probability distribution from which the row weights $\rho$ of the LDPC matrix $H$ are drawn, while the column degree distribution $\varrho'(x)$ is a probability distribution from which the column weights $\rho'$ of the LDPC matrix $H$ are drawn.

For our system, we fix the column degree distribution to be $\varrho'(x) = \Delta(x - 3)$, ie. the column weight is always $\rho \equiv 3$. We also fix the row degree distribution to be $\varrho'(x) = \Delta(x - 3 \cdot \frac{m}{k})$, ie. all rows will have the same weight.
4.1.2 Four Cycles in LDPC

Since the LDPC code is randomly generated, it is very likely that in its graphical representation, two hash nodes \( X_a, X_b \) are each connected to the same two translated bit nodes \( Z_i, Z_j \), which can be detected as a four cycle in the factor graph. Four cycles have the potential of adding redundant information to the hashed sequence \( x^{kb} \), whose effects on the compression rate are unclear. Therefore, we experimented with removing four cycles in the randomly generated code, an operation supported by an off-the-shelf LDPC code generator. We discuss the issue of four-cycles in relation to rate performance in Section 8.1.1.

4.2 Translator Selection

In Sections 3.1.3 and 3.3.3, we described the concept of a translator and its associated graphical model. The translator, as its name suggests, translates symbols from the alphabet \( \mathcal{U} = \{-2^{b-1}, \ldots, -1, 0, 1, \ldots, 2^{b-1} - 1\} \) to the alphabet \( \mathbb{Z}_2 = \{0, 1\} \). In Section 3.2, we made a choice to use the b-bit Gray code. We now describe two choices of translators: (i) the b-bit standard binary code and (ii) the b-bit Gray code.

4.2.1 Standard Binary Code

The most intuitive way to encode a symbol from the alphabet

\[
\mathcal{U} = \{-2^{b-1}, \ldots, -1, 0, 1, \ldots, 2^{b-1} - 1\}
\]

is with the b-bit standard binary code. There are two choices that can be made to code the negative numbers: (i) two’s complement, where the first bit is used as a sign bit to represent \(-2^b\), or (ii) add \(2^{b-1}\) to each symbol to make the sequence non-negative, on which we then use the standard binary representation.

4.2.2 Gray Code

Gray code is another way to encode \( \mathcal{U} = \{-2^{b-1}, \ldots, -1, 0, 1, \ldots, 2^{b-1} - 1\} \) with b-bits. Gray code has the special property that adjacent symbols in \( \mathcal{U} \) have a Hamming distance of exactly 1 in its coded \( \mathbb{Z}_2 \) representation. Given the 1-dimensional nature of the quantizer bins, we shall be using the PAM Gray code.

4.3 Doping

One important aspect with the translator is its ability to facilitate decoding. Since each translator will create a different distribution on the translated bits \( z^{mb} \), we need to be careful in initializing message passing in these bits.

As alluded to in Section 3.4, the belief propagation decoding will likely be at a point of convergence if all messages are initialized to be trivial, meaning there will...
be no dynamics or updates in the messages over iterations. To introduce dynamics into the system, a potential solution could be random initializations to disrupt the initial equilibrium, but in practice such random initialization may cause messages to be contradictory or and may start the algorithm in the wrong search space, which is difficult for an iterative algorithm to escape.

To solve this problem, we use doping so that we can start the decoding process around the neighborhood of the real solution, at the cost of increasing the total compression rate. In this Section, we explore the details of the mechanics of doping.

From the set of $Z$ nodes, we select a subset $\mathcal{D}$ of nodes to dope where $|\mathcal{D}| = n \cdot r_{dop}$. The doped bits are sent by the encoder along with the compressed bit stream to the decoder, which increases the total rate. The doped bits are interpreted as deterministic messages in the decoding process, which provides an anchor in both initialization and further iterations of belief propagation.

The choice of the bits being doped, however, will significantly affect the performance of the decoding algorithm in terms of the threshold compression rate for convergence. Here, we present three doping schemes and discuss their performance, namely (i) random doping, which chooses the bits to be doped randomly, (ii) sample doping, which selects size-$b$ groups of bits that each correspond to a quantized sample $u_a$, and (iii) lattice doping, which chooses the dope bits at regular intervals.

### 4.3.1 Random Doping

Random doping selects the set $\mathcal{D}$ of dope bits randomly. Random doping is most in line with the model-code separation. Random doping requires knowledge of neither the source model, nor the translator, nor the code. Random doping is extremely flexible in that the architecture allows for arbitrary dope rates for random doping.

Random doping, however, is suboptimal with respect to the compression rate. Intuitively, the distribution of the $Z$ bits are not random: its distribution is implicitly defined by the source model, the quantizer, and the translator. With knowledge of these components, we can choose our dope bits more wisely to decrease the total rate.

### 4.3.2 Sample Doping

![Sample Doping Diagram](image)

Figure 4.2: Sample doping, where all bits corresponding to whole samples are doped. The shaded nodes represent the bits doped.
Sample doping is on the other extreme of the spectrum of doping schemes. It chooses groups of bits, where each group corresponds to a quantized sample $u_a$. The choice of doped bits can be chosen with knowledge of the source, quantizer, translator, and code. However, this raises the concern of model-code separation by mixing source information into the decoder, something we actively try to avoid with our architecture.

### 4.3.3 Lattice Doping

A middle ground between random doping and sample doping is lattice doping, which does not use source information in deciding which bits to dope, but also achieves the purpose of giving each quantized sample $u_a$ an initialization. In lattice doping, we exploit the structure of the $b$ translator. Since every $b^{th}$ bit in the translated sequence $z_{mb}$ corresponds to the bit of the same significance in every sample samples (eg. every $b^{th}$ starting from the first bit always corresponds to the most significant bit in each sample), we can exploit this structure by doping bits at a regular interval. Depending on the properties of the choice of translator, we may want to dope different bits, eg. dope the most significant bit vs. dope the least significant bit. Figure 4.3 illustrates this concept.

While taking advantage of the knowledge of the system, lattice doping does not require knowledge of the model, rather, only the translator. This maintains the model-code separation while achieving better performance than random doping.

![Figure 4.3: Lattice doping, where the same bit of each sample is doped. The shaded nodes represent the bits doped.](image)

### 4.3.4 Lattice Doping Extensions

In the light of the discussion on lattice doping and the structure of the translator, multiple related ideas can be explored, including (i) multiple lattice doping, which dope multiple bits for each $b$-bit group that corresponds to a sample $u_a$, doping the same multiple bits for each group. Figure 4.4 illustrates this concept: for a 3-bit translator, we dope the second and third bits of every 3-bit group in this example.

However, it is not intuitively clear whether each of these extensions will contribute to better performance, for doping is very dependent on the choice of the translator. In particular, with multiple lattice doping, it is not clear whether an increase in dope rate $r_{dope}$ can be compensated for by the decrease in the nominal rate $r_{code}$ to decrease the total rate $r$. 

58
We shall present and discuss the empirical performance of each of these doping choices with respect to different translators in Section 8.2.

![Diagram](image)

Figure 4.4: Multiple lattice doping, in this example the first and the third bis are doped. The shaded nodes represent the bits doped.

### 4.4 Decoding Mechanics

In this section, we expand on the general message passing equations listed in Algorithm 3.29, focusing on the implementation of message passing in the code subgraph (lines 18–19 and 16) and the translator subgraph (lines 13 and 15).

#### 4.4.1 Message Passing for Code Subgraph

The message passing equations involving the translated nodes $Z$ are, from Algorithm 3.29,

$$
m_{Z_i, F_{Z_i}}^{(r)}(z_i) = m_{Z_i, F_{Z_i}}^{(r-1)}(z_i) \prod_{v \in N_{k,Z_i}^F} m_{Z_i, F_{Z_i}}^{(r-1)}(z_i) \tag{4.5}
$$

$$
m_{F_{Z_i}^F, Z_i}^{(r)}(z_i) := \sum_{x_a, z_{a \in N_{a}^Z}} m_{F_{Z_i}^F, z_{a \in N_{a}^Z}}^{(r)}(z_i, x_a) \cdot 1\{x_a = x_a\} \prod_{j \in N_{a \in N_{a}^Z}^Z \{i\}} m_{Z_j, F_{Z_j}^F}^{(r-1)}(z_j) \tag{4.6}
$$

$$
m_{Z_i, F_{Z_i}^Z}^{(r)}(z_i) := \prod_{a \in N_{a}^Z} m_{Z_i, F_{Z_i}^Z}^{(r-1)}(z_i) \tag{4.7}
$$

We observe that the hashed bits $x^{hb}$ comes into the equations through the term $1\{x_a = x_a\}$ in the $F_{Z_i}^Z$ to $Z$ messages, which gives a sequence $z^{mb}$ a probability of 0 if it does not satisfy the constraints imposed by $x^{hb} = h(z^{mb})$.

We first note that since the variables of the code graph is defined on the binary alphabet $Z_2 = \{0, 1\}$, we can simplify the message passing equations by propagating the log-likelihood ratio (LLR). Operating in the log domain has the benefit of numerical stability, given that Equations 4.5 and 4.6 both involves taking products of small
values. Hence, we define the LLR messages to be

\[ \tilde{m}_{ia}^{(r)} := \log \left( m_{z_i \mathcal{F}_a^c}^{(r)}(0) \right) - \log \left( m_{z_i \mathcal{F}_a^c}^{(r)}(1) \right) \]  
(4.8)

\[ \tilde{m}_{ai}^{(r)} := \log \left( m_{\mathcal{F}_a^c z_i}^{(r)}(0) \right) - \log \left( m_{\mathcal{F}_a^c z_i}^{(r)}(1) \right) \]  
(4.9)

and we re-write the incoming messages from the translator graph as

\[ \tilde{\phi}_i^{(r)} := \log \left( m_{\mathcal{F}_c z_i}^{(r-1)}(0) \right) - \log \left( m_{\mathcal{F}_c z_i}^{(r-1)}(1) \right) \]  
(4.10)

---

**Figure 4.11:** Message passing on the code subgraph: the orange arrow is a \( z \) to \( \mathcal{F}_c^c \) message, the blue arrow is a \( \mathcal{F}_c^c \) to \( z \) message, and the red arrow is a \( z \) to \( \mathcal{F}_c^c \) message. Note that the information about the compressed bit stream comes into the system only through the red messages.

With these notations, we take the difference of the log of Equation 4.5, we get

\[ \tilde{m}_{ia}^{(r)} = \tilde{\phi}_i^{(r-1)} + \sum_{b \in \mathcal{N}_i^{\mathcal{F}_c^c} \setminus \{a\}} \tilde{m}_{bi}^{(r-1)} \]  
(4.12)

For Equation 4.6, we note by the normalization property that

\[ m_{\mathcal{F}_a^c z_i}^{(r)}(0) + m_{\mathcal{F}_a^c z_i}^{(r)}(1) = 1 \]  
(4.13)

\[ m_{z_i \mathcal{F}_a^c}^{(r)}(0) + m_{z_i \mathcal{F}_a^c}^{(r)}(1) = 1 \]  
(4.14)

and that

\[ m_{\mathcal{F}_a^c z_i}^{(r)}(0) - m_{\mathcal{F}_a^c z_i}^{(r)}(1) = \tanh \left( \frac{\tilde{m}_{ai}^{(r)}}{2} \right) \]  
(4.15)

\[ m_{z_i \mathcal{F}_a^c}^{(r)}(0) - m_{z_i \mathcal{F}_a^c}^{(r)}(1) = \tanh \left( \frac{\tilde{m}_{ai}^{(r)}}{2} \right) \]  
(4.16)

\[ m_{\mathcal{F}_a^c z_i}^{(r)}(0) - m_{\mathcal{F}_a^c z_i}^{(r)}(1) = \prod_{j \in \mathcal{N}_a^{\mathcal{F}_c^c} \setminus \{i\}} \left( m_{\mathcal{F}_a^c z_i}^{(r)}(0) - m_{\mathcal{F}_a^c z_i}^{(r)}(1) \right) \]  
(4.17)
we have the LLR messages to be

$$
\tilde{m}_{ai}^{(r)} = 2 \cdot \tanh \left( \prod_{j \in N_a^k \setminus \{i\}} \tanh \frac{\tilde{m}_{ija}^{(r-1)}}{2} \right) \cdot (-1)^{1[x_a=1]} \tag{4.18}
$$

where \( \tanh(x) := \frac{e^x - e^{-x}}{e^x + e^{-x}} \) denotes the hyperbolic tangent function and \( \text{atanh}(x) \) denotes the inverse hyperbolic tangent function.

