Model-Code Separation Architectures for Compression Based on Message-Passing

by

Ying-zong Huang

B.S. with Honors, Stanford University (2004)
M.S., Stanford University (2004)

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2015

Signature of Author: ....
Department of Electrical Engineering and Computer Science
December 31, 2014

Certified by: ....
Gregory W. Wornell
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: ....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Committee on Graduate Students

Model-Code Separation Architectures for Compression Based on Message-Passing

by
Ying-zong Huang

Submitted to the Department of Electrical Engineering and Computer Science
on December 31, 2014, in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

Abstract

Data is compressible by presuming a priori knowledge known as a data model, and applying an appropriate encoding to produce a shorter description. The two aspects of compression — data modeling and coding — however are not always conceived as distinct, nor implemented as such in compression systems, leading to difficulties of an architectural nature.

For example, how would one make improvements upon a data model whose specific form has been standardized into the encoding and decoding processes? How would one design coding for new types of data such as in biology and finance, without creating a new system in each case? How would one compress data that has been encrypted when the conventional encoder requires data-in-the-clear to extract redundancy? And how would mobile acquisition devices obtain good compression with lightweight encoders? These and many other challenges can be tackled by an alternative compression architecture.

This work contributes a complete "model-code separation" system architecture for compression, based on a core set of iterative message-passing algorithms over graphical models representing the modeling and coding aspects of compression. Systems following this architecture resolve the challenges posed by current systems, and stand to benefit further from future advances in the understanding of data and the algorithms that process them.

In the main portion of this thesis, the lossless compression of binary sources is examined. Examples are compressed under the proposed architecture and compared against some of the best systems today and to theoretical limits. They show that the flexibility of model-code separation does not incur a performance penalty. Indeed, the compression performance of such systems is competitive with and sometimes superior to existing solutions.

The architecture is further extended to diverse situations of practical interest, such as mismatched and partially known models, different data and code alphabets, and lossy compression. In the process, insights into model uncertainty and universality, data representation and alphabet translation, and model-quantizer separation and low-complexity quantizer design are revealed. In many ways, the proposed architecture is uniquely suitable for understanding and tackling these problems.

Throughout, a discourse is maintained over architectural and complexity issues, with a view toward practical implementability. Of interest to system designers, issues such as rate selection, doping, and code selection are addressed, and a method similar to EXIT-chart analysis is developed for evaluating when compression is possible. Suggestions for system interfaces and algorithmic factorization are distilled, and examples showing compression with realistic data and tasks are given to complete the description of a system architecture accessible to broader adoption.

Ultimately, this work develops one architecturally principled approach toward flexible, modular, and extensible compression system design, with practical benefits. More broadly, it represents the beginning of many directions for promising research at the intersection of data compression, information theory, machine learning, coding, and random algorithms.

Thesis Supervisor: Gregory W. Wornell
Title: Professor of Electrical Engineering and Computer Science

Acknowledgements

I am indebted to the help of many people without whom this work could not have come to fruition. Foremost among them is my advisor Professor Gregory Wornell. When I first came to MIT, I had an inkling about doing the type of research represented by this work, but it would take a long journey of learning and reflection to get here. Without his investment in me, his tremendous patience and faith, his way of developing my skills and perspectives, his long-term support of my research goals, his interest and encouragement, or his technical input, this journey would have been unimaginable. For this support and guidance I feel extremely grateful, and for having his deep insight, broad vision, and consummate intellectual ideals to draw on I feel fortunate. Most of all, being enabled to create what I genuinely set out to create is my best dream fulfilled, which significance cannot be properly measured in words.

I also owe many thanks to my thesis committee for their helpful comments. Professor Devavrat Shah raised important questions that clarified parts of the work related to coding, and Dr. Emin Martinian contributed additional perspectives that led to a more refined view on performance. Over the years I have learned from my committee in many ways. Sometimes it is through their teaching, and other times through mentorship and collaboration. The seeds of a few ideas came partly from my earlier collaboration with Dr. Martinian and from his prior work on the subject.

A number of people contributed additional input into this work. Ying Liu introduced several graphical algorithms to me; Chong Wang offered a detailed explanation of Ising models; Venkat Chandar pointed out the linear processing bound due to Ancheta; David Romero referred me to the literature on large-alphabet LDPC codes; Ying Yin provided expertise on sequential hierarchical graphical models; Da Wang made suggestions about existing compression algorithms; Qing He referred me to the literature on neural networks and discriminative models; Professor Bill Freeman gave me a crash course on modern image models; and Professor Yuval Kochman, who was also a collaborator when we overlapped at SIA, identified connections to more related problems. There are others with whom I have exchanged ideas about the content or presentation of the work. I would like to thank all of them.

Given the practical angle of this work, I greatly benefited from prior research conducted in the industry with these additional collaborators: Drs. Jonathan Yedidia and Ali Azarbayejani at MERL where I developed practical graphical algorithms; Drs. John Apostolopoulos, Mitchell Trott, and Ton Kalker at HP Labs where I worked on information, multimedia, and security problems; and Drs. Sanjeev Mehrotra and Jin Li at Microsoft Research where I developed error correction systems and learned about audio compression. These researchers made a significant impact on the perspectives and tools I used to approach this work. In many ways, this topic also returns to my own earlier academic experience, and I thank those who first introduced me to a rich and beautiful field and set me on the path of finding out more: I enjoyed interacting closely with Profs. Lizhong Zheng, Vivek Goyal, and Polina Golland at MIT, and I am grateful for the wisdom and opportunities I was given by Profs. Robert Gray, Abbas El Gamal, and Brian Wandell at Stanford University.

A journey of discovery always begins from a home base and I had one of the most welcoming ones. I thank all my SIA colleagues past and present for our lively discussions, shared experiences, and fellowship. The recent cohort that saw me go through the uncertain parts of this work or shared in its satisfaction I am especially thankful for; and those who have been along since my first day — Tricia, Giovanni — or recurringly so — Maryam, Hiroyuki, Uri — add something special to the feeling of SIA as a second home. Likewise, I am fortunate to have had the company of my areal colleagues in RLE and LIDS, my classmates, and inter-office frequenters. Each person I've had the privilege to meet here has been an important part of my experience, and I remember our time with warm memories.

In undeniable ways, this endeavor has also been a journey of self-discovery alongside a journey of scholarship; certainly one could not have happened without the other. I am blessed with friends and family, here and away, who had an influence on me during these years. They challenged me and inspired me, brought me along for seriousness and folly, listened to me, humored my ideas, cheered me up, moved me deeply. Most importantly, they saw in me things I did not know I had. I express my heartfelt gratitude to them.

Lastly, I give my most reverent thanks to my parents, who sustained me in life and spirit during my age of impression, and gave me what I've always considered the ultimate support and understanding during my age of standing.

Research and training for this work have been supported in part by grants from Draper Laboratory, Air Force Office of Scientific Research, Microsoft Research, Hewlett-Packard, and by a fellowship from the National Science Foundation.

to J.Y. & Z.N.,
who endowed me with
'mind & hand'

Model-Code Separation Architectures for Compression Based on Message-Passing

Contents

List of Figures

List of Tables

1 Introduction

Data compression has a rich history of research and practice that have given us both the familiar systems we use and the science that describes their operation. There are now hundreds of compression systems that achieve good performance across targeted settings [1, 2] and some widely used algorithms (e.g. arithmetic coding, Lempel-Ziv) that are proven to attain the limits of optimal performance. Therefore, it may be surprising to learn that current systems have significant and deep-seated inadequacies that limit the wider application and continued advancement of data compression.

1.1 Motivation

To see this, let us begin with a toy example.

1.1.1 Toy example

Suppose we have an n-bit data source (Source A) that takes on two possible values, either one of

$$\overbrace{0000...00}^{n} \quad \text{or} \quad \overbrace{1111...11}^{n}$$

with equal probability. How should we design a system to compress this data?

We may arrive at a system like that in Fig. 1.1. This encoder-decoder pair (or *codec*) is ideal in one sense: it compresses n bits to 1 bit, the best we can do. It also seems like a natural and intuitive design.