The final output \( Z \) to \( V \) message will then be

$$
m_{Z_t}^{(r)} = \prod_{a \in N_t^k} m_{Z_t}^{(r-1)}(z_i)
\quad (4.19)
$$

$$
= \frac{1}{2} \left( 1 - \tanh \left( \frac{1}{2} \sum_{a \in N_t^k} \tilde{m}_{ia}^{(r-1)} \right) \right) \tag{4.20}
$$

### 4.4.2 Message Passing for Translator Subgraph

The messages involving the translator factor nodes \( \mathcal{F}^T \) are, from algorithm 3.29.13 and 3.29.15

$$
m_{\mathcal{F}^T u_a}^{(r)}(u_a) := \sum_{z_{N_a^T}} f_{a}^{(r)}(u_a, z_{N_a^T}) \prod_{i \in N_a^k} m_{z_i}^{(r-1)}(z_i) \tag{4.21}
$$

$$
m_{\mathcal{F}^T z_i}^{(r)}(z_i) := \sum_{u_a, z_{N_a^T} \setminus \{i\}} \left( f_{a}^{(r)}(u_a, z_{N_a^T}) \cdot m_{u_a}^{(r-1)}(u_a) \prod_{j \in N_a^k \setminus \{i\}} m_{z_j}^{(r-1)}(z_j) \right) \tag{4.22}
$$

which are straight forward to implement.

An important note is that \( z_i \in \mathbb{Z}_2 \) and \( u_a \in \mathcal{U} \cong \mathbb{Z}_{2^b} \). Hence, the \( \mathcal{F}^T \) to \( \mathcal{U} \) messages will be an \( m \times 2^b \) table, with the \((a, k)^{th}\) entry signifying the estimated probability that quantized sample \( u_a \) is in that particular bin \( k \in \mathcal{U} \).

On the other hand, the \( \mathcal{F}^T \) to \( \mathcal{Z} \) messages will be an \( mb \times 2 \) table, with the \((i, c)^{th}\) entry signifying the estimated probability that translated bit \( z_i \) is equal to \( c \in \mathbb{Z}_2 \).
4.5 Summary

In this chapter, we have discussed various aspects of code and translator selections. We introduced the different methods of doping, the results of which we will discuss in Chapter 8. Finally, we provided the implementation details that solves the practical concerns of message passing in the code and translator subgraphs.
Quantizer Structure

A quantizer is a function that takes in a continuous value or vector and outputs a discrete value or vector. As described in 3.1.2, a quantizer is necessary for any lossy compression system, and it holds a central place in our architecture by implicitly controlling the rate-distortion trade-off, as we shall see in the following sections.

In this chapter, we first describe two ideas for the structure of our quantizer in Section 5.1. Then, we describe our particular choice of quantizer and elaborate on the message passing equations associated with our choice of quantizer in Section 5.2, detailing the approximations we use to keep the algorithm efficient and tractable. Finally, in Section 5.3, we note the flexibility of the architecture in allowing for component-wise optimizations whose structural changes do not affect that of the other components.

5.1 Quantizer Selection

Here, first we present two quantizers which can be adapted into the general architecture of graphical model decoding. In particular, the sparse structure of these quantizers allow for efficient computation and thus decoding.

5.1.1 Uniform Scalar Quantizer

A most simple quantizer would be the uniform scalar quantizer

\[ q_{us}(x) = \lfloor x \rfloor \]

(5.1)

which simply rounds down the input value. The corresponding quantizer subgraph would be a set of \( n \) non-interacting nodes with edges connecting the source and the translator.

While the simplicity of the quantizer allows for efficient computation, there are two immediate issues that arise: (i) not taking advantage of the correlations the sample values, and (ii) the lack of flexibility to provide different degrees of refinement. While
the first issue of not using correlation does impact performance, it turns out this is not too much of a concern, with two reasons. First, the code \( h(\cdot) \) already takes advantage of underlying correlation, so the actual performance loss would intuitively be mitigated. Second, using source correlation in the quantizer design, which is a part of the \textit{Encode} algorithm, violates the code-model separation, as it uses knowledge of the model in the process of encoding. The more relevant concern is thus the second issue, the rigid nature of the quantizer.

### 5.1.2 Low Density Hashing Quantizer

With the above concerns in mind, we shall use a low density hashing quantizer, which gives the quantizer more flexibility in terms of precision, but without using any knowledge about the correlation or variance of the underlying source. One thing should be noted, that all quantizers need to know the mean of the sequence it is quantizing to provide reasonable performance, for without such knowledge it would be impossible to even device a quantizer with a bounded length output.

As alluded to in Section 3.2, a low density hashing quantizer resembles an LDPC code in structure. In particular, we shall do a sparse linear transform of the source sequence before applying the uniform scalar quantizer, mathematically defined by

\[
u^m = q_{us}(Qs^n + Q_0)
\]  

(5.2)

where \( Q \in \mathbb{R}^{m \times n} \) is a sparse matrix with maximum row weight \( \xi \) and \( Q_0 \in \mathbb{R}^m \) is a shift vector. We note that the uniform scalar quantizer is a special case of the low density hashing quantizer, with \( Q = I \) and \( Q_0 = 0 \).

Note that a matrix multiplication is not the only operation possible. While maintaining the low density structure, the function \( q_a(\cdot) \) for factor \( a \) can be changed. In the case of matrix multiplication, we chose

\[
q_a(s; Q, Q_0) = \sum_{i \in \mathbb{N}_Q^0} Q_{ai}s_i + Q_{0a}
\]

(5.3)

Other functions can replace this particular choice of \( q_a(\cdot) \) and the general structure of the algorithm will not change, although the message passing equations implementation will be dependent on the choice of \( q_a(\cdot) \).

#### 5.1.2.1 Tuning of \( m \) and the Sub-1-Bit Regime

We note while \( m \) provides another parameter for fine-tuning the compression rate, either with \( m < n \) for compression for \( m > n \) for better precision, often times it is not necessary, for the code \( h(\cdot) \) and the translator \( t(\cdot) \) already serves similar functions. Therefore, generally we will keep \( m = n \). The only scenario in which the value of \( m \) plays a role is in the sub-1 bit regime, in which the compression rate \( r < 1 \). In this case, the translator \( t(\cdot) \) will not be able to provide any further lower precision, and the code \( h(\cdot) \) will find it difficult to provide further lower compression rate below 1
with a translation bit length of 1. In this case, we can change the $m$ parameter to compression the source with the quantizer before passing it onto the translator and code components. This operation, however, is still not too well understood and more research needs to be done with compressing in the sub-1-bit regime.

5.1.2.2 Practical Concerns and Implications

Of the various configurations of the low density hashing quantizer that we experimented with, we find the most success in having $Q$ as a diagonal matrix, meaning a quantizer graph with a set of non-interacting nodes. This, however, is different from the quantizer presented in Section 5.1.1 in that the quantizing width, instead of being uniformly 1, can be different for each element, as defined as the inverse of the diagonal entries of the matrix $Q$. By adjusting the width, we can fine tune the rate-distortion trade-off: a larger width means less entropy in the quantized sequence $u^m$ hence more compression, but at a cost of a less precise reconstruction; while a smaller width gives more entropy to $u^m$ hence less compression, but a more precise reconstruction. Note that regardless of the source variance, this scheme provides a level of performance parallel to the rate-distortion curve, hence this quantizer is source agnostic. To achieve a particular compression rate or a particular distortion though, the width of the quantizer will need to be chosen with the knowledge of the source variance, which is something that no system can avoid.

Despite the simple structure we chose for $Q$ as a diagonal matrix, the following development of the message passing equations is general for any quantizer that follows the transform-then-quantize pattern, ie. general $Q$ and $Q_0$.

5.2 Decoding Mechanics

As described above, the uniform scalar quantizer is a special case of the low density hashing quantizer, with the quantizer matrix as $Q = I_\alpha$ and the maximum row weight of $Q$ as $\xi = 1$.

As described in Algorithm 3.29 lines 6, 7, 9, and 10, the message passing equations on the quantizer nodes $F^Q$ are

$$m_{\gamma_i,s_i}^{(\tau)}(s_i) := \prod_{\nu \in \mathcal{N}_i^Q \setminus \{a\}} m_{\nu_i,s_i}^{(\tau-1)}(s_i) \prod_{j \in \mathcal{N}_i^Q} m_{j:s_i}^{(\tau-1)}(s_i)$$  \hspace{1cm} (5.4)

$$m_{\gamma_a:u_a}^{(\tau)}(u_a) := \int_{s_{\mathcal{N}_a^Q}} f_a^Q(s_{\mathcal{N}_a^Q}, u_a) \prod_{i \in \mathcal{N}_a^Q} m_{i:s_i}^{(\tau-1)}(s_i)$$ \hspace{1cm} (5.5)

$$m_{u_a:\gamma_a}^{(\tau)}(u_a) := m_{\gamma_a:u_a}^{(\tau-1)}(u_a)$$ \hspace{1cm} (5.6)

$$m_{\gamma_i:s_i}^{(\tau)}(s_i) := \sum_{u_a} \int_{s_{\mathcal{N}_a^Q \setminus \{i\}}} f_a^Q(s_{\mathcal{N}_a^Q}, u_a) \cdot m_{u_a:\gamma_a}^{(\tau-1)}(u_a) \prod_{j \in \mathcal{N}_a^Q \setminus \{i\}} m_{j:s_j}^{(\tau-1)}(s_j)$$ \hspace{1cm} (5.7)
and the messages among the source nodes $\mathcal{S}$ are (Algorithm 3.29.4)

$$m_{\mathcal{S}_i \mathcal{S}_j}^{(r)}(s_j) := \int_{s_i} \left( \phi_i^{C}(s_i) \psi_{ij}^{C}(s_i, s_j) \prod_{k \in N_i \setminus \{j\}} m_{\mathcal{S}_k \mathcal{S}_i}^{(r-1)}(s_i) \prod_{a \in N_i^{Q}} m_{\mathcal{S}_a \mathcal{S}_i}^{(r-1)}(s_i) \right)$$  \hspace{1cm} (5.8)

We first note that the messages among the source nodes $\mathcal{S}$ are Gaussian. To maintain the messages as Gaussian after the first iteration, we would need the messages from $\mathcal{F}^Q$ to $\mathcal{S}$ to be Gaussian. However, the $\mathcal{F}^Q$ to $\mathcal{S}$ messages described in Equation 5.7 are integrals over regions that satisfy the binning constraints. Therefore, we propose an approximation to maintain the Gaussian nature of the messages, as described below:

![Figure 5.9: Message passing on the quantizer subgraph: the red arrow is a $\mathcal{S}$ to $\mathcal{F}^Q$ message, the blue arrow is a $\mathcal{F}^Q$ to $\mathcal{S}$ message, the brown arrow is a $\mathcal{U}$ to $\mathcal{F}^Q$ message, the orange arrow is a $\mathcal{F}^Q$ to $\mathcal{U}$ message.]

5.2.1 The $\mathcal{F}^Q$ to $\mathcal{S}$ Messages

The integral in Equation 5.7 is taken over all variables except $i$, the receiver node of the message. The first part of the integrand is

$$f_a^{Q}(s_{N_i^{Q}}, u_a)$$  \hspace{1cm} (5.10)

which evaluates to 1 if and only if the $q(s_{N_i^{Q}}) = u_a$, i.e. the $s$ sequences hashes to the correct bin. This implicitly defines the limits of integration. For the low density hashing quantizer for which $q(s) = [Q_0 + Qs]$. The set over which we integrate is thus

$$\left\{ s_{N_i^{Q}\setminus\{i\}} : u_a \leq Q_{0a} + \sum_{j \in N_i^{Q}\setminus\{i\}} Q_{aj}s_j < u_a + 1 \right\}$$  \hspace{1cm} (5.11)

The second part of the integrand is the $\mathcal{U}$ to $\mathcal{F}^Q$ message

$$m_{\mathcal{U}_a \mathcal{F}^Q}^{(r-1)}(u_a) =: p_{\mathcal{U}a}^{(r-1)}(u_a)$$  \hspace{1cm} (5.12)
which represents the probabilities of each of the $2^b$ bins, estimated at iteration $\tau - 1$. The last part of the integrand

$$
\prod_{j \in N_0^Q \setminus \{i\}} m_{s_j, T_0 \sigma_0}^{(\tau-1)}(s_j) 
$$

is the product of the Gaussian $S$ to $T_0$ messages each in a different variable, hence this term is a joint Gaussian.

With these observations, we can re-write the message as

$$
m_{s_i, T_0}^{(\tau)} = \sum_{u_a} \int_{s_N \setminus \{i\}} \left( f_a^{T_0}(s_{N_0^Q}, u_a) \cdot m_{u_a, T_0}^{(\tau-1)}(u_a) \prod_{j \in N_0^Q \setminus \{i\}} m_{s_j, T_0}^{(\tau-1)}(s_j) \right) 
$$

which is a weighted sum of integrals of a joint Gaussian over parallel $(\xi - 1)$-dimensional slices of the $\mathbb{R}^\xi$ space.

### 5.2.1.1 Gaussian Approximations for $T_0$ to $S$ Messages

We note that the integral of a joint Gaussian over a $(\xi - 1)$-dimensional slice can be expressed as the difference between two Gaussian cdf’s whose pdf’s have the same variance but different means. The mean and variance of the pdf’s associated with the two Gaussian cdf’s are

$$
\mu_{cdf1, u_a} = -\frac{1}{Q_{ai}} \left( Q_{0a} + \sum_{j \in N_0^Q \setminus \{i\}} Q_{aj} \mu_j \right) + \frac{1}{Q_{ai}} u_a 
$$

$$
\mu_{cdf2, u_a} = -\frac{1}{Q_{ai}} \left( Q_{0a} + \sum_{j \in N_0^Q \setminus \{i\}} Q_{aj} \mu_j \right) + \frac{1}{Q_{ai}} (u_a + 1) 
$$

and the variance is

$$
\sigma_{cdf}^2 = \frac{1}{Q_{ai}^2} \sum_{j \in N_0^Q \setminus \{i\}} Q_{aj}^2 \sigma_j^2 
$$

where $\mu_j$ and $\sigma_j^2$ are the mean and the variance of the $S$ to $T_0$ Gaussian messages.

We note that when $\frac{1}{Q_{ai}}$ is small, the difference of two Gaussian cdf’s can be well approximated with a Gaussian pdf. In particular, when $\frac{1}{Q_{ai}}$ approaches 0, the difference of the two cdf’s approaches the Gaussian pdf, since by definition the pdf is the derivative of the cdf.

The center point of the difference will be the average of the two means. However, when $\frac{1}{Q_{ai}}$ is non-0, we need to adjust the variance by adding the variance of a
uniform distribution with the same width \( \frac{1}{\tilde{Q}_{ai}} \), which has the form \( \frac{u^2}{12} \). Hence, if we approximate the difference with a Gaussian pdf, the parameters will be

\[
\mu_{\text{diff},ua} = -\frac{1}{\tilde{Q}_{ai}} \left( Q_{0a} + \sum_{j \in \mathbb{N}^a_0 \setminus \{i\}} Q_{aj} \mu_j \right) + \frac{1}{\tilde{Q}_{ai}} \left( u_a + \frac{1}{2} \right) 
\] (5.19)

\[
\sigma^2_{\text{diff}} = \frac{1}{\tilde{Q}_{ai}^2} \sum_{j \in \mathbb{N}^a_0 \setminus \{i\}} Q_{aj}^2 \sigma^2_j + \frac{1}{12\tilde{Q}_{ai}^2} 
\] (5.20)

The final message is a weighted sum of these integrals over slices. With the above approximation, this will thus be a Gaussian mixture. Since we require the message to be a Gaussian, we do one final approximation by approximating a Gaussian mixture as a single Gaussian, with its mean and variance being the mean and variance of the mixture. Thus, we pass the messages in information form as

\[
m_{\mathcal{G}_a S_i}(s_i) \approx \mathcal{N}^{-1}(s_i; \eta_{\mathcal{G}_a S_i}, \lambda_{\mathcal{G}_a S_i}) = \mathcal{N}(s_i; \mu_{\mathcal{G}_a S_i}, \sigma^2_{\mathcal{G}_a S_i}) 
\] (5.21)

where

\[
\lambda_{\mathcal{G}_a S_i} = \left( \sigma^2_{\mathcal{G}_a S_i} \right)^{-1} 
\] (5.22)

\[
\eta_{\mathcal{G}_a S_i} = \lambda_{\mathcal{G}_a S_i} \mu_{\mathcal{G}_a S_i} 
\] (5.23)

with

\[
\mu_{\mathcal{G}_a S_i} = \sum_{u_a = -2^{b-1}}^{2^{b-1}-1} \hat{\mu}_{u_a}(u_a) \cdot \mu_{\text{diff},u_a} 
\] (5.24)

\[
= -\frac{1}{\tilde{Q}_{ai}} \left( Q_{0a} + \sum_{j \in \mathbb{N}^a_0 \setminus \{i\}} Q_{aj} \mu_j \right) + \frac{1}{\tilde{Q}_{ai}} \left( \sum_{u_a} u_a \cdot \hat{\mu}_{u_a}^{(r-1)}(u_a) + \frac{1}{2} \right) 
\] (5.25)

and

\[
\sigma^2_{\mathcal{G}_a S_i} = \sigma^2_{\text{diff}} + \sum_{u_a = -2^{b-1}}^{2^{b-1}-1} \hat{\mu}_{u_a}(u_a) \cdot \mu_{\text{diff},u_a}^2 - \left( \sum_{u_a = -2^{b-1}}^{2^{b-1}-1} \hat{\mu}_{u_a}(u_a) \cdot \mu_{\text{diff},u_a} \right)^2 
\] (5.26)

\[
= \frac{1}{\tilde{Q}_{ai}^2} \sum_{j \in \mathbb{N}^a_0 \setminus \{i\}} Q_{aj}^2 \sigma^2_j + \frac{1}{12\tilde{Q}_{ai}^2} + \frac{1}{\tilde{Q}_{ai}^2} \left( \sum_{u_a} u_a^2 \cdot \hat{\mu}_{u_a}(u_a) - \left( \sum_{u_a} u_a \cdot \hat{\mu}_{u_a}(u_a) \right)^2 \right) 
\] (5.27)

where

\[
\hat{\mu}_{u_a}(u_a) := m_{\mathcal{G}_a S_i}^{(r-1)}(u_a) 
\] (5.28)

68
as defined in Equation 5.12.

5.2.2 The $S$ to $\mathcal{F}^Q$ Messages

With the $\mathcal{F}^Q$ to $S$ messages approximated as Gaussians, it follows that the messages among $S$ (Equation 5.8) are also Gaussian. Hence, the messages from $S$ to $\mathcal{F}^Q$ (Equation 5.4) are also Gaussian, and thus Equation 5.4 can be expressed as

$$m_{S_{i:j}^Q}(s_i) := \prod_{b \in \mathcal{N}_i^Q \setminus \{a\}} m_{\mathcal{F}_b^Q S_i}^{(\tau-1)}(s_i) \prod_{j \in \mathcal{N}_i^c} m_{S_j S_i}^{(\tau-1)}(s_i)$$

$$= \prod_{b \in \mathcal{N}_i^Q \setminus \{a\}} \mathcal{N}^{-1}(s_i; \eta_{\mathcal{F}_b^Q S_i}^{(\tau-1)}, \lambda_{\mathcal{F}_b^Q S_i}^{(\tau-1)}) \prod_{j \in \mathcal{N}_i^c} \mathcal{N}^{-1}(s_i; \eta_{S_j S_i}^{(\tau-1)}, \lambda_{S_j S_i}^{(\tau-1)})$$

$$= \mathcal{N}^{-1}(s_i; \eta_{S_i}^{(\tau)}, \lambda_{S_i}^{(\tau)})$$

where

$$\eta_{S_i}^{(\tau)} = \sum_{b \in \mathcal{N}_i^Q \setminus \{a\}} \eta_{\mathcal{F}_b^Q S_i}^{(\tau-1)} + \sum_{j \in \mathcal{N}_i^c} \eta_{S_j S_i}^{(\tau-1)}$$

$$\lambda_{S_i}^{(\tau)} = \sum_{b \in \mathcal{N}_i^Q \setminus \{a\}} \lambda_{\mathcal{F}_b^Q S_i}^{(\tau-1)} + \sum_{j \in \mathcal{N}_i^c} \lambda_{S_j S_i}^{(\tau-1)}$$

5.2.3 The $\mathcal{F}^Q$ to $U$ Messages

The messages from $\mathcal{F}^Q$ to $U$, reproduced from Equation 5.5,

$$m_{\mathcal{F}_a^Q U_a}(u_a) := \int_{s_{\mathcal{F}_a^Q}} \left( f_a^Q(s_{\mathcal{F}_a^Q}, u_a) \prod_{i \in \mathcal{N}_a^Q} m_{S_i \mathcal{F}_a^Q}^{(\tau-1)}(s_i) \right)$$

are the integrals of a Gaussian, with integration limits implicitly defined by the quantizer factors $f_a^Q(s_{\mathcal{F}_a^Q}, u_a)$. For each $u_a$, the integral can thus be expressed in terms of the standard Gaussian cdf. To simplify our notation, let

$$\hat{p}_{u_a}(u_a) := m_{\mathcal{F}_a^Q U_a}(u_a)$$

$$\mathcal{N}(s, \mu_a, \Sigma_a) := \prod_{i \in \mathcal{N}_a^Q} m_{S_i \mathcal{F}_a^Q}^{(\tau-1)}(s_i)$$

where $\mu_a$ is a vector and $\Sigma_a$ is a diagonal matrix.
Letting $Q_a$ denote the $a^{th}$ row of $Q$, we have

$$
\hat{p}_{ua}(u_a) = \int_{\{s: u_a \leq Q_a s + Q_0a < u_a + 1\}} \mathcal{N}(s, \mu_a, \Sigma_a) \quad (5.37)
$$

$$
= \int_{\{s: Q_a s + Q_0a - u_a \geq 0\}} \mathcal{N}(s, \mu_a, \Sigma_a) - \int_{\{s: Q_a s + Q_0a - (u_a + 1) \geq 0\}} \mathcal{N}(s, \mu_a, \Sigma_a) \quad (5.38)
$$

Noting that the integral is taken over a half-space, we do a change in coordinates to convert it into the standard single variable Gaussian form, the integrals for which we can express in terms of the standard Gaussian cdf $\Phi(\cdot)$:

$$
\int_{\{s: Q_a s + Q_0a - u_a \geq 0\}} \mathcal{N}(s, \mu_a, \Sigma_a) \quad (5.40)
$$

$$
= \int_{\{s': \Sigma_a^{1/2} Q_a s' + (Q_a \mu_a + Q_0a - u_a) \geq 0\}} \mathcal{N}(s', 0, I_n) \quad (5.41)
$$

$$
= \int_{\{s'': c_a \leq s'' < \infty\}} \mathcal{N}(s'', 0, 1) \quad (5.42)
$$

$$
= 1 - \Phi(c_a) \quad (5.43)
$$

where the scalar $c_a$ is

$$
c_a := - \frac{Q_a \mu_a + Q_0a - u_a}{\|\Sigma_a^{1/2} Q_a\|_2} \quad (5.44)
$$

with $\|\cdot\|_2$ denoting the $L^2$ norm. Hence, we have

$$
m^{(r)}_{\mathcal{F}^Q u_a}(u_a) = \Phi \left( \frac{- Q_a \mu_a + Q_0a - (u_a + 1)}{\|\Sigma_a^{1/2} Q_a\|_2} \right) - \Phi \left( \frac{- Q_a \mu_a + Q_0a - u_a}{\|\Sigma_a^{1/2} Q_a\|_2} \right) \quad (5.45)
$$

as the $\mathcal{F}^Q$ to $U$ messages.