However, suppose we discover that our knowledge of the source is only an approximation, and that we actually have a Markov source (Source B) with high recurrence probability, as in Fig. 1.2. Our prior codec becomes useless, as there may be sequences with both 0's and

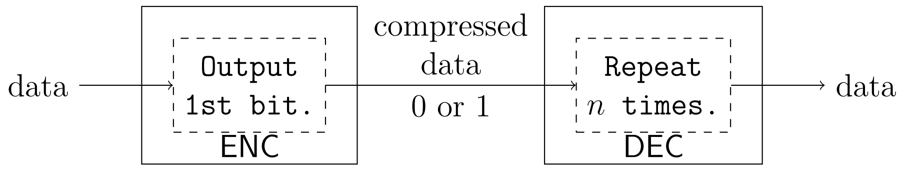


Figure 1.1: A compression system for Source A.

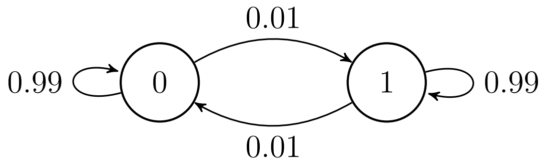


Figure 1.2: State diagram of Source B, a Markov source with recurrence probability close to 1.

1's. What is worse, it is not apparent how to modify the system to suit Source B. We would have to completely redesign the encoder to, e.g. check all the input bits, code the length of each run in the sequence, build a coding tree and apply entropy coding to those values, etc. Correspondingly, the decoder would have to be redesigned.

This example illustrates that something in the way a compression system is designed can cause its entire system pipeline to be tied to specific assumptions about data, which we call *data model*. Even small changes lead to system overhaul and incompatibility. Despite the simplicity of this example, almost all practical compression systems today have this type of design at their core.

1.1.2 Crisis in compression

The toy example captures a range of inadequacies that affect the design and use of current systems. These become more apparent as our data processing needs grow.

For example, systems over the past decades to compress a variety of data, including text, programs, audio, images, video, etc., followed a similar design process: domain experts manually analyzed the structure of data to come up with a new coding system nearly from scratch each time, relying on a great deal of artfulness and intuition. Evidently, this process does not scale with complexity or adapt to system changes well. Modern video compression systems represent a late example for a complex data type and application that took many years of joint effort by hundreds of experts to construct and standardize into a prescribed set of capabilities.

Today, our data processing needs are even more complex. Numerous domains generate prolific data sets such as financial time series, bioscience data, and search/ranking/social network databases, whose size, dynamism, and intricate internal structure present great opportunities for compression but also make them difficult to intuit. Designing compression systems for such data by artful means is extremely challenging, so they are now compressed by lower-performing or generic solutions, if at all. Alongside, there are evolving storage and communication paradigms, such as mobile devices backed by powerful cloud servers on the network. These open up exciting new modalities for applications, but demand a different allocation of information and complexity along the system pipeline. At the moment, compression has no direct answer for these concerns, having assumed local processing and bulk transmission to be normative. However, a patchwork of adjustments exist for individual applications.

To design compression systems for the present era and future, we must meet a growing list of requirements that current systems fail to satisfy, among them:

- *Design flexibility*: We should have a design process that effectively utilizes various sources of data knowledge, whether it be expert knowledge, or increasingly, machine knowledge.
- *Upgrade flexibility*: Once we design a system, we should have a method to easily change various parts of the system in order to make improvements, particularly in the data model.
- *Security*: If the data is stored with an untrusted third party provider, we want it to be compressible in the encrypted domain, securely.
- *Mobility*: We want our power constrained mobile acquisition devices to have state-of-the-art compression capability even with a lightweight encoder.
- *Distributivity*: Our data may be distributed across a network of machines or sensors, and we want them to be locally compressible.
- *Robustness*: Networks may shard compressed data for routing or corrupt them with losses, and we should allow graceful recovery without changing the compression mechanism.

Instead of viewing these requirements as isolated engineering goals, what if there is a path forward that obtains systemic flexibility generally, to support all of them? This is possible if we re-consider data compression at the level of system architecture.

1.2 Compression architecture

What is it about current systems that prevents them from having the flexibility we desire? We will see, after we categorize the many systems that now exist and realize how they are constructed similarly, that it is in how they treat data models in coding.

1.2.1 The compression landscape

From a bird's eye view, the compression landscape is divided into four domains, contingent on the amount of data model assumed and the acceptability of reconstruction imperfection (Table 1.1).

Along one axis, systems can be categorized by whether their data model is (almost fully) specified or (almost fully) unspecified at design time.

- If specified, the system is for compressing specialized data, e.g., text, audio, images, and videos; a small number of parameters may still be learned from input.

Fidelity\Model	*Specified*	*Unspecified*
Perfect	I. Entropy Coding (processing+Huffman/ arithmetic coding)	II. Universal Entropy Coding (LZW, CTW, adaptive entropy coding)
Imperfect	III. Rate-Distortion Coding (processing+ quantization+coding)	IV. Universal Rate-Distortion Coding

Table 1.1: Research problems (and some canonical designs) in compression systems. Most familiar systems are in Domains II and III, but Domain I systems are important for being the basis for them.

- If unspecified, the system is for compressing generic data (i.e. universal compression), e.g., binary files, unspecified data types, etc., where significant model structure and parameters are expected to be learned from input.

Along the other axis, compression systems can be categorized by whether we insist on perfect reconstruction or allow for imperfect reconstruction.

- If the former, then the system is for lossless compression; it is usually applied to instructional content of digital origins, e.g. programs, documents, and data files.
- If the latter, then the system is for lossy compression (i.e. rate-distortion); and it is usually applied to perceptual content of ultimately analog origins, e.g. music, photos, and movies.

Domain I is the repository of systems that compress data both losslessly and with the model fully specified. Scientific inquiry into compression began here and formed the basis upon which other systems are built. For example, Domain II systems are typically Domain I systems with additional switching, learning, or adaptivity. Domain III systems are typically Domain I systems with additional quantization. The way they perform the core coding portion of their task, however, turns out to be very similar.

Incidentally, relatively lesser attention is received in Domain I now because the *coding problem* is thought to have mature solutions. For example, provably good *entropy coders* like Huffman coding and arithmetic coding suffice in the sense that they can implement any data model. In reality, the form of the data model they require — Huffman tree, arithmetic coding intervals — presents complexity problems, such that most of the compression gain in practical systems also derives from a preceding processing stage (often a representation transform, which includes prediction), and entropy coders are only used in a clean-up role to remove simple, usually memoryless, residual structures (Fig. 1.3).

1.2.2 Joint design: an architectural dilemma

The style of design in Domain I can be termed a *joint model-code architecture*, because it treats the two tasks of compression — *data modeling* and *coding* — as one. By *data modeling*,

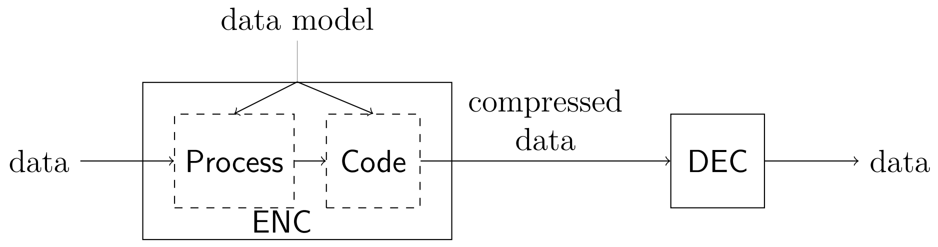


Figure 1.3: A lossless compression system with the joint model-code architecture. In this common two-stage design, the data model is used to design both Process and Code.

we mean introducing data assumptions including all prior knowledge — *data model* — into the system; and by *coding*, we mean assigning the compressed output to each reproduction of input.

Remark 1.1. Let us immediately reiterate that our definitions of *modeling* and *coding* differ from traditional terminology in important and subtle ways. There are at least two definitions of "modeling" in common use — *model learning*, as in e.g. [3, 4, 5], and *model-deconstructive processing*, as in e.g. [1]. The second definition refers essentially to Process in Fig. 1.3. Neither definition is what we mean, though systems that perform these types of "modeling" often introduce data knowledge into the system at the same time. There are also at least two definitions of "coding" in common use — *encoding*, as referring to ENC in its entirety in Fig. 1.3, and *codeword output*, as referring to a final functional block that produces output. The second definition refers essentially to Code in Fig. 1.3. Again, neither definition is what we mean, though in the simplest systems all definitions of "coding" may become equivalent.