### 5.3 Summary

In this chapter, we have discussed two types of quantizers and their advantages and disadvantages, in addition to the practical concerns in each of them. We then derived the message passing equations for our choice of low density hashing quantizer, the equations of which also works for any general quantizer with the transform-then-quantize pattern. We note that with the structure of our general architecture, with
the quantizer being one modular part of the system, the quantizer itself can be optimized and replaced individually without affecting any other parts of the system, a property that is rare among existing lossy compression algorithms. This modularity and flexibility of the architecture renders it ready for change.
Source Modeling

As described in Section 3.1.1, the source model refers to the underlying probability distribution $p_{sn}$ that generates the source sequence $s^n$. In Section 3.3.1 we briefly discussed the graphical model representation of the source model based on its factorization structure, which is formalized as the source subgraph portion $\mathcal{C}$ of the full graph $\mathcal{G}$. In this chapter, we will explore the source modeling details. In Section 6.1, we will describe in details two source models, namely (i) the Gaussian iid source, and (ii) the Gauss-Markov source, in which we will present their respective graphical model representation. Then, in Section 6.2, we describe their message passing equations and marginalization, an operation which gives us an estimate $\hat{s}^n$. Finally, we note the universal nature of the source subgraph, an immediate advantage of our architecture that allows for compression of general sources.

6.1 Graphical Model Representation of Source Models

Probabilistic graphical models, introduced in Section 2.3, are universal in their ability to represent any probability distributions. In this section, we present two source models with relatively simple graphical representations:

6.1.1 Gaussian iid Source

A Gaussian iid source $\mathcal{N}^{-1}(0, \lambda^s I)$, with $\lambda^s$ being the inverse of the variance, is distributed according to

$$p_{sn}(s^n) = \mathcal{N}^{-1}(s; 0, \lambda^s I) \propto \exp \left\{ -\frac{1}{2} s^T \lambda^s Is \right\} = \prod_{i=1}^{n} \exp \left\{ -\frac{1}{2} \lambda^s s_i^2 \right\} \quad (6.1)$$

which is trivially a pairwise model (Section 2.3.1.1) with only node potentials

$$\phi^C_i(s_i) = \exp \left\{ -\frac{1}{2} \lambda^s s_i^2 \right\} \quad (6.2)$$
for all $i$, and no edges or edge potentials $\psi_{ij}^c$. This gives rise to the following graphical model (Figure 6.3):

![Graphical Model](image)

Figure 6.3: Graphical Model for the Gaussian iid source, which only has node potentials $\phi_i^c$.

### 6.1.2 Gauss-Markov Source

A Gauss-Markov source is a Gaussian Markov chain that is generated by the following process:

\[
\begin{align*}
  s_1 &\sim \mathcal{N}^{-1}(0, \lambda_0^s) \\
  s_i &= a s_{i-1} + w_i, \quad w_i \sim \mathcal{N}^{-1}(0, \lambda^s) \quad \text{for } i \in [2..n]
\end{align*}
\]

where $\lambda_0^s$ is the precision parameter for $s_1$ and $\lambda^s$ is the precision parameter for $w_i$. The superscript $s$ is to distinguish the parameters of the source model from the parameters of message passing, as we shall discuss in Section 6.2. The joint pdf can thus be expressed as

\[
\begin{align*}
  p_{s^n}(s^n) &= \prod_{i=1}^{n} p_{s_i}(s_i | s_1, \ldots, s_{i-1}) \\
  &= p_{s_1}(s_1) \cdot \prod_{i=2}^{n} p_{s_i|s_{i-1}}(s_i | s_{i-1}) \\
  \propto \exp \left\{ -\frac{1}{2} \lambda_0^s s_1^2 \right\} \cdot \prod_{i=2}^{n} \exp \left\{ -\frac{1}{2} \lambda^s (s_i - a s_{i-1})^2 \right\} \\
  &= \exp \left\{ -\frac{1}{2} \lambda_0^s s_1^2 \right\} \cdot \prod_{i=2}^{n} \exp \left\{ -\frac{1}{2} \lambda^s s_i^2 \right\} \cdot \prod_{i=2}^{n} \exp \left\{ -\frac{1}{2} \lambda^s a^2 s_{i-1}^2 \right\} \\
  &\quad \cdot \prod_{i=2}^{n} \exp \{ \lambda^s a s_i s_{i-1} \} \\
  &= \exp \left\{ -\frac{1}{2} \left( \lambda_0^s + a^2 \lambda^s \right) s_1^2 \right\} \cdot \prod_{i=2}^{n-1} \exp \left\{ -\frac{1}{2} \left( \lambda^s + a^2 \lambda^s \right) s_i^2 \right\} \\
  &\quad \cdot \exp \left\{ -\frac{1}{2} \lambda^s s_n^2 \right\} \cdot \prod_{i=2}^{n} \exp \{ a \lambda^s s_i s_{i-1} \} \\
  &= \prod_{i=1}^{n} \phi_i^c(s_i) \cdot \prod_{i=2}^{n} \psi_{i-1,i}^c(s_{i-1}, s_i)
\end{align*}
\]

74
which is pairwise model (Section 2.3.1.1) with node potentials \( \phi^c_i \) such that

\[
\phi^c_i(s_i) \propto \begin{cases} 
\exp \left\{- \frac{1}{2} \left( \lambda^s + a^2 \lambda^s \right) s_i^2 \right\} & i = 1 \\
\exp \left\{- \frac{1}{2} \left( \lambda^s + a^2 \lambda^s \right) s_i^2 \right\} & i \in \{2..n-1\} \\
\exp \left\{- \frac{1}{2} \lambda^s s_i^2 \right\} & i = n 
\end{cases}
\] (6.12)

and edge potentials \( \psi^c_{ij} \) for \( j = i - 1 \) where

\[
\psi^c_{i-1,i} \propto \exp \{ a \lambda^s s_i s_{i-1} \} \quad i \in \{2..n\}
\]

which gives rise to the following graphical model (Figure 6.14):

![Figure 6.14: Graphical Model for the Guass-Markov source, which has node potentials \( \phi^c_i \) and edge potentials \( \psi^c_{ij} \) in the structure of a Markov chain.](image)

### 6.2 Decoding Mechanics

As described in Algorithm 3.29 line 4, the message passing equations among the source nodes \( S \) are

\[
m_{S_i s_j}^{(\tau)}(s_j) := \int_{s_i} \left( \phi^c_i(s_i) \psi^c_{ij}(s_i, s_j) \prod_{k \in N_i^c \setminus \{j\}} m_{S_k s_i}^{(\tau-1)}(s_i) \prod_{a \in N_i^q} m_{S_a s_i}^{(\tau-1)}(s_i) \right) \]

(6.15)

for pairwise models. With the Gaussian approximation scheme presented in Section 5.2.1, the \( F^G \) to \( S \) messages \( m_{F^G S_i}^{(\tau-1)}(s_i) \) are approximated as a Gaussian pdf. Therefore, we can use the well known results in Gaussian message passing on the source graph.

#### 6.2.1 Gaussian Potentials

We note that the node potentials \( \phi^c_i \) and edge potentials \( \psi^c_{ij} \) are in the quadratic exponential form for both the iid model and the Gauss-Markov model. In general, this is true for all Gaussian graphical models regardless of the underlying correlation of the variables: all node potentials and edge potentials assume the quadratic exponential form. All Gaussian graphical models can have their node and edge potentials be
written as

\[
\phi^C_i(s_i) \propto \exp \left\{ -\frac{1}{2} \lambda^\phi_i s_i^2 + \eta^\phi_i s_i \right\} \\
\psi^C_{ij}(s_i, s_j) \propto \exp \left\{ -\lambda^\psi_{ij} s_i s_j \right\}
\]

for some model dependent \( \lambda^\phi_i, \eta^\phi_i, \) and \( \lambda^\psi_{ij} \) for all nodes \( s_i \) and all edges \((s_i, s_j)\), where the superscripts are used to distinguish the parameters of the node and edge potentials from the parameters of the messages, which we discuss in Section 6.2.2.

In the case of the iid source, we have

\[
\eta^\phi_i = 0 \\
\lambda^\phi_i = 1
\]

and for the Gauss-Markov source, we have

\[
\eta^\phi_i = 0 \\
\lambda^\phi_i = \begin{cases} 
\lambda_0^s + a^2 \lambda^s & i = 1 \\
\lambda^s + a^2 \lambda^s & i \in \{2..n - 1\} \\
\lambda^s & i = n
\end{cases} \\
\lambda^\psi_{ij} = -a \lambda^s
\]

### 6.2.2 Gaussian Message Passing

Figure 6.23: Belief propagation on the source subgraph \( \mathcal{C} \). The source messages \( m^{(r)}_{s_i s_j} \) (orange arrow) is computed as the integral over \( s_i \) of the product of the node potential (red arrow), neighboring edge potentials (blue arrows) except that from node \( s_j \), and incoming messages (green arrow).

With the potentials above, we consider the message passing equation in 6.15. We note that the integral in the expression is an marginalization operation of the joint distribution of \( s_i, s_j \) by integrating out \( s_i \). Parameterizing the Gaussian messages from \( \mathcal{F}^Q \) to \( \mathcal{S} \) in the information form

\[
m^{(r)}_{\mathcal{F}^Q \mathcal{S}^Q}(s_i) =: \mathcal{N}^{-1}(s_i; \eta^{(r)}_{\mathcal{F}^Q \mathcal{S}^Q}, \lambda^{(r)}_{\mathcal{F}^Q \mathcal{S}^Q})
\]
and applying Facts 2.10 and 2.15, we can thus derive the messages as

\[ m_{s_i,s_j}^{(\tau)}(s_j) \propto \mathcal{N}^{-1}(x_j; \eta_{s_i,s_j}^{(\tau)}, \lambda_{s_i,s_j}^{(\tau)}) \quad (6.25) \]

where

\[ \eta_{s_i,s_j}^{(\tau)} := -\chi_{s_i,j} \left( \sum_{k \in N_i^s \setminus \{j\}} \lambda_{s_k,s_i}^{(\tau-1)} + \sum_{a \in \mathcal{N}_i^0} \lambda_{s_a,s_i}^{(\tau-1)} \right)^{-1} \]  

\[ \lambda_{s_i,s_j}^{(\tau)} := -\chi_{s_i,j} \left( \sum_{k \in N_i^s \setminus \{j\}} \lambda_{s_k,s_i}^{(\tau-1)} + \sum_{a \in \mathcal{N}_i^0} \lambda_{s_a,s_i}^{(\tau-1)} \right)^{-1} \lambda_{s_i,j}^{(\tau)} \]  

For the iid model, the messages would be trivial, given the lack of edges in the source subgraph. For the Gauss-Markov model, the sums in Equations 6.26 and 6.27 will only consist of one term, given the Markov chain structure of the source subgraph.

### 6.2.2.1 Initialization

We initialize all messages to be trivial, i.e.

\[ m_{s_i,s_j}^{(0)}(s_j) \propto \mathcal{N}^{-1}(s_j; \eta_{s_i,s_j}^{(0)}, \lambda_{s_i,s_j}^{(0)}) \quad \text{for } (i,j) \in \mathcal{E}_s \]  

where

\[ \eta_{s_i,s_j}^{(0)} = 0 \]  

\[ \lambda_{s_i,s_j}^{(0)} = 1 \]  

### 6.2.3 Marginalization and Convergence

At the end of each iteration \( \tau \) of message passing, we estimate \( \hat{S}^{(\tau)} \) with the well known marginalization equation for graphical models, reproduced from Algorithm 3.29.21:

\[ \hat{p}_s^{(\tau)}(s_i) := \prod_{a \in \mathcal{N}_i^0} m_{s_a,s_i}^{(\tau)}(s_i) \prod_{j \in \mathcal{N}_i^c} m_{s_j,s_i}^{(\tau)}(s_i) \quad (6.31) \]

which, in terms of the \( S \) message parameters \( \eta_{s_i,s_i}^{(\tau)} \) and \( \lambda_{s_i,s_i}^{(\tau)} \) and the \( \mathcal{F}^0 \) to \( S \) message parameters \( \eta_{s_a,s_i}^{(\tau)} \) and \( \lambda_{s_a,s_i}^{(\tau)} \), we can express Equation 6.31 as

\[ \hat{p}_s^{(\tau)}(s_i) \propto \mathcal{N}^{-1}(s_i; \eta_i^{\text{mrg}}, \lambda_i^{\text{mrg}}) \]  

77
where

\[
\eta_{i}^{\text{mrg}} = \eta_{i}^{\phi} + \sum_{j \in N_{i}^{S}} \eta_{j}^{(\tau)} + \sum_{a \in N_{i}^{Q}} \eta_{j}^{(\tau)}
\]

\[
\lambda_{i}^{\text{mrg}} = \lambda_{i}^{\phi} + \sum_{j \in N_{i}^{S}} \lambda_{j}^{(\tau)} + \sum_{a \in N_{i}^{Q}} \lambda_{j}^{(\tau)}
\]

The minimum mean squared error (MMSE) estimate \(\hat{s}^{(\tau)}\) (which coincides with the maximum likelihood (ML) estimate) will thus be the mean of the marginal, ie.

\[
\hat{s}_{i}^{(\tau)} := (\lambda_{i}^{\text{mrg}})^{-1} \eta_{i}^{\text{mrg}}
\]

We use convergence of the estimate \(\hat{s}\) to determine when to terminate the algorithm. In particular, we terminate at iteration \(\tau^{*}\) and return the estimate \(\hat{s}^{(\tau^{*})}\) when the MMSE estimates \(\hat{s}\) are \(\varepsilon\)-close under the \(L_{\infty}\) norm, ie. satisfy

\[
\varepsilon > \|\hat{s}^{(\tau^{*})} - \hat{s}^{(\tau^{*}-1)}\|_{\infty} = \max_{i \in \{1..n\}} \left|\hat{s}_{i}^{(\tau^{*})} - \hat{s}_{i}^{(\tau^{*}-1)}\right|
\]

for some convergence criterion \(\varepsilon\), or for a maximum number of iterations \(\tau^{\max}\). For our experiments, we use \(\varepsilon = 0.01\) and \(\tau^{\max} = 100\).

### 6.3 Summary

The source model, representing the decoder’s prior knowledge of the source, can be expressed as an undirected Gaussian graphical model. With our architecture, the source model forms a modular part of the decoding algorithm not entangled with any other parts of the system, and can therefore afford flexibility that to our knowledge no existing systems provide.

Two of the advantages include (i) having a universal compressor for Gaussians whose structure does not radically change depending on the underlying correlation, and (ii) the ability to refine a model, hence improve the compression rate, after a source sequence has been compressed, when we have more information about the source to provide a better source model for the decoder, as we shall discuss in Section 9.2.

In addition, with the source model being represented as a graphical model, we can run inference tasks on top of the decoding algorithm, eg. to learn the parameters of the model, an extension we will briefly explore in Section 9.3.
Compression Performance

In this chapter, we present the experimental results of applying the our compression architecture to two classes of Gaussian sources we described in Chapter 6, namely the Gaussian iid source and the Gauss-Markov source.

We first describe our experimental setup in Section 7.1. Next, we present the compression rates and the corresponding distortions of our algorithm, and compare our performance with the theoretical lower limit of compression (i.e., the rate-distortion bound, Section 2.2.4) and the theoretical bounds of some classes of algorithms for compressing Gaussian iid and Gauss-Markov sources in Sections 7.2 and 7.3 respectively.

7.1 Experimental Setup

For each source model, we select a set of representative parameter values of the model, which we then use to generate samples through Gibbs sampling (Section 2.3.3). For each set of model parameters, we test the architecture on different distortion levels, which are implicitly controlled by the quantizer function \( q(\cdot) \) and the precision parameter \( b \), the number of bits used to represent the quantized values.

We first generate 10 different random LDPC codes \( h(\cdot) \) with an off-the-shelf LDPC generator, with uniform column weight \( \rho' = 3 \) and no four-cycles. With a well generated LDPC matrix, this implies a row weight of \( \rho = O(1) \). With each \( h(\cdot) \), we generate 10 source sequences \( s^n \). For these 100 sequences, we run the system with With \( q(\cdot) \) known, we calculate \( u^m = q(s^n) \), and we report the lowest rate

\[
    r^* := \min_{kb, q} \{ r_{\text{code}} + r_{\text{dope}} = \min_{kb, q} \frac{kb + |q|}{n} \} \quad (7.1)
\]

and its associated distortion

\[
    \text{SQNR}_{dB} = -10 \log (10 \cdot \Delta_{\text{MSE}}(\hat{s}^n, s^n)) \quad (7.2)
\]
for which the system converges within $\tau^* = 100$ iterations with $\hat{u}^m = u^m$.

In particular, for each distortion level $\delta$, we report two rates:

1. **PGM-Proto**: The rate $r^*_p(\delta)$ for which at least one source sequence $s^n$ converges for some code $h(\cdot)$ for our probabilistic graphical model based data compressor, and
2. **PGM-Convg**: The rate $r^*_h(\delta)$ for which all generated source sequences $s^n$ converges for some code $h(\cdot)$ for our probabilistic graphical model based data compressor.

These two values are of special interest because of their implication on convergence: the PGM-Proto rate $r^*_p(\delta)$ signifies the rate that can be achieved with a perfect code $h(\cdot)$, while the PGM-Convg rate $r^*_h(\delta)$ signifies a good code $h(\cdot)$ that our system can use, one that causes belief propagation to almost always converge regardless of the source sequence.

### 7.2 Gaussian i.i.d. Sources

We presented the model of the Gaussian i.i.d. source in detail in Section 6.1.1. Now, in this Section, we discuss its rate-distortion bound, the bounds for existing algorithms for Gaussian i.i.d. sources, and the performance of our architecture, with respect to the PGM-Proto rate $r^*_p(\delta)$ and the PGM-Convg rate $r^*_h(\delta)$.

#### 7.2.1 Rate-Distortion Bound

We first describe the rate-distortion bound (Definition 7.3), the theoretical lower bound on the rate-distortion trade-off that any compression system can achieve.

**Theorem 7.3. (Rate-Distortion Bound for Gaussian i.i.d. Sources)**. For high rates, the rate-distortion function of a Gaussian i.i.d. source with variance $\sigma^2_{\text{iid}}$ is

$$ R(\delta; s_{\text{iid}}) = -\frac{1}{2} \log_2 \left( \frac{\delta}{\sigma^2_{\text{iid}}} \right) $$

(7.4)

For our test samples, we have $\sigma^2_{\text{iid}} = 1$, so expressed in terms of $\text{SQNR}_{\text{dB}}$, the bound is about

$$ r^*_{\text{iid,RD}} \approx 0.166 \cdot \text{SQNR}_{\text{dB}} $$

(7.5)

#### 7.2.2 Known Lower Bounds for Classes of Algorithms

Here, we present two classes of algorithms known for compressing Gaussian i.i.d. sources, and the theoretical rate-distortion lower bounds achieved by each of them.
7.2.2.1 Lloyd-Max Algorithm

The Lloyd-Max Algorithm is an iterative algorithm that is similar to k-means clustering. With $\beta$ quantization levels, it starts by randomly choosing $\beta$ representative values, and set threshold values to be the equidistant point of each pair of representative values. Then within each quantization region, it calculates its centroid as the new representative value. The algorithm repeats the two steps until convergence.

For a Gaussian iid source, Lloyd-Max is run on every one of the $n$ entries of the source sequence. For rate-distortion comparison, we give its lower bound below.

**Theorem 7.6. (Lloyd-Max on Gaussian iid Sources).** For a Gaussian iid source with variance $\sigma_{\text{iid}}^2$, the Lloyd-Max Algorithm achieves a rate-distortion bound of

$$R_{\text{LM}}(\delta; s_{\text{iid}}) = -\frac{1}{2} \log_2 \left( \frac{\delta}{\sigma_{\text{iid}}^2} \right) + \frac{1}{2} \log_2 \left( \frac{\sqrt{3\pi}}{2} \right)$$

(7.7)

Expressed in terms of SQNR$_{\text{dB}}$ for our samples, the Lloyd-Max bound is about

$$r_{\text{iid,LM}}^+ \approx 0.166 \cdot \text{SQNR}_\text{dB} + 0.722$$

(7.8)

We note that Lloyd-Max does not have a model free encoder, i.e. the algorithm needs to know the model at encode time to be able to compute the quantized values.

7.2.2.2 Entropy-Coded Uniform Scalar Quantizer

The entropy-coded uniform scalar quantizer (ECUS) is an algorithm that uses entropy coding, usually done with a trellis, on the output symbols of a uniform scalar quantizer, which we have described in Section 5.1.1. As proven in Gish and Pierce [10], the performance achieved by the entropy-coded uniform scalar quantizer is the optimum. Here, we present its lower bound:

**Theorem 7.9. (ECUS Quantizer on Gaussian iid Sources).** For a Gaussian iid source with variance $\sigma_{\text{iid}}^2$, the entropy-coded uniform scalar quantizer achieves a rate-distortion bound of

$$R_{\text{ECUS}}(\delta; s_{\text{iid}}) = -\frac{1}{2} \log_2 \left( \frac{\delta}{\sigma_{\text{iid}}^2} \right) + \frac{1}{2} \log_2 \left( \frac{\pi e}{6} \right)$$

(7.10)

Expressed in terms of SQNR$_{\text{dB}}$ for our samples, the entropy-coded uniform scalar quantizer bound is about

$$r_{\text{iid,ECUS}}^+ \approx 0.166 \cdot \text{SQNR}_\text{dB} + 0.255$$

(7.11)

We note that with the quantization matrix $Q = I_n$, our algorithm can be treated as an ECUS quantizer scheme, since the quantizer function $q(\cdot)$ with $Q = I$ is a uniform scalar quantizer and the code $h(\cdot)$ is an entropy coder in nature.
7.2.3 Results

Here, we plot the three bounds mentioned above (rate distortion RD, Lloyd-Max LM, and entropy-coded uniform scalar quantizer ECUS) with the performance of our architecture, plotted as the PGM-Proto rate and PGM-Convg rate.

![Graph showing performance of algorithms]

Figure 7.12: Performance of our algorithm on Gaussian iid sources against the rate distortion bound, Lloyd-Max, and EPCM. The y-axis is the rate and the x-axis is the SQNR.

Note that for these plots, we use $Q = wI$ for some constant $w$, hence the performance is lower-bounded by ECUS, which our architecture closely follows. We also consistently beat Lloyd-Max, at the same time not using source information.

7.3 Gauss-Markov Sources

We presented the model of the Gauss-Markov source in detail in Section 6.1.2. We recall that a Gauss-Markov source has 3 parameters: (i) $\lambda_0 = \frac{1}{\sigma_0^2}$ which is the inverse of the variance of the first element in the source sequence, (ii) $a$ which is the auto-correlation factor, and (iii) $\lambda = \frac{1}{\sigma^2}$ which is the inverse of the variance of the additive noise, also known as the innovation. For our experiments, we choose $\lambda_0 = 1$, and $a = 0.7$, and $\lambda = \frac{1}{0.51}$. This set of parameters has the property that the marginal distribution of each element of the source is $\sim N(0, 1)$, a property we will discuss later.

In this Section, we discuss its rate-distortion bound, the bounds for existing algorithms for Gauss-Markov sources, and the performance of our architecture, with respect to the PGM-Proto rate $r^*_a(\delta)$ and the PGM-Convg rate $r^*_b(\delta)$.  

82
7.3.1 Rate-Distortion Bound

We first describe the rate-distortion bound (Definition ??) of the Gauss-Markov source [3].