For monolithic entropy coders like Huffman coding and arithmetic coding, the entwining of modeling and coding (as we define them) is easy to see because the data model *is* the coding machinery. But this is even true for practical two-stage systems (Fig. 1.3). Despite appearances of encoder decomposition, there is no clear distinction between what Process does and what Code does. Both partake in the modeling task and both partake in the coding task. The presumed data model (equivalently, the coding machinery) is spread across the two stages, which now must be carefully designed together.

In terms of compression systems, a joint model-code architecture is limiting. Firstly, designing algorithms to simultaneously perform modeling and coding functions requires art and compromise. We already saw how that compelled the two-stage design, but even there, a good representation for residual entropy coding may not be the natural outcome of easy pre-processing (or any pre-processing) of the input; it may be awkward to inversely represent data knowledge in the first place. Secondly, a joint architecture embeds design-time decisions into the coding machinery and therefore fossilizes the same in the fabric of the entire system pipeline; this is because the coding machinery determines everything downstream, so the compressed output and therefore the decoder are also tied to the same choices that may need to be changed later. Finally, the compressed output of a joint architecture is highly specialized and fragile, so it does not admit non-catastrophic modification, corruption, or

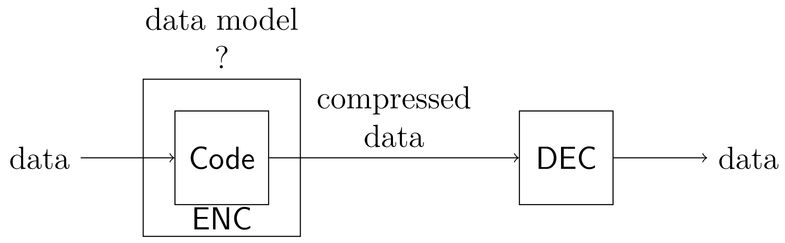


Figure 1.4: A postulated compression system with a model-code separation architecture. Code must essentially operate model-free.

further atomization unless specifically designed for. Clearly, joint model-code architecture is the true source of major systemic inflexibility in current systems.

Remark 1.2. Joint model-code architecture is not only seen in lossless compression but in lossy compression as well. JPEG image compression standards (ITU-T T.81, ITU-T T.83; 1993) define an encoding process — DCT+quantization+entropy coding — to which compliant systems must adhere. Embedded in the system is an implied data model under which still images decorrelate well in the discrete cosine transform (DCT) domain. Subsequently, research into wavelets found that they form an even better basis for images. Consequently, JPEG2000 image compression standards (ITU-T T.800, ITU-T T.803; 2002) define an encoding process that replaces the DCT with a selection of wavelet transforms. Though JPEG2000 adds many additional features to the compression format to entice adoption, the core improvement to performance is essentially the result of an improved data model. Nevertheless, JPEG and JPEG2000 are incompatible and to this day acceptance of the latter is low. Furthermore, image processing and vision research continue to improve models of images and perception, but some of the best results remain costly, if not intractable, to bring into the current joint model-code architecture of image compression systems.

So, even in the simplest setting of Domain I systems, with important ramifications for all compression systems, the story does not end with entropy coders or naive decomposition, but rather begins with the intriguing interplay between modeling and coding — an architectural question at the core of compression.

1.2.3 Separation: an architectural proposal

Suppose there is an alternative in what can be termed a *model-code separation architecture* (Fig. 1.4). In this architecture, the data model and potentially other assumptions are introduced into the system separately from the coding machinery and therefore can be altered without affecting the system as a whole.

This would require, at the least, a method of *model-free coding* that is agnostic to data assumptions, and a method of *model representation* that the coding machinery can easily use. If this can be done practically and without sacrificing compression performance, then the primary difficulties encountered so far are resolved, not to speak of other gains allowed by virtue of this systemic flexibility. This thesis posits exactly such a practical architecture.

1.3 Thesis guide

1.3.1 Highlight

In this thesis, we present a practical model-code separation architecture that divides into a part for data modelers and a part for coding theorists, mediating them using the common language of graphs and messages. Systems designed with this architecture can meet the requirements of contemporary and future compression that current systems cannot (Section 1.1.2), and in a principled and unified way.

Moreover, this architecture is flexible, modular, and extensible. Consequently, designers work within their domain expertise, and a wide range of requirements are met by building upon a baseline design. State-of-the-art tools from coding and graphical inference are incorporated in a transparent way, and system parts are reusable for situations that have not been considered. The architecture also bridges theory and practice in a way that is easy to reason about. We can evaluate performance against well defined ground truths, identify the sources of performance gain and loss, make informed design decisions, and obtain better insight into constituent parts of compression.

This thesis is developed primarily over Domain I of Table 1.1. In later parts our attention also turn to the ramifications for other categories of systems. By the conclusion, it will be apparent how to approach design under this architecture for the full spectrum of traditional and non-traditional compression scenarios that may interest researchers and practitioners.

1.3.2 Organization

The treatment of the subject is in the order of its natural development. In some chapters, additional background sections are marked by stars.

The main conceptual underpinnings of model-code separation and an adapted interpretation of several important tools are given in Chapter 2 (Background and Tools), followed by a complete description of a baseline lossless compression system in Chapter 3 (Proposed Scheme). Chapter 4 (Compressing Binary Sources) is a direct application of the described system to compressing binary data with binary codes.

Chapter 5 (Coding Details) examines the coding aspects of the system, including code selection, doping, and fine-grained decoder behavior. Chapter 6 (Modeling Details) examines the data modeling aspects of the system, addressing model mismatch, uncertainty, and universality, and introducing conceptual extensions to deal with parametric models.

Chapter 7 (Architectural Generalization) extends the system architecture to handle more complex situations involving processing of input. This is considered through the prototypical application of compressing large-alphabet sources with binary codes, the examples of which are in Chapter 8 (Compressing Large-Alphabet Sources).

Chapter 9 (Lossy Compression) considers the question of lossy compression by introducing an analogous architecture involving model-quantizer separation in additional to model-code separation. A novel low-complexity quantizer is proposed.

Chapter 10 (Toward Realistic Applications) gives a flavor of applications with real data, including the compression of grayscale images in the encrypted domain.

Finally, Chapter 11 (Conclusion) briefly summarizes the work and elaborates on a few ideas for further exploration.

1.3.3 Notation

We write e.g. s for a scalar or vector variable, depending on context. If necessary, we specify the length by a superscript, e.g. s^n, a particular element or elements by a subscript e.g. s_i. Likewise, we write s, s^n, and s_i, respectively, for the random scalar or vector version of the same; and we write s, s^n, and s_i, respectively, for constant values.

We write e.g. M for a matrix and indicate its dimensions by subscript as $M_{k\times n}$.

We write e.g. $\mathbf{S}$ for an alphabet of symbols, or a vector space.

We write e.g. $\mathcal{V}$ for a set, and e.g. $\mathscr{C}$ for a collection, class, or family of objects.

We write e.g. $\mathcal{G}$ for a graph, and put $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. We put $\mathcal{G} = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$ for a bipartite graph where edges in $\mathcal{E}$ only connect between subsets of nodes $\mathcal{V}_1$ and $\mathcal{V}_2$. We put $\mathcal{N}_i^{\mathcal{G}}$ for the open neighborhood of $v_i \in \mathcal{V}$, and $\mathcal{N}_{[i]}^{\mathcal{G}}$ for its closed neighborhood.

We write e.g. $\mathcal{Z}$ for a function value, and e.g. $\mathbb{H}(\cdot)$ for a statistical function of random variables or vectors.

We write e.g. Samp for an algorithmic pattern, subroutine, or block, and e.g. `GZIP` for a system or scheme.

We write e.g. **Sym** for miscellaneous special objects like named groups and distributions.

2 Background and Tools

To build practical compression systems for the model-code separation architecture, we will find ourselves concerned with the twin questions of low-complexity *model-free coding* and low-complexity *model representation*. To address these questions, we draw inspiration from theory and practice. This chapter introduces three important tools and interprets them for the compression context: classical source coding theory for architectural insight, sparse linear coding as a coding machinery, and probabilistic graphical models as a data model representation. Some important terminology and notational shorthands are also defined.