Theorem 7.13. (Rate-Distortion Bound for Gauss-Markov Sources). For high rates, the rate-distortion function of a Gaussian-Markov source with parameters $a$ and $\lambda$ is

$$ R(\delta; s_{GM(a,\lambda)}) = -\frac{1}{2} \log_2 \left( \frac{\delta \lambda}{1 - a^2} \right) \quad (7.14) $$

For our test samples, we have $a = 0.7$ and $\lambda = \frac{1}{0.51}$, so expressed in terms of $\text{SQNR}_{dB}$, the bound is about

$$ r^*_{GM(a,\lambda),RD} \approx 0.166 \cdot \text{SQNR}_{dB} - \frac{1}{2} \log_2 \left( \frac{\lambda}{1 - a^2} \right) \quad (7.15) $$

$$ \approx 0.166 \cdot \text{SQNR}_{dB} - 0.971 \quad (7.16) $$

7.3.2 Known Lower Bounds for Classes of Algorithms

Here, we present two classes of algorithms known for compressing Gauss-Markov sources, and the theoretical rate-distortion lower bounds achieved by each of them.

7.3.2.1 Entropy-Coded Uniform Scalar Quantizer

We described in Section 7.2.2.2 the ECUS quantizer. While the algorithm is the same, the rate-distortion bound achieved by ECUS for a Gauss-Markov source is slightly different.

Theorem 7.17. (ECUS Quantizer on Gauss-Markov Sources). For a Gauss-Markov source with parameters $a$ and $\lambda$, the entropy-coded uniform scalar quantizer achieves a rate-distortion bound of

$$ R_{\text{ECUS}}(\delta; s_{GM(a,\lambda)}) = -\frac{1}{2} \log_2 (\delta \lambda) + \frac{1}{2} \log_2 \left( \frac{\pi e}{6} \right) \quad (7.18) $$

Expressed in terms of $\text{SQNR}_{dB}$ for our samples, the entropy-coded uniform scalar quantizer bound is about

$$ r^*_{GM(a,\lambda),\text{ECUS}} \approx 0.166 \cdot \text{SQNR}_{dB} - \frac{1}{2} \log_2 (\lambda) + 0.255 \quad (7.19) $$

$$ \approx 0.166 \cdot \text{SQNR}_{dB} - 0.231 \quad (7.20) $$

since $a = 0.7$ and $\lambda = \frac{1}{0.51}$.

We note that as with 7.2.2.2, our algorithm can be treated as an ECUS quantizer scheme when $Q = I$. 

83
7.3.2.2 Differential Pulse Code Modulation

The Differential Pulse Code Modulation (DPCM) algorithm is specifically designed to quantize a Markov Chain. In particular, it quantizes the innovation, the additive Gaussian white noise $w_i$, described in Equation 6.5.

The DPCM reconstruction thus simulates the Markov generation process, adding the representative value of the quantized innovation $\hat{w}_i$ to $a s_{i-1}$ as the reconstruction for $s_i$. Here, we present its lower bound:

**Theorem 7.21. (DPCM Quantizer on Gauss-Markov Sources).** For a Gauss-Markov source with parameters $a$ and $\lambda$, the differential pulse code modulation quantizer achieves a rate-distortion bound of

$$R_{\text{DPCM}}(\delta; s_{\text{GM}(a,\lambda)}) = -\frac{1}{2} \log_2 \left( \frac{\delta \lambda}{1 - a^2} \right) + \frac{1}{2} \log_2 \left( \frac{\pi e}{6} \right) \quad (7.22)$$

Expressed in terms of SQNR$|dB$ for our samples, the entropy-coded uniform scalar quantizer bound is about

$$r^*_{\text{GM}(a,\lambda),\text{DPCM}} \approx 0.166 \cdot \text{SQNR}|dB - \frac{1}{2} \log_2 \left( \frac{\lambda}{1 - a^2} \right) + 0.255 \quad (7.23)$$

$$\approx 0.166 \cdot \text{SQNR}|dB - 0.716 \quad (7.24)$$

since $a = 0.7$ and $\lambda = \frac{1}{0.51}$.

We note that DPCM's encoder is not model free. In fact, in addition to needing to know that the source has a Markov structure, it also needs to know the specific parameters $a$ and $\lambda$ to be able to produce the quantized values.

7.3.3 Results

Here, we plot the three bounds mentioned above (rate distortion $R_D$, entropy-coded uniform scalar quantizer $ECUS$, and differential pulse code modulation $DPCM$) with the performance of our architecture, plotted as the PGM-Proto rate and PGM-Convg rate.

Note that for these plots, we use $Q = wI$ for some constant $w$, hence the performance is lower-bounded by ECUS, which our architecture closely follows. We are, however, unable to beat DPCM, which achieves a performance very close to the rate-distortion bound. We note that despite the worse performance, our model-code separation architecture affords the flexibility that DPCM cannot, for at compression time the DPCM encoder needs to know everything about the source to be able to compress, whereas our architecture requires no such information.

7.4 Summary

In this chapter, we presented the performance of our algorithm and compared it with the lower bounds of classes of algorithms, in addition to its theoretical lower bound, the rate-distortion bound. Specifically, we compressed Gaussian iid sources...
Figure 7.25: Performance of our algorithm on Gauss-Markov models against the rate distortion bound, ECUS, and DPCM. The y-axis is the rate and the x-axis is the SQNR.

and Gauss-Markov sources, noting that some other algorithms which have a better performance have access to the source model at encoding time, information that we do not have and an entanglement we try to avoid.

In the remaining chapters, we will discuss the optimization decisions made to the system, as well as potential extensions to the architecture.
Discussion and Analysis

In Chapters 3, we described the general architecture of our compression algorithm, defining the four main components of (i) source model, (ii) quantizer $q(\cdot)$, (iii) translator $t(\cdot)$, and (iv) code $h(\cdot)$. In Chapters 4, 5, and 6, we presented various possible choices of each of the four components. Then, in Chapter 7, we discussed the performance of our algorithm on compressing the Gaussian iid source and the Gauss-Markov source. In this chapter, we will now compare how different choices of the architectural components affect compression performance.

8.1 LDPC Code Optimization

The code $h(\cdot)$ is the main component responsible for the compression, and hence its parameters controls the compression rate. To optimize for the compression rate, therefore, we naturally focus on the code component first.

8.1.1 Four Cycles

As introduced in Section 4.1.2, it is very likely that a randomly generated LDPC code contains four cycles. Here, we explore the effects of removing four cycles from the code.

We experiment with compressing Gaussian iid sources with sequence length $n = 500$. In Figure ??, we plot the performance of removing four-cycles with keeping four-cycles at different compression rates. As can be seen in the plot, the two sets of points closely trace each other. Removing four-cycles in the LDPC code does not seem to have an effect on the compression performance.

8.1.2 Column Weights

The degree distribution is an important parameter of an LDPC code. Here, we experiment with different fixed column weights $\rho$. 
As we can see in Table ???, increasing the column weight from $\rho' = 3$ decreases performance in terms of compression rate, while the number of iterations needed for convergence remains similar. However, increasing $\rho'$ increases the time complexity for decoding. Intuitively, with more dependencies between the translated bits $z$ and the hashed bits $x$, more information is needed for decoding.

On the other hand, decreasing the column weight to $\rho' = 2$ also decreases performance in terms of compression rate. Intuitively, with each translated bits $z_i$ connected to only two hashed bits $x_a$, it becomes harder to reconstruct the $z$ bits. This is evident in Table ???, in which more iterations are needed for convergence.

8.2 Doping Optimization

Doping is an operation that has to be designed in conjunction with the translator. For this section, we will focus on Gray code as the translator.

8.2.1 Random Doping and Lattice Doping

Recall that random doping (Section 4.3.1) chooses the set $D$ of dope bits from $Z$ randomly, and that lattice doping (Section 4.3.3) dopes every $b^\text{th}$ bit of $Z$. Therefore, random doping is translator agnostic and lattice doping is translator aware. As discussed in Section 4.3, lattice doping should, intuitively, outperform random doping since it uses knowledge of the probability distribution of $Z$ induced by the translator. Here, we present quantitatively the gains from using lattice doping over random doping.

We compressed Gaussian iid sources with length $n = 500$. As seen in Figure ??, the performance of random doping (with the same dope rate as lattice doping when $\frac{1}{b}$ for $b \geq 2$) is consistently around 0.3 bits worse than lattice doping for a fixed distortion.

We next compressed Gauss-Markov sources with length $n = 500$. For $b \geq 2$, we note that the total rate for random doping is about 0.2 bits worse than that of the optimal lattice doping for a fixed distortion, a result consistent with that of the iid sequences.

8.2.2 Random Doping and Sample Doping

Sample doping, as introduced in Section 4.3.2, chooses the dope bits in groups of $b$, where $b$ is the output size of each translator. We now present the results of using sample doping.

Noting that sample doping is meaningful only when $b \geq 2$, we found that for both Gaussian iid and Gauss-Markov sources, sample doping performs around 0.3 bits worse than optimized lattice doping for any fixed distortion. The results can be seen in Figure ???. This can be explained by the asymmetry of information within the bits induced by the translator. Under Gray coding, doping the second most significant bit wastes the dope bit because the second bit always indicates whether
the sample is within certain bins of the middle, which for a Gaussian distribution is disproportionately true. Therefore, when we dope samples, some bits within each sample are not as useful as they can be, and thus sample doping performs worse than lattice doping.

This phenomenon calls for a search for better translator schemes, ideally, one that induces a $\text{Bern} \left( \frac{1}{2} \right)$ distribution on the bits.

### 8.2.3 Multiple Lattice Doping

In Section 4.3.4, we described potential doping schemes that extend the advantages of lattice doping. Here, we present the results of multiple lattice doping. Multiple lattice doping, refers to doping bits of the same significance for each $b$-bit group. For the Gaussian iid source, we see that multiple lattice doping gives the best results for $b \geq 3$, the regime for which multiple lattice doping applies. Figure ?? compares multiple lattice doping with normal lattice doping, showing that it outperforms normal lattice doping significantly.

For the Gauss-Markov source, multiple lattice doping provides less gain than that of the Gaussian iid source. As seen in Figure ??, the only significant gain happens in higher bit regimes (eg. $b = 5$). The difference in performance of multiple lattice doping between the iid source and the Markov source can be explained by the correlation structure: in the iid source each $b$-bit group is uncorrelated to one another, hence each group needs some initialization for better performance. For the Markov source, however, doping within each $b$-bit group provides redundant information, for the decoder can already infer from the Markov structure a sample’s localization given its immediate neighbors.

### 8.2.4 Random Sample Lattice Doping

### 8.2.5 Hashing Doped Bits

Garcia-Frias and Zhong suggested in their work [8], which uses LDPC codes to compress binary Markov chains, that not hashing the doped bits will increase the rate performance for lossless compression. With our architecture, however, this idea does not seem to contribute any significant improvement over hashing the doped bits. In addition, in the case of random doping, it can even worsen the performance, since there is no anchor with which the algorithm can start message passing in the code graph. In the case of lattice doping, the negative effects are not as significant, for the non-doped initial messages are inherently biased, given our uniform scalar quantization of Gaussians.

### 8.2.6 Zero Doping

As mentioned in Section 3.4, doping provides an anchor for initialization for message passing. We now explore whether doping is necessary and its impact on performance.
We compressed Gaussian iid sources with length $n = 500$. From Figure ??, we see that the performance of zero doping for $b \geq 2$ is consistently around 0.1 bits worse than lattice doping. Next, we compressed Gauss-Markov sources with length $n = 500$ with zero doping. For $b \geq 2$, the total rate for zero doping is consistently 0.2 worse than that of optimized lattice doping.

Intuitively, while we do not explicitly dope any of the bits, the inherent asymmetry of the distribution induced by the translator already implicitly provides a starting point for message passing, although the performance is not as good as explicit doping.

### 8.2.7 Doping in the Low Bit Regime

It should be noted that the low bit regime ($b = 1$ and $m \leq n$) is a special case for our system. Many of the translator properties do not apply for when the translator only has a 1-bit output. Hence, for $b = 1$, the dope bits need to be carefully chosen, since lattice doping, which works well for higher bit regimes $b \geq 2$ by exploiting the inherent asymmetry of the translated bits, does not have a logical parallel. Other methods, such as zero doping and random doping, do not seem to work well, as we shall discuss.