2.1 Source coding theory

Claude Shannon initiated the study of data processing and communication problems as statistical information problems in 1948 [6]. In particular, he introduced two *source-coding theories* for the problem of compression, one for lossless compression and one for lossy compression (i.e. rate-distortion theory, see also Chapter 9). Both theories assume the knowledge of a stochastic *source model* (a kind of data model) that represents the prior knowledge we have about the class of data we wish to compress. Given some arbitrarily long data symbols $s^n = s_1, ..., s_n$, where each s_i belongs to a finite alphabet $\mathbf{S}$, the source is taken to be a random vector s^n, whose source model is the probability mass function (pmf) $p_{\mathsf{s}^n}(\cdot)$ over $\mathbf{S}^n$. The symbols s^n are interpreted as a random drawing of s^n occurring with probability $p_{\mathsf{s}^n}(s^n)$.

2.1.1 Entropy and achievability

For lossless compression, the theory defines the *entropy rate* as

$$\mathbb{H}(\mathsf{s}) \triangleq \lim_{n\to\infty} \frac{1}{n}\mathbb{H}(\mathsf{s}^n) = \lim_{n\to\infty} \frac{1}{n}\mathbb{E}(-\log p_{\mathsf{s}^n}(\mathsf{s}^n)) \tag{2.1}$$

whose value lower bounds the average number of bits required to represent each symbol of s^n.

A *random codebook scheme* can approach this bound arbitrarily closely. Roughly, the scheme is to build a codebook $\mathscr{C}_n$ of $2^{n(\mathbb{H}(\mathsf{s})+\epsilon)}$ entries, each n symbols in length, filled entirely with random draws of s^n. By letting n grow large, an asymptotic equipartition property (AEP) ensures that all the statistically typical sequences are found in the codebook, and consequently any sequence arising from s^n can be represented by not much more than the $n(\mathbb{H}(\mathsf{s}) + \epsilon)$ bits it takes to specify a codebook entry label.[1] In this way, we obtain a compression of s^n from the original $\log|\mathbf{S}|$ bits per symbol to about $\mathbb{H}(\mathsf{s})$ bits per symbol.

An alternative, *random binning scheme* can also approach the entropy rate. In this scheme known as Slepian-Wolf coding [8], the encoder generates $2^{n(\mathbb{H}(\mathsf{s})+\epsilon)}$ "bins" and randomly assigns all sequences in $\mathbf{S}^n$ to them. AEP again ensures that each bin will contain no more than one statistically typical sequence on average, so with high probability, any sequence arising from s^n can be represented by not much more than the $n(\mathbb{H}(\mathsf{s}) + \epsilon)$ bits it takes to specify a bin label. Again we obtain compression to about $\mathbb{H}(\mathsf{s})$ bits per symbol.

2.1.2 Applied system design

The greater interest of source coding theory to this work lies in the ways it suggests for building compression systems. Already two distinctive design patterns are suggested here. The random codebook scheme typifies an *early-binding system*, where the encoder uses p_{s^n} to identify the statistically typical sequences to match against input data. The random binning scheme typifies a *late-binding system*, where the encoder only knows how many bins to create, and it is the decoder that uses p_{s^n} to identify the statistically typical sequences to infer the input data. Clearly, early-binding permits (and entices) joint model-code design, while late-binding imposes model-code separation, so indeed random-binning is a popular scheme in applications where separation is required, such as locally compressing separated sources.

More to the point, in the random binning scheme we see the key components we seek for model-code separation: randomized bin assignment as a prototype of model-free coding, and the pmf p_{s^n} as a prototype of model representation. Of course, naive random binning has complexity (e.g. of storing bin assignment, of searching bins for typical sequences) that grows exponentially in n. To that end, we need to replace the coding and representation components with lower complexity analogs.

2.2 Sparse linear coding

We now know that some of the best channel codes for reliable transmission are among the class of randomly generated sparse linear codes, made practical by message-passing decoding on their graphs. These codes are continually improved and extended for features by coding theorists. They have implementations on hardware platforms including GPU's and ASIC's. And, we have rigorous evaluations of them, including gaps to theoretical limits. Interestingly, there is evidence that these codes are also good for compressing data. In Shannon's original information theory, data compression and reliable transmission are two closely related facets — in some cases, dual problems — with similar proofs of achievable rates by random linear coding.

[1]This particular construction is restricted to stationary ergodic sources, for which the Shannon-McMillan-Breiman theorem provides the AEP. Non-ergodic sources require a different construction [7].

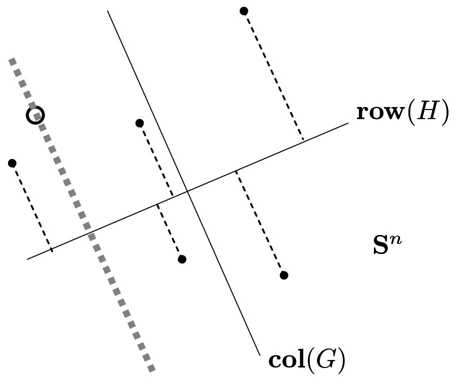


Figure 2.1: Linear subspaces of $\mathbf{S}^n$ can serve as codebooks or index collections for coding. Here, the dots indicate the set of typical words according to some probability law, and the circle indicates the true realization among them. For channel coding, $\mathbf{L} = \mathbf{col}(G)$ is the codebook. For source coding, $\mathbf{L}^\perp = \mathbf{row}(H)$ is the index collection. The gray dashed subspace is the syndrome decoding coset in channel coding, or the hash constraint set in source coding.

Next, we describe how linear codes are good for compression via a channel-coding duality and how sparse linear codes in particular make for good low-complexity model-free coding.

2.2.1 Linear codes

Linear subspaces of a finite-field vector space $\mathbf{S}^n$ are structured objects that admit low-complexity storage and processing. When identified with codebooks or index collections, they are called *linear codes*. A typical processing is an embedding or projection operation involving a *coding matrix*.

An $(n-k)$-dimensional linear subspace $\mathbf{L} \subseteq \mathbf{S}^n$ is defined by a tall matrix $G_{n\times(n-k)}$ or a wide matrix $H_{k\times n}$. Equivalently, $\mathbf{L}$ has the generator form $\mathbf{L} = \{v : v = Gu\}$ or the parity form $\mathbf{L} = \{v : 0 = Hv\}$, where $HG = 0$. In coding literature, a tall or embedding matrix like G is termed a *generator matrix*, while a wide or projection matrix like H is termed a *parity matrix*. When $\mathbf{L}$ or a related subspace is used in coding, G and H are the coding matrices that define encoder and decoder operations.

2.2.2 LDPC codes for error correction

Linear codes are prevalent for error correction because of a result in channel coding theory:

Fact 2.1. *Linear codes over finite-field alphabets (as codebooks) achieve capacity over symmetric discrete memoryless channels (DMC's) [9, 10, 11]. In particular, additive-noise DMC's are symmetric.*

In channel coding, encoding to a linear code is a complexity $\mathcal{O}(n(n-k))$ operation, as we map an information word $u \in \mathbf{S}^{n-k}$ into a codeword $v \in \mathbf{L} \subseteq \mathbf{S}^n$ by embedding with the matrix $G_{n\times(n-k)}$: $v = Gu$. With channel noise w applied, the received word set is $\mathbf{L} + w = \{y : y = Gu + w\}$, or equivalently $\mathbf{L} + w = \{y : Hw = Hy\}$. The optimal decoding seeks to recover w from Hy by maximum-likelihood (ML) inference, which has general complexity $\mathcal{O}(n\,|\mathbf{S}|^k)$ [12], too high to be useful.

For further reduction in decoding complexity, (suboptimal) iterative local decoding methods are developed to exploit additional structure in linear codes if available. For instance, low-density parity-check (LDPC) codes are linear subspaces $\mathbf{L}$ whose parity matrix H has row and column weights that become a negligible fraction of n as n grows [13, 14]. These codes approach capacity for symmetric DMC's [15, 16], and some varieties perform well even with iterative local decoding [17, 18, 19]. For such decoding, the complexity is $\mathcal{O}(k\,|\mathbf{S}|^\rho\, I)$ where ρ is the largest row weight of H, and I is the number of iterations that is $\mathcal{O}(1)$ for any fixed rate. Thus for LDPC channel coding we have $\mathcal{O}(n^2)$ encoding and $\mathcal{O}(n)$ decoding complexity.[2]

2.2.3 LDPC codes for compression

Via a coding duality where the channel law of an additive-noise DMC is identified with the source distribution of a discrete memoryless source (DMS), a related result is known:

Fact 2.2. *Linear codes over finite-field alphabets (as index collections) achieve entropy in the lossless data compression of DMS's [22, 23].*

In this case, source encoding takes the form of a projection with the matrix $H_{k\times n}$ (Fig. 2.1): $\tilde{v} = H\tilde{u}$, mapping a source word $\tilde{u} \in \mathbf{S}^n$ into an index word $\tilde{v} \in \mathbf{S}^k \cong \mathbf{L}^\perp$, at complexity $\mathcal{O}(nk)$. Source decoding seeks to recover $\tilde{u}$ from $H\tilde{u}$, again with complexity $\mathcal{O}(n\,|\mathbf{S}|^k)$ for general ML, but by using LDPC codes with iterative local decoding, the equivalent result has these codes approach entropy for DMS's at low complexity: the encoding complexity is $\mathcal{O}(\rho k)$ while the decoding complexity is $\mathcal{O}(k\,|\mathbf{S}|^\rho\, I)$ as in channel coding. In essence, both encoding and decoding are $\mathcal{O}(n)$.

It is asserted in [24] (Theorem 2.1) that linear codes also achieve entropy rate for general sources, including sources with memory and non-stationary sources. While this does not automatically imply that LDPC codes approach the same, particularly with other than ML decoding, recent results in LDPC coding for channels with memory [25, 26] provide grounds for optimism that a much larger class of sources than DMS's too lay open to high-performance, low-complexity LDPC coding.

2.2.4 Applied system design

LDPC source coding applies the source distribution at the decoder (as does LDPC channel coding with the channel law), so it forms a late-binding system. With its linear encoding and decoding complexity and entropy-approaching performance characteristics, it is an ideal candidate for low-complexity model-free coding. Indeed, in many distributed source coding, or "practical Slepian-Wolf" problems, we see their use frequently.

[2] $\mathcal{O}(n)$ encoding is possible [20]. Linear programming (LP) decoding may have similar complexity [21].

2.3 Probabilistic graphical models

In recently years, the machine learning community has refined a toolkit based on probabilistic graphical models to perform important reasoning tasks on complex data. Among them are a compact way to represent statistical structures in data, inference algorithms to compute updates from prior knowledge, and learning algorithms to train models automatically from data (or with some intervention from domain experts). While these tools are used primarily for data mining and understanding, they can be turned equally well to the problem of data compression.

Next, we describe how graphical models represent data and how iterative algorithms make them good for low-complexity model representation.

2.3.1 Graphical model representation

Choosing a good data model representation is an art. We are simultaneously concerned with the richness, specificity, and complexity of the representation. Some examples of representations are enumeration, list of examples, list of features, algorithms (i.e. functions) that generate examples, or data structures (i.e. mathematical objects) that are interpreted by algorithms to generate examples or filter features.

Over time, graphs have become a preferred tool for model representation in machine learning and statistics. Directed acyclic graphs (DAG's) are used to represent causal dependency in Bayesian networks, undirected graphs (UG's) are used to represent configurational compatibility in Markov random fields (MRF's), factor graphs (FG's) are used to represent factorization of probability distributions, and computation graphs are used to represent functional composition in neural networks (NN's). In each case, graphs admit low-complexity storage for the data model and double as data structure for low-complexity, local computation.

Probabilistic graphical models (PGM's) refer to data models built on DAG-represented causality assertions and UG-represented compatibility assertions [27]. The general representational power of PGM's is cemented by the following result:

Fact 2.3. *Any strictly positive distribution p_{s^n} over $\mathbf{S}^n$ can be represented by an undirected graph $\mathcal{G} = (\mathcal{S}, \mathcal{E})$ over the n source nodes $\mathcal{S} \triangleq \{\mathsf{s}_1, ..., \mathsf{s}_n\}$, in the sense that p_{s^n} simultaneously factorizes over the maximal cliques of $\mathcal{G}$,*

$$p_{\mathsf{s}^n}(s^n) = \frac{1}{\mathcal{Z}} \prod_{C \in \mathrm{cl}(\mathcal{G})} \psi_C(s_C)$$

and (by the Hammersley-Clifford theorem [28]) respects the list of global conditional independence relationships generated by graph-separation tests on $\mathcal{G}$ [29, 30].

A similar statement exists regarding distributions p_{s^n} whose factors ψ_C are conditional probability distributions $p_{\mathsf{s}_i|\mathsf{s}_{\pi_i}}$ defined over DAG nodes s_i and parents s_{π_i}. Thus, PGM's have the feature that the representational complexity of the data model (as measured by the number and size of statistical assertions that compose the graph) directly relates to the algebraic complexity of p_{s^n} that sets the complexity of many Bayesian computations.

2.3.2 Iterative algorithms

Sampling and marginalization are two important tasks on data models. Being able to construct iterative algorithms with local computations for both tasks is what makes PGM's particularly useful for compression. Marginalization (Section 2.3.2.2) in particular is a key step in decoding for the model-code separation architecture, which we will see many times later.

2.3.2.1 Gibbs sampling

Sampling a Markov random field $\mathsf{s}^n \sim p_{\mathsf{s}^n}$ benefits from its undirected graph representation $\mathcal{G}$ by the method of local Gibbs sampling. Suppose we have a single-element sampler $\mathsf{Samp}(q_{\mathsf{s}})$ capable of drawing $s \in \mathbf{S}$ according to a (possibly unnormalized) distribution $q_{\mathsf{s}}(\cdot)$ in $\mathcal{O}(|\mathbf{S}|)$ time. Begin with an initialization s^n and a walking order $\sigma \in \mathbf{Sym}(n)$. In each pass of an iterative procedure, we re-sample all n values of s^n in turn,

$$i = \sigma(1), ..., \sigma(n): \quad s_i \Leftarrow \mathsf{Samp}\left(p_{\mathsf{s}^n}(\cdot, s_{\sim i})\right) \equiv \mathsf{Samp}\left(\prod_{C \in \mathrm{cl}(\mathcal{G}): \mathsf{s}_i \in C} \psi_C(\cdot, s_{C \setminus i})\right)$$

where the right-hand equivalence uses the relationship $\mathsf{s}_i \perp\!\!\!\perp \mathcal{S} \backslash \mathcal{N}^{\mathcal{G}}_{[i]} \;\Big|\; \mathcal{N}^{\mathcal{G}}_i$ to provide local computation.

Under some technical conditions, an asymptotically independent sample from p_{s^n} is obtained after enough iterations. This procedure has $\mathcal{O}(\kappa\, |\mathbf{S}|^\eta)$ storage complexity where κ is the number of cliques and η is the maximum clique size of $\mathcal{G}$, and essentially $\mathcal{O}(n\, |\mathbf{S}|\, I)$ time complexity where I is the number of iterations.

2.3.2.2 Belief-propagation marginalization

Approximate marginalization of $\mathsf{s}^n \sim p_{\mathsf{s}^n}$ can be carried out on one of its PGM representations using the iterative sum-product or belief-propagation (BP) algorithm.

Because BP performs repeated operations on the factors of p_{s^n}, it is best described in terms of a graph whose structure shows the factors explicitly. To do so when the data model is provided by a UG $\mathcal{G} = (\mathcal{S}, \mathcal{E})$, we convert $\mathcal{G}$ to its equivalent bipartite *factor graph* form $\mathcal{F} = (\mathcal{S}, \Psi, \mathcal{E}')$, where $\mathcal{S}$ are *variable nodes*, Ψ are *factor nodes* (one corresponding to each unfactorizable clique C of $\mathcal{G}$), and $\mathcal{E}' = \{(\psi_C, \mathsf{s}_i) : \psi_C \in \Psi, \mathsf{s}_i \in C\}$ are edges between factor nodes and their corresponding clique variable nodes (Fig. 2.2). Then, let us view each node in the graph as a local computation device accepting inputs and emitting outputs on its edges, or *ports*. For convenience, label input or output ports of a node by the adjacent node to which they are connected in $\mathcal{F}$.