For the Gaussian iid source, our experiments show that there is no difference in performance between random doping and lattice doping. This is because of the lack of inherent asymmetry of the translated bits, which lattice doping exploits. We observe that compression rate is decreasing with increasing dope rate, with the worse performance being $r_{\text{dope}} = 0$ (with $r^*_p = 1.23$) and the best performance being $r_{\text{dope}} = 1$, which trivially achieves a rate of $r = 1$. This result is source-dependent: with an iid source, there is no correlation that the entropy code $h(\cdot)$ can take advantage of, since the translated sequence is distributed according to $\text{Bern}(\frac{1}{2})$, which has an entropy of 1. Thus, no entropy coding is possible, and the best that can be done is to transmit the original sequence verbatim.

However, for Gauss-Markov sources for $b = 1$, the dope rate $r_{\text{dope}}$ can be tuned to achieve non-trivial results. As seen in Figure ??, zero doping achieves a total rate of $r^*_p = 1.14$, while the rate decreases to around $r^*_p = 0.82$ for $r_{\text{dope}} = 0.4$, before increasing to $r^*_p = 1$ trivially for $r_{\text{dope}} = 1$. The optimized random doping achieves the same rate of $r^*_p = 0.82$ for lattice doping. This shows that when there is correlation structure, entropy coding is possible, and random doping can perform well.

Our preliminary results seems to suggest that random doping would work well with the low bit regime. However, more research needs to be devoted to exploring the sub 1-bit regimes for our compression architecture.

### 8.3 Translator Optimization

The next component to be tuned would be the translator. As mentioned, we have two candidates: standard binary code and Gray code. We now experiment with the two translators and compare their performance.
Table ?? presents the total compression rate of Gray code vs. standard binary code, with different bits doped under lattice doping (Section 4.3). As we can see, the choice of translator, as well as the choice of the doped bit with lattice doping, affects the rate performance significantly. This points to the inherent asymmetrical importance of the bits within each sample, a concept that should be further explored. In particular, the design of the translator needs to be coordinated with the dope bits.

One intuitive explanation that gray code provides a better performance than the standard binary code is the standard binary code's frequent change of bit values between adjacent coded values, a problem that Gray code avoids by enforcing only 1-bit Hamming distance between adjacent coded values. In particular, consider the case when standard binary code is used, and that the decoder is almost certain that the least significant bit is 1. This information does not localize the search space, for the resulting distribution eliminates every other possible quantized sample bin value, hence it is does not make finding the actual bin too much easier. Gray code, on the other hand, provides a higher degree of localization, which allows it to outperform standard binary code by a total rate of around 0.03b.

8.4 Quantizer Optimization

The quantizer component is perhaps the component with the most choices for optimization. Here, we compare two choices for the quantization matrix $Q$: (i) $Q = I$, and (ii) $Q_{ai} \sim \text{Bern}(p) \cdot N(0, 1)$ for some small $p$, where $Q$ is sparse and the non-0 entries are drawn from the standard normal distribution.

From our experiments, the identity matrix outperforms all trials of a sparse matrix with entries $\sim N(0, 1)$. While this is a disappointing result, it suggests that much more work can be done in the optimization of the quantizer component $q(\cdot)$ in the exploration of the low bit (sub 1-bit) regime.

8.5 Summary

In this chapter, we expanded on the results in Chapter 7 and analyzed the components of the compression architecture. We justify our design choices by experimenting with different options of each component and explaining intuitively why certain choices will give better rate performance. In particular, we examined the choice of code, doping, translator, and quantizer. Our experiments with the quantization portion reveals that much more work can be done in choosing a good $Q$ matrix.
In this chapter, we present four practical compression problems that can be handled elegantly given the modularity and flexibility of our system. In Section 9.1, we discuss the impact of loss or corruption of the compressed data. Next, in Section 9.2, we experiment with decoding with an incorrect data model. Then, in Section 9.3, we present an augmented source model which can learn the parameters of the data source at decompression time. Finally, in Section 9.4, we present existing schemes of encrypted compression and propose a new encrypted compression structure that allows for homomorphic encryption.

9.1 Robustness to Data Corruption

One important consequence of using an LDPC code to compute the hash $x^{kb}$ is its robustness to data corruption. In this section, we experiment with bit erasure and bit flip of the compressed hash $x^{kb}$, and observe the system's performance in these scenarios.

9.1.1 Bit Erasure

Bit erasure is the scenario in which certain bits of the compressed bit stream $x^{kb}$ are known by the decoder to be inaccurate, which the decoder will disregard. It is obvious that if enough bits are erased such that the number of remaining bits are below the decoding rate-distortion threshold, decoding is impossible. Therefore, to account for potential bit erasure, we should compress with redundancy, i.e. with more bits than necessary for decoding.

To simulate bit erasure, we experimented with randomly choosing a subset $\mathcal{E}$ of the compressed bits to be erased. Table ?? summarizes our experiment results. We note that for all instances, the effective Proto rate, defined to be

$$r_{\text{effective}} := \frac{|D| + kb - |E|}{n}$$

(9.1)
remains relatively constant when $\mathcal{E}$ is small, but when $\mathcal{E}$ is large, the effective rate increases due to the

With a higher erasure ratio, however, we start noticing that some of the translated bits $z$ were decoded incorrectly because all the hashed bits $x$ connected to it were erased. The rest of the $z$ bits, however, are still decoded correctly, thereby causing a distortion that is barely higher than complete correct decoding (ie. with a slightly lower $\text{SQNR}_{IB}$).

By the same explanation, we note that the effective Conv rate is sensitive to the erasures, as it becomes harder and harder to decode completely correctly with more and more bit erasures.

### 9.1.2 Bit Flip

Bit flip is the scenario in which certain bits of the compressed bit stream $x^{kb}$ are flipped, unbeknown to the decoder. In our experiment, we randomly flip each compressed bit with probability $p_{\text{flip}}$, with the set of erased hashes denoted as $\mathcal{F}$. Table ?? summarizes our experiment results, recording the Proto rate. We note that the Conv rate cannot be reported, for we cannot find a code that always correctly decodes regardless of the positions of the bit flips.

We observe that for bit flips are much more destructive than bit erasures for our system. For low bit flip ratios, our algorithm can still largely recover from the corruption with more redundant bits, but for bit flip ratios higher than 0.02, the algorithm simply cannot converge, for there are many conflicting messages that causes the algorithm to loop between different nonsensical states.

### 9.1.3 Discussion

As we presented above, our system can elegantly handle data corruption with no additional architectural change. Using an LDPC code to compute the hash, our algorithm is robust towards bit erasures, with the effective rate essentially constant even with considerable erasures. On the other hand, LDPC compression is less robust towards bit flips, as the corrupted bits contain erroneous information that tend to propagate throughout the system, causing confusion. However, with a low bit flip ratio, data can still be reconstructed to a large extent, if not perfectly.

The architecture’s robustness to data corruption is remarkable. As Heydegger’s 2009 paper [12] analyzed the robustness of image compression algorithms, a single bit flip can have catastrophic implications for most image compressors, especially JPEG and PNG. With ZIP files, even a single bit erasure, let alone a bit flip, will render complete loss of the data.

While one of the major challenges of digital communication is combating channel noise, it is truly surprising that very little attention has been paid to this subject in its dual problem of data compression. As Heydeger noted, a file’s robustness to data corruption was not given the attention it deserves in the design of general file formats, or even in the design of compression algorithms, where data corruption has a much more catastrophic consequence. Heydeger’s work was the first, and remains one of
the very few, comprehensive survey of this topic. We believe that data robustness should be an important criterion in designing a compression system, and the lack of research in this fundamental subject is truly alarming.

9.2 Source Model Mismatch

As described in Section 3.5.1, the Encode algorithm does not need to know the source model, while the Decode algorithm uses its knowledge of the source to recover the MSE estimate $\hat{s}^n$ of the source sequence $s^n$.

We now consider the effects of decoding with a model that does not correspond to the true source model. Such a scenario is of practical interest, for often times the true model is unknown or is not available, and we have to resort to estimating or learning the source structure and parameters, which may contain less structure than the true model.

9.2.1 Imprecise Model

In Section 3.5.1, in the discussion of rate selection, we mentioned as part of the data storage scenario that the encoder may sometimes want encode with a rate much higher than necessary for reconstruction, and if it is discovered later that the decoder can decode at a lower rate, the compressed sequence $x^{kb}$ can be truncated without losing any information.

Consider the starting scenario, in which the decoder applies an imprecise source model for decoding, assuming that the rate $r$ is high enough for decoding both the correct source model and the imprecise one. In particular, we consider the example of attempting to decode a Gauss-Markov source as an iid source. Noting that the marginal distribution of a Gauss-Markov source (with appropriate parameters $\lambda^s_0$, $\lambda^s$, and $a$ such that $a^2 + \lambda^s = \lambda^s_0$) is an iid source, the iid model when used in this case is not incorrect, but imprecise, since it does not make use of the correlation of the source $s^n$.

We present the results of our simulation in Table ???. We note that even with the imprecise model, we can still decode and have a reconstruction $\hat{s}^n$ that has expected distortion fitting the bounds of that of an iid source. If later we realize that the source can be better modeled as a Gauss-Markov model, we do not need to change the compressed sequence nor the compression architecture itself, but simply replacing the source subgraph in the Decode algorithm will give us the expected performance. The immense flexibility of this architecture is extremely desirable, for it solves the backward compatibility issue that most existing data compression algorithms face. To our knowledge, there does not exist an algorithm that has this flexible decoding property that allows the source model to be refined after the source sequence has been compressed.

The rate-distortion gain of decoding with a more fitting model comes from both the quantization subgraph and the code subgraph. The distortion, controlled by the
quantizer subgraph, is decreased with a better model, in addition to that the rate, controlled by code subgraph, can be lowered with more correlation in the source.

9.2.2 Wrong Model

Next, we explore the effects of decoding with a wrong model. In particular, we experimented with decoding a Gaussian iid source with a Gauss-Markov model. We note that unlike the previous scenario presented in Section 9.2.1, the model at the decoder side is not an imprecise model, rather, one that assumes too much structure.

Table ?? summarizes the results of our experiment. Our experiments show that for the configuration we tried, when the decoder assumes a model with structure non-existent in the actual source, the algorithm does not converge at moderate rates. We started observing a few instances of 0 hashing errors (ie. finding the correct bins $u$) with higher rates (about 0.6 bit higher than the rate of decoding with the correct iid model). Even though the algorithm manages to find the right bin, the messages, however, are not converging. Rather, they flicker around before the algorithm reached its maximum allowed iteration count $\tau^* = 100$, explaining why the $\text{SQNR}_{dB}$ is lower than both of the correct model and the assumed model. In addition to decoding at higher rates, the non-convergence also lead to increase in computation complexity.

This experiment shows that while decoding with too much incorrectly assumed structure has rate and computation consequences, it is not completely destructive. The hashed stream can still be decoded, albeit at higher rates, with a higher distortion. In fact, since we still have the access to the compressed bit stream, if we notice such behavior, we can attempt decoding with a model of less structure as a remedy.

9.2.3 Discussion

The above two experiments suggest that it is safe for the decoder to assume less structure at the cost of a worse compression rate, but assuming structure that are not present in the actual source is catastrophic. Hence, with enough compressed bits, the most conservative approach to decoding would be to use the iid model, which assumes the least structure.

This also confirms the intuition that with more structure hence correlation in the source, the more we can compress. Practically, this suggests that the compression rate of the iid model is the maximum that we would need, meaning that it is always safe to compress to the iid rate in the lack of any information about the source, as we can always decode at the iid rate, at the cost of some loss of rate performance. As we know more about the structure of source, we can refine the model, and we will start seeing gains in both rate and distortion. With more correctly identified structure, we can also start truncating from the compressed bit stream as we can decode at lower rates.
9.3 Source Parameter Learning

In Section 9.2, we observed that decoding is still possible when the model is inaccurate. In this section, we explore the potential of when the model has known structure but unknown parameters. Consider the case when we are compressing a Gaussian iid source with

\[ s^n \sim N^{-1}(0, \theta I) \]  \hspace{1cm} (9.2)

with the scalar parameter \( \theta \) uncertain.