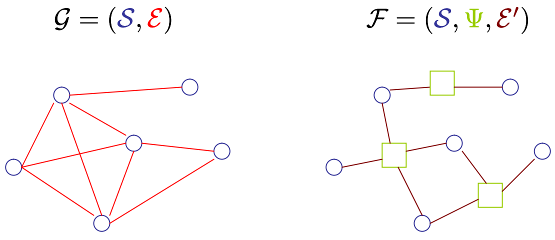

$\mathcal{G} = (\mathcal{S}, \mathcal{E})$
$\mathcal{F} = (\mathcal{S}, \Psi, \mathcal{E}')$

Figure 2.2: Conversion of an undirected graph $\mathcal{G}$ (left) to its factor graph $\mathcal{F}$ (right), assuming the maximal cliques of $\mathcal{G}$ represent factors of p_{s^n} that cannot be further factored.

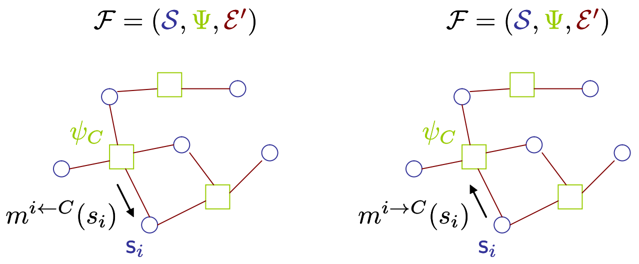

$\mathcal{F} = (\mathcal{S}, \Psi, \mathcal{E}')$
ψ_C
$m^{i\leftarrow C}(s_i)$
s_i
$\mathcal{F} = (\mathcal{S}, \Psi, \mathcal{E}')$
ψ_C
$m^{i\rightarrow C}(s_i)$
s_i

Figure 2.3: Examples of nodes exchanging messages (unary functions) on the factor graph $\mathcal{F}$, viewed as local I/O.

Notationally it is also useful to write factors and factor operations compactly. Any function $f(s_1, ..., s_l)$ over l variables, $f : \mathbf{S}^l \to \mathbf{R}$, can be represented by a tensor $f^{i_{l'+1},...,i_l}_{i_1,...,i_{l'}}$ of total order l. It can be interpreted as a machine with number of input (subscripted indices) and output (superscripted indices) ports summing to l. We may therefore refer to a function and its tensor interchangeably. For the special case of a $(1, 0)$-tensor m^i representing a unary function $m : \mathbf{S} \to \mathbf{R}$, we call it a *message*.

Define two atomic operations on messages,

- The *accumulation* operation on a set of messages $\mathcal{M}$: $\circ_{m\in\mathcal{M}} m$, where $\circ$ is the Hadamard product.

- The *filtration* operation by a factor $\psi : \mathbf{S}^l \to \mathbf{R}$ on a set of messages $\mathcal{M}$: $\psi(\otimes_{m\in\mathcal{M}} m)$, where $\otimes$ is the tensor product, and the leftmost operation is a tensor contraction.

With these, we can finally describe the computation rules governing BP. Referring to Fig. 2.3, denote a *variable-node output message* of the node s_i, on the port ψ_C, by $m^{i\rightarrow C}(s_i)$. Denote a *factor-node output message* of the node ψ_C, on the port s_i, by $m^{i\leftarrow C}(s_i)$. Thus to each edge of $\mathcal{E}'$ are associated two directional messages, so of course $m^{i\rightarrow C}(s_i)$ and $m^{i\leftarrow C}(s_i)$ are also respectively a *factor-node input message* and a *variable-node input message*. Note that we may conveniently shorten e.g. $m^{i\rightarrow C}(s_i)$ to $m^{i\rightarrow C}$, with the variable being understood and $m^{i\rightarrow C}$ being treated as the tensor m^i with additional decoration marking port and I/O direction.

Define the three node computations of BP,

- The *variable node output computation.* For every output port ψ_C of s_i, accumulate messages on ports not ψ_C:

$$m^{i\to C}(s_i) \Leftarrow \prod_{\psi_D \in \mathcal{N}_i^{\mathcal{F}} \setminus \psi_C} m^{i\leftarrow D}(s_i) \tag{2.2}$$

  where $\mathcal{N}_i^{\mathcal{F}}$ is the set of factor nodes adjacent to s_i. In tensor notation,

$$m^{i\to C} \Leftarrow \circ_{\psi_D \in \mathcal{N}_i^{\mathcal{F}} \setminus \psi_C} m^{i\leftarrow D} \tag{2.3}$$

- The *factor node output computation.* For every output port s_i of ψ_C, filter messages on ports not s_i by ψ_C:

$$m^{i\leftarrow C}(s_i) \Leftarrow \sum_{s_{C\setminus i}} \psi_C(s_i, s_{C\setminus i}) \prod_{\mathsf{s}_j \in \mathcal{N}_C^{\mathcal{F}} \setminus \mathsf{s}_i} m^{j\to C}(s_j) \tag{2.4}$$

  where $\mathcal{N}_C^{\mathcal{F}}$ is the set of variable nodes adjacent to ψ_C, and the left summation is over the values of all the variables indicated. In tensor notation,

$$m^{i\leftarrow C} \Leftarrow \psi^i_{C\setminus i}(\otimes_{\mathsf{s}_j \in \mathcal{N}_C^{\mathcal{F}} \setminus \mathsf{s}_i} m^{j\to C}) \tag{2.5}$$

- The *total belief computation.* For every s_i, accumulate messages on all ports:

$$b^i(s_i) = \prod_{\psi_D \in \mathcal{N}_i^{\mathcal{F}}} m^{i\leftarrow D}(s_i) \tag{2.6}$$

  In tensor notation,

$$b^i = \circ_{\psi_D \in \mathcal{N}_i^{\mathcal{F}}} m^{i\leftarrow D} \tag{2.7}$$

We further simplify the tensor notation by writing the three computations in shorthand respectively as:

Variable node output:	$m^{i\to C} \Leftarrow m^{i\leftarrow\sim C}$
Factor node output:	$m^{i\leftarrow C} \Leftarrow \psi^i_{\sim i} m^{\sim i\to C}$
Total belief computation:	$b^i = m^{i\leftarrow *}$

Table 2.1: Tensor shorthand notation for the node computations of the BP algorithm. A message with a higher-order tensor superscript means it is formed by a tensor product. Wildcards refer to an open neighborhood, either $\mathcal{N}_i^{\mathcal{F}}$ or $\mathcal{N}_C^{\mathcal{F}}$, relative to the other side of the arrow.

We use this shorthand notation in other parts of the thesis where the meaning is unambiguous.

BP begins with an initialization of all messages on edges and some walking order over the nodes known as a *schedule*, and iteratively applies the variable node and factor node output computations until convergence (if it occurs). The total belief computation obtains the marginal estimation by

$$\hat{p}_{\mathsf{s}_i} = b^i / \left\| b^i \right\|_1 \tag{2.8}$$

If we are only interested in an inferential result as is usually the case, such as the likely value of s_i, estimating from this approximate marginal suffices, i.e. $\hat{s}_i = \arg\max_{s_i} \hat{p}_{\mathsf{s}_i}(s_i)$, or simply

$$\hat{s}_i = \arg\max_{s_i} b^i(s_i) \tag{2.9}$$

Note that BP may not converge, nor give the correct marginals even if it does, in the presence of graph loops. However, many empirical results and some theoretical analyses show that its marginal approximations are often good enough for correct inference. The complexity of BP marginalization is $\mathcal{O}(\kappa\, |\mathbf{S}|^\eta\, I)$ [3] — where κ is the number of unfactorizable cliques, η is the maximum unfactorizable clique size of $\mathcal{G}$, and I is the number of iterations — being largely bottlenecked by the tensor product in the factor node output computation.

2.3.3 Pairwise models

A less powerful class of *pairwise models* whose factorization takes the form

$$p_{\mathsf{s}^n}(s^n) = \frac{1}{\mathcal{Z}} \prod_{\mathsf{s}_i \in \mathcal{S}} \phi_i(s_i) \prod_{(\mathsf{s}_i, \mathsf{s}_j) \in \mathcal{E}} \psi_{ij}(s_i, s_j) \tag{2.10}$$

are frequently used to represent data — pairwise relationships are sometimes better estimated from empirical statistics. These also represent data models with limited complexity: BP on these models encounters $\eta = 2$ and $\kappa = \mathcal{O}(n + |\mathcal{E}|)$, so marginalization complexity is at most $\mathcal{O}(n^2)$, though usually closer to $\mathcal{O}(n)$.[4] We will see these models in the remainder of the thesis, so let us remark on running BP on them.