While our encoder is model-free, our decoder needs the model parameters for successful decoding. Without an exact parameter, it may seem that decoding is impossible. With the decoder being a probabilistic graphical model, however, it is natural to use the inference capabilities inherent to graphical models to infer the model parameters, as Huang noted [14]. Therefore, we can incorporate this lack of knowledge into our model by augmenting the source subgraph, noting that the joint distribution can be factored as

\[ p_{s^n, \Theta}(s^n, \theta) = p_{\Theta}(\theta) \prod_{i=1}^{n} p_{s_i | \Theta}(s_i | \theta) \]  \hspace{1cm} (9.3)

with

\[ p_{s_i | \Theta}(s_i | \theta) = N^{-1}(s_i; 0, \theta) \]  \hspace{1cm} (9.4)

which corresponds to the source graph illustrated by Figure 9.5.

![Figure 9.5: Augmented source model to handle source models with known structures but unknown parameters.](image)

As illustrated in the diagram, the source variables \( s_i \) are connected to the parameter \( \Theta \). The messages passed between the source nodes \( S_i \) and the parameter nodes will thus be the estimated probability density function of \( \theta \). With \( \Theta \) being a continuous variable, it is impossible to keep track of its pdf precisely. Therefore, we can model it as a Gaussian variable and pass its mean and variance as messages between nodes, with Gaussian updates after each iteration.

With this augmented source model, we can further learn the true parameter \( \theta \) over time, as the same decoder processes more and more data of the same model, refining the source model over time. This ability to learn the model, combining the concept of machine learning into data compression, is immensely powerful.
9.4 Encrypted Compression

Encrypted compression refers to compressing encrypted data without decrypting the data first. For traditional model-specific compression algorithms, such a task is impossible, for the source model is necessary at compression time. However, with a source-agnostic encoder, our architecture can handle such a task with some minor modifications.

9.4.1 One-Time Pad Encryption


A one-time pad secret key $k^n$ is a binary bit stream of length $n$ that is randomly generated according to $\text{Bern}(\frac{1}{2})^n$. The pad length must be the same length as the plaintext $s^n$, and the encrypted ciphertext $g^n$ is taken to be

$$g^n = s^n \oplus k^n \tag{9.6}$$

where $\oplus$ denote bitwise addition over $\mathbb{Z}_2$, i.e. the exclusive-or (xor) operation. As a notation, we use bar (eg. $\overline{s}$) to denote an encrypted version of a variable.

As Shannon has proved in his 1949 paper [19], a one-time pad achieves perfect secrecy in the sense that decryption is impossible without knowledge of the secret key $k^n$. However, as its name suggests, a one-time pad $k^n$ can only be used once. If the same one-time pad is reused in part or in whole, the secrecy guarantees of the scheme breaks down, and information about the plaintext $s^n$ can be recovered in part or in whole by an eavesdropper of the ciphertexts who has no a priori knowledge of $k^n$ [1].

Therefore, to build the crypto-compression system around a one-time pad, there needs to be two communication channels: a public one that transmits the compressed bit stream, and a secure one that transmits the one-time pad which has the same length as the plaintext source. Given the non-reusable nature of the one-time pad, with the transmission cost of the pad we could have simply transmitted the unencrypted plaintext over the secure channel.

![Figure 9.7: The block diagram showing information flow in the encrypted compression architecture proposed by Johnson et al. Note that there are two channels: the public one for the compressed ciphertext, and the private one for the one-time pad. An eavesdropper who only has access to the compressed ciphertext will not be able to decompress and decrypt the data.](image-url)
It may seem tempting to replace the random one-time pad with a pseudo-random one, with the encrypter communicating only the random seed over the secure channel to the decoder. This transforms the one-time pad into a stream cipher, which has a much weaker security guarantee. In particular, with the random seed being much smaller than the pad itself, stream cipher is susceptible to brute-force attacks, attacks that are ineffective on truly random one-time pads. Moreover, as with one-time pads, the secret key (i.e., the random seed) of the stream cipher cannot be reused. Hence, the amount of data that the system can compress is fundamentally limited by the size of the stream cipher secret key.

![Decoding graphs](image)

Figure 9.8: The decoding graphs for our system modified for quantized sources encrypted with a discrete one-time pad (left) and for sources encrypted with an additive Gaussian one-time pad (right). Note that the secret key $k$ enters the system through factor nodes (purple) corresponding to how it encrypts the data.

We note that our system can easily be modified to accommodate the compression of data encrypted with a one-time pad, given the source-agnostic nature of our encoder. For the sake of simplicity, we let our quantization matrix $Q$ be the identity matrix $I$. At compress time, the compressor is given a quantized source sequence encrypted by an additive one-time pad drawn uniformly from $\mathbb{Z}_{2^b}$ (which is equivalent to xor'ing the translated bits $z_m^b$ with a one-time pad drawn uniformly from $\mathbb{Z}_{2^b}$). At decompress time, the decoder runs BP on the joint graph illustrated in Figure 9.8 (left), which takes both $x_k^b$ and $k_m$ as inputs. The one-time pad enters the graph through the additional factor nodes, highlighted in purple, which adjusts for the shift caused by the one-time pad before passing source model information from the source and quantizer subgraphs to the rest of the graph.

Another technique suggested by Johnson et al. in [15] for continuous sources is to encrypt the original sequence with additive Gaussian noise, which are then quantized...
and translated into a binary sequence by a trellis code. We note that while Johnson et al. used a trellis code for compression, the idea of encryption with additive Gaussian noise can also easily be incorporated to our LDPC based system, as shown in Figure 9.8 (right). However, the issue of transmitting the secret key remains, and for a key that is itself Gaussian, quantization needs to be done on the key before transmission over the secure channel (Figure 9.7), another challenge that remains to be resolved.

### 9.4.2 Potentials for Homomorphic Encryption

Compressing encrypted data and joint decryption-decompression is of much theoretical interest. On the other hand, the ability to compute on compressed data addresses a very practical concern. Given that our compression architecture is based on linear codes, the linearity of matrix multiplication transfers over as an inherent property of our system as well. To keep the whole system linear, however, our translator of choice \( t(\cdot) \) needs to maintain this property as well. One such translator is the two's complement translator (Section 4.2.1), which preserves addition. With enough bits, the integer overflow issue of the two's complement representation can be resolved.

Let \( s_1 \) and \( s_2 \) be two source sequences generated by two source models \( \sim \mathcal{N}(\mu_1, \Sigma_1) \) and \( \sim \mathcal{N}(\mu_2, \Sigma_2) \) respectively, and that they are compressed by our architecture with the same set of parameters, ie. same \( q(\cdot), t(\cdot), \) and \( h(\cdot) \). If we only have access to compressed bit streams \( x_1 \) and \( x_2 \), we note that with an appropriate choice of translator, we have

\[
x_1 + x_2 = Hz_1 + Hz_2
= H(z_1 + z_2)
= H \cdot (t(u_1) + t(u_2))
= H \cdot t(u_1 + u_2)
= H \cdot t(Qs_1 + Qs_2 + Q_0)
= H \cdot t(Q(s_1 + s_2) + 2Q_0)
= x_{12} + H \cdot t(Q_0)
\]  

(9.9)

where \( x_{12} \) denotes the bit vector compressed from \( (s_1 + s_2) \). Hence, our compression system is homomorphic for addition. For decompressing the sum of two vectors, however, we will need to use a source model that corresponds to the sum of the two distributions. With the two sequences being drawn independently from their respective models, the distribution of the sum will thus be

\[
\mathcal{N}(\mu_1 + \mu_2, \Sigma_1 + \Sigma_2)
\]  

(9.10)

With our compression system being homomorphic, it is natural to extend our system with homomorphic encryption. In fact, any encryption homomorphic in addition can be applied to the compressed vector. However, encrypting after compression is not a flexible architecture, as we lose the ability to manipulate the final result, the benefits of which include further compression, as described in Section 9.2. Encrypting
before compression can harness the power of our compression architecture better.

In particular, we are interested in the integer vector homomorphic encryption scheme proposed in Zhou and Wornell’s 2014 paper [23], which has been shown to be secure (via a reduction from the Learning with Errors problem) and computationally viable by Yu, Lai, and Payor [22] in 2015.

Zhou’s integer vector homomorphic encryption scheme is designed for client-cloud interactions, where the client stores encrypted data in the cloud (an untrusted party), with the cloud doing blind computation on the encrypted data. The scheme supports three operations on integer vectors, namely addition, linear transformation, and weighted inner product. As a brief exploration of potential extensions to our system, incorporating this scheme into our architecture will allow homomorphic encryption and homomorphic compression for the addition operation.

Zhou’s scheme defines the ciphertext $\bar{u} \in \mathbb{Z}^n$ by secret key $K \in \mathbb{Z}^{n \times n}$ of an integer vector $u \in \mathbb{Z}^n$ to be a vector that satisfies

$$K\bar{u} = wu + e$$

(9.12)

for some large scalar $w$ and for some error vector $e$. With the secret key, decryption becomes

$$u = \left\lfloor \frac{K\bar{u}}{w} \right\rfloor$$

(9.13)

The method of encryption is mathematically heavy and is omitted here, for it distracts from our discussion in the sense that we are primarily concerned about compressing an already encrypted source and its joint decryption-decompression. Interested readers are encouraged to refer to [22], in which the encryption method is detailed.

Hence, for compressing the already encrypted source $\bar{u}^n$, we simply run our Encode
procedure without quantization, i.e. only translating and hashing, with

\[
\tilde{x}^{kb} = H \tilde{z}^{mb} = H \cdot t(\tilde{u})
\]  

(9.14)
(9.15)

and for joint decryption-decompression, we shall run loopy belief propagation on the graph in Figure 9.16, noting that the secret key \( K \in \mathbb{Z}^{n \times n} \) is used in the factor nodes in implementing the decryption equation 9.13. It should be emphasized again that the source model in this case would be that in Equation 9.10, which corresponds to the sum of the two random variables.

![Figure 9.16: The decoding graphs for our system modified for Zhou’s integer vector homomorphic encryption scheme. Note that the secret key \( K \) enters the system through factor nodes \( \mathcal{F}^X \) (purple).](image)

We observe that the architectural advantages of our general system still holds for this joint encryption-compression system. In particular, should it be found out that less bits are required for correct reconstruction due to the discovery of a better model, extra bits in the compressed bit stream can be dropped. This operation is not possible if we have compression before encryption.

We note that this homomorphic encryption-compression scheme is by no means a mature system, but rather an exploration in extending our general architecture, meant to illustrate the flexibility of our system given its modularity. In particular, being only able to compute sums of Gaussian vectors seem limiting. It should be noted that many more ideas and research need to be explored, for it is not obvious how one would extend LDPC compression to be homomorphic for linear transform and weighted inner product, the other two operations supported by Zhou's scheme, given that matrix multiplication by \( H \) is in general an non-commutative operation.
Moreover, homomorphic encryption itself is an active field of research, the interests for which was spurred by Gentry's dissertation in 2009 [9]. In particular, most existing homomorphic encryption systems are tied to Lattice-Based Encryption, which is known to be susceptible to Chosen Ciphertext Attacks, an attack model in which the adversary has access to a decryption oracle. Refining homomorphic encryption techniques, as well as the possibility of incorporating it to compression, is a problem that requires further research.

9.5 Summary

In this chapter, we presented four applications and extensions of our system, with the first two (corruption robustness and model imprecision) being realistic scenarios our system can elegantly handle, and the other two (model learning and encrypted compression) being of an exploratory nature. As we can see in these brief surveys of different ideas, we can see the immense potential of the architecture, which calls for further research.
Conclusion

We have presented a data compression algorithm for Gaussian sources that separates the concerns of source modeling and encoding. With graphical model as our representation and message passing as our main tool for decoding, we can make the structure of the algorithm extremely flexible. Its modularity allows it to be a general compressor for Gaussians, working with different sources with different underlying source models. Its sparse nature gives the algorithm a low time complexity. Its method of decoding, message passing, implicitly handles the rate-distortion trade-off, affording it a reasonable performance despite the strong restriction that the algorithm does not have information about the source at compression time.

With more data, and more types of data, available, this information age needs a data compression algorithm that is ready for change, and this architecture is a very promising candidate.
Bibliography


107