The node computations of the BP algorithm are somewhat simplified because each bilateral factor node only has two neighbors, so we can write

$$\begin{aligned} \text{Variable node output:} \quad & m^{i \to (i,j)} \Leftarrow \phi^i m^{i \leftarrow (i, \sim j)} \\ \text{Factor node output:} \quad & m^{i \leftarrow (i,j)} \Leftarrow \psi^i_j m^{j \to (i,j)} \\ \text{Total belief computation:} \quad & b^i = m^{i \leftarrow *} \end{aligned}$$

The fact that the factor node computation is so simple means it is customarily rolled into

[3]This can be reduced in cases with further model structure [31].

[4]Even for triangle-free graphs, the maximum number of edges can be $\lfloor n^2/4 \rfloor$ by Turán's theorem; however, inherently local models like finite memory sources have bounded variable degree, and sources for which ergodic effects are apparent at n (most usefully compressible sources) have variable degree negligible compared to n, both cases giving sub-quadratic complexity.

the variable node computation, and we effectively eliminate factor nodes completely and understand neighbors of s_i as its variable node neighbors in $\mathcal{G}$:

Variable node output:	$m^{j\leftarrow i} \Leftarrow \psi_i^j \phi^i m^{i\leftarrow\sim j}$
Total belief computation:	$b^i = \phi^i m^{i\leftarrow *}$

Table 2.2: Tensor shorthand notation for the pairwise model without factor nodes.

Thus UG's without unfactorizable cliques larger than 2 do not need to be converted to FG's explicitly for BP because the UG's are simply FG's with the unambiguous factor nodes elided.

2.3.4 Applied system design

PGM's allow sampling and approximate marginalization on data models at complexity scaling with a particular statistical notion of model complexity. For the large class of real-world data models for which PGM's do give a succinct description, they can produce and verify statistically typical sequences with an acceptable degree of practicality, and therefore serve as an essential candidate for low-complexity model representation.

2.4 Summary

The Shannon random codebook scheme and the Slepian-Wolf random binning scheme represent two design patterns for lossless compression systems. The former is the inspiration for the lineage of current systems, while the latter gives us the insight we need to build systems with a new model-code separation architecture.

With LDPC codes from channel coding and PGM's from machine learning, we now have the necessary tools to implement low-complexity model-free coding and model representation, respectively, for practical random binning, and therefore practical model-code separation design.

3 Proposed Scheme

In this chapter, we introduce a basic compression system with the model-code separation architecture. We show a canonical design for lossless compression, featuring a model-free *hashing encoder* and a process-replicating *inferential decoder*. In later parts of the chapter, we discuss unique questions that arise about this design. But let us begin by returning to the toy example of Chapter 1 to show how this architecture differs already from existing methods.

3.1 Toy example

Recall that in the toy example (Section 1.1.1), we have Source A that takes on two possible values, n bits of all 0 or all 1, with equal probability (so the entropy is 1 bit), and Source B that is Markov with recurrence probability q close to 1 (so that the entropy is $(n-1)h(q)+1$ bits).[1] Our prior codec could not deal with the switch from Source A to Source B, even though they are quite similar with similar entropy.

But imagine if the encoder can compute a k-bit hash on the input sequence that divides all input sequences into 2^k groups. Let this hash be the compressed output. In particular, say that the all 0 and all 1 sequences do not collide for any k. Then if we know we have Source A, the decoder can check which of these two sequences hashes to a 1-bit compressed value and infer the correct sequence, thus compressing to the source entropy. If it turns out we have Source B, the decoder can be modified to check which of the $2^{(n-1)h(q)+1}$ typical sequences hashes to the $(n-1)h(q)+1$ bits of compressed value (again, assuming no hash collision), and infer the correct sequence. In this case, the encoder does not need to be modified because it is model-free, and the decoder contains the same inference engine, only needing to take into account which are the typical sequences by referring to the data model.

We have already seen in Section 2.1 that this is possible with the Slepian-Wolf random binning scheme, but its complexity is prohibitive. However, in Sections 2.2 and 2.3, we introduced LDPC codes and PGM's as components that may take the roles of low-complexity model-free coding and low-complexity model representation, respectively. Next we show how these components fulfill the promise of a practical model-code separation compression architecture.

[1] $h(x) \triangleq -x\log x - (1-x)\log(1-x)$.

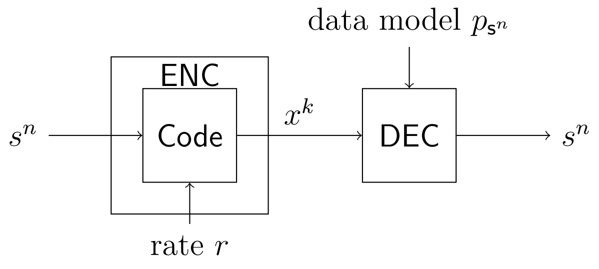


Figure 3.1: System diagram for the model-code separation architecture. On the left is a model-free encoder. On the right is an inferential decoder.

3.2 Basic system construction

We describe a baseline lossless compression system with the model-code separation architecture (Fig. 3.1) and sketch the requisite components that make a complete implementation. In this chapter, assume all objects are over the finite-field alphabet $\mathbf{S} = \mathbf{GF}(q)$.

3.2.1 Required inputs

Let $\mathrm{s}^n \in \mathbf{S}^n$ be an n-vector *source data* sequence presumed to be a random sample of s^n. To compress it, we require

- A code: a collection $\mathscr{H}(n,k)$ of $k \times n$ parity matrices of a rate k/n linear source code ensemble; to ease presentation, any specific example assumes an LDPC code ensemble (Section 2.2.3).
- A data model: p_{s^n} in some factored form or in a PGM representation; to ease presentation, any specific example assumes a pairwise model (Section 2.3.3).

3.2.2 Encoder

The encoder, as promised, is model-free and performs a nearly trivial hashing operation. Setting k to target $r_{\text{code}} = k/n$ as the *nominal compression rate* (further discussed in Section 3.2.5), and choosing a random $H \in \mathscr{H}(n,k)$ as the encoding matrix, it produces

$$\mathrm{x}^k = H\mathrm{s}^n \tag{3.1}$$

as the compressed result. A variety of terms can describe this encoding, such as binning, projection, or hashing.

The encoder produces some additional output described in Section 3.2.4.

3.2.3 Decoder

The compressed value x^k corresponds to many possible s^n, so a decoder needs to apply additional information to recover the true s^n. This is done by a graphical inference engine in the decoder that applies both the coding constraints of H and the data model p_{s^n}.

3.2.3.1 Code subgraph

Since H enforces k hard constraints of the form $x_a = \sum_{i=1}^{n} H_{a,i} s_i$, it can be represented by a bipartite factor graph $\mathcal{C} = (\mathcal{S}, \mathcal{X}, \mathcal{F})$, with k factor nodes $\mathcal{X} \triangleq \{f_1, ..., f_k\}$ and n source nodes $\mathcal{S} \triangleq \{\mathsf{s}_1, ..., \mathsf{s}_n\}$. There is an edge between factor node f_a and source node s_i if and only if $H_{a,i} \neq 0$, forming the neighborhoods denoted by $A = \mathcal{N}_a^{\mathcal{C}}$.

The meaning of $\mathcal{C}$ is observed by viewing it as the graph for the *hash constraint function*

$$c(s^n) \triangleq \prod_{a=1}^{k} f_a(s_A) = \prod_{a=1}^{k} \mathbb{1}\left\{\mathrm{x}_a = \sum_{i:H_{a,i}\neq 0} H_{a,i} s_i\right\}(s_A) \tag{3.2}$$

where each factor is an indicator on one hard constraint, and $c(s^n) = 1$ if and only if all constraints are satisfied. Since $c(s^n)$ is an (unnormalized) probability distribution, the entire PGM machinery of Section 2.3 applies. Indeed, iterative local decoding of the linear source code with H is identical to approximate marginalization of $c(s^n)$ on $\mathcal{C}$.

$\mathcal{C}$ is called the code subgraph.

3.2.3.2 Source subgraph

If p_{s^n} is available algebraically, it is represented directly by a factor graph. If the data model is given as a directed or undirected PGM, we can convert it to factor graph form using the procedure in Section 2.3.2.2.

Without loss of generality, let us suppose $\mathsf{s}^n \sim p_{\mathsf{s}^n}$ is represented by an undirected graph $\mathcal{G} = (\mathcal{S}, \mathcal{E})$ over the n source nodes $\mathcal{S} \triangleq \{\mathsf{s}_1, ..., \mathsf{s}_n\}$, in the sense that p_{s^n} is factored over the maximal cliques of $\mathcal{G}$:

$$p_{\mathsf{s}^n}(s^n) = \frac{1}{\mathcal{Z}} \prod_{C \in \mathrm{cl}(\mathcal{G})} \psi_C(s_C) \tag{3.3}$$

Then let its equivalent factor graph form be $\mathcal{G}' = (\mathcal{S}, \Psi, \mathcal{E}')$.

Either $\mathcal{G}$ or $\mathcal{G}'$ may be called the source subgraph. For pairwise models, we refer to UG's and elided FG's both as $\mathcal{G}$, since they are identical (see Section 2.3.3).

3.2.3.3 Decoding algorithm

Let $\mathcal{U} \triangleq \mathcal{G}' \cup \mathcal{C} = (\mathcal{S}, \Psi \cup \mathcal{X}, \mathcal{E}' \cup \mathcal{F})$ be a combined source-code graph in factor graph form, where the source nodes $\mathcal{S}$ are shared between the source and code subgraphs (Fig. 3.2 shows an example for a pairwise model where Ψ nodes are elided). The source nodes divide $\mathcal{U}$ into a source side and a code side.

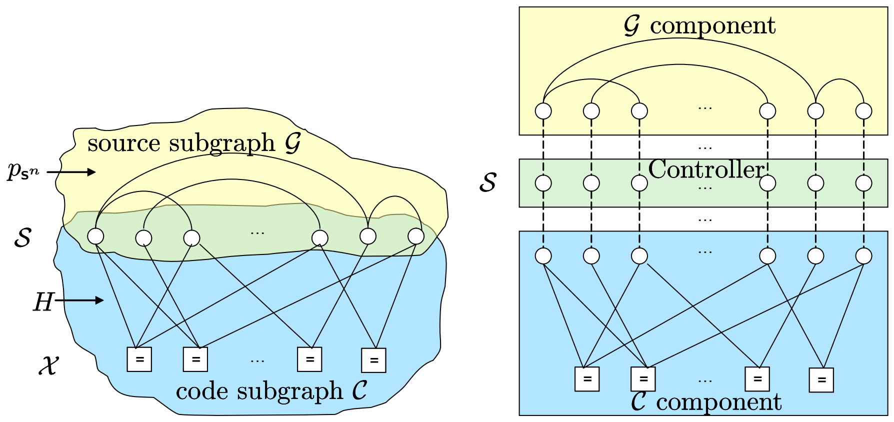


Figure 3.2: The structure of the combined decoder source-code graph $\mathcal{U}$ for the system in Section 3.2 (left). The differently colored subgraphs behave as if modular components connected via dashed virtual ports (right).

The decoder runs belief propagation (BP) on the combined graph, representing approximate marginalization of the *joint objective*

$$u(s^n) \triangleq c(s^n)p_{\mathsf{s}^n}(s^n) \tag{3.4}$$

For each source node s_i there are now two sets of neighbor nodes, one on each side. It is notable that only the source nodes interact with both sides. The factor nodes only interact with nodes in its own subgraph. In the combined BP, messages are passed along all edges, and we can derive the expressions for computation on the three sets of nodes, $\mathcal{S}$, Ψ and $\mathcal{X}$ (notation as in Table 2.1).

First, the nodes that do not interact with another side directly (we use μ for source-side messages and m for code-side messages; wildcards only expand within the subgraph of their message):

- Ψ-*node output computation.* For every output port s_i of ψ_C, filter messages on ports not s_i by ψ_C:
$$\mu^{i\leftarrow C} \Leftarrow \psi^i_{\sim i}\mu^{\sim i\rightarrow C} \tag{3.5}$$
- $\mathcal{X}$-*node output computation.* For every output port s_i of f_a, filter messages on ports not s_i by f_a:
$$m^{i\leftarrow a} \Leftarrow f^i_{\sim i}m^{\sim i\rightarrow a} \tag{3.6}$$

Next, nodes that open ports to both sides of the joint graph:

- $\mathcal{S}$-node *output computation – source side.* For every output port ψ_C of s_i, accumulate messages on ports not ψ_C:
$$\mu^{i\rightarrow C} \Leftarrow \mu^{i\leftarrow\sim C}\left[m^{i\leftarrow *}\right] \tag{3.7}$$

- *$\mathcal{S}$-node output computation – code side.* For every output port f_a of s_i, accumulate messages on ports not f_a:

$$m^{i\to a} \Leftarrow m^{i\leftarrow\sim a}\left[\mu^{i\leftarrow *}\right] \tag{3.8}$$

And finally,

- *Total belief computation.* For every s_i, accumulate messages on all ports:

$$b^i = \left[\mu^{i\leftarrow *}\right]\left[m^{i\leftarrow *}\right] \tag{3.9}$$

Let us note that the bracketed terms, despite being Hadamard products of many messages, may be conceptually interpreted as single lumped messages that do not change with respect to which port the output is being emitted on. Therefore, when a node of $\mathcal{S}$ is emitting on one subgraph, the lumped message from the other subgraph can be pre-computed and treated like an *external message* on a *virtual port* opened on the latter's side. This means BP on $\mathcal{U}$ is exactly the same as BP on each subgraph alone, with the simple addition of an external I/O port and some computation to figure the external message. This makes the decoder (and by extension, the entire system) architecturally modular, with external messages and virtual ports acting as the only interface between graph-inferential components.

We summarize by restating the three computations of any subgraph component whose internal factorization is like that described in Section 2.3.2.2:

Variable node output:	$m^{i\to C} \Leftarrow m^{i\leftarrow\sim C}\left[M^{i\leftarrow}\right]$
Factor node output:	$m^{i\leftarrow C} \Leftarrow \psi^{i}_{\sim i} m^{\sim i\to C}$
External message output:	$\left[M^{i\to}\right] \Leftarrow m^{i\leftarrow *}$

Table 3.1: Node computations of the BP algorithm within a component, with a virtual port opened for each s_i to receive external message $M^{i\leftarrow}$ from the outside and emit external message $M^{i\to}$ to the outside.

Finally, while each component can compute a total belief using external messages, it makes sense for a main *controller* to handle component-agnostic operations. The controller is simply another component over the port-preserving "intersection" subgraph $\mathcal{A} \triangleq \mathcal{G}' \cap^* \mathcal{C} = (\mathcal{S}, \emptyset, \mathcal{E}' \cup \mathcal{F})$. All of its ports are to be interpreted as virtual ports to receive external messages, thus for a controller,

$$\text{Total belief computation:} \qquad b^i = \left[M^{i\leftarrow *}\right]$$

where the wildcard is taken over all virtual ports.

The modular computations for the pairwise model is given as example in Table 3.2 and the setup is depicted in Fig. 3.2.

$\mathcal{G}$ component –

$$\mathcal{S}\text{ node output:}\quad \mu^{j\leftarrow i} \Leftarrow \psi_i^j \phi^i \mu^{i\leftarrow\sim j}\left[M^{i\leftarrow}\right]_{\mathcal{G}} \tag{3.10}$$

$$\mathcal{S}\text{ external message output:}\quad \left[M^{i\rightarrow}\right]_{\mathcal{G}} \Leftarrow \phi^i \mu^{i\leftarrow *}$$

$\mathcal{C}$ component –

$$\mathcal{S}\text{ node output:}\quad m^{i\rightarrow a} \Leftarrow m^{i\leftarrow\sim a}\left[M^{i\leftarrow}\right]_{\mathcal{C}} \tag{3.11}$$

$$\mathcal{X}\text{ node output:}\quad m^{i\leftarrow a} \Leftarrow f^i_{\sim i} m^{\sim i\rightarrow a} \tag{3.12}$$

$$\mathcal{S}\text{ external message output:}\quad \left[M^{i\rightarrow}\right]_{\mathcal{C}} \Leftarrow m^{i\leftarrow *}$$

Controller –

$$\begin{aligned} \text{Connectivity:}\quad & \left[M^{i\leftarrow}\right]_{\mathcal{G}} = \left[M^{i\rightarrow\mathcal{G}}\right] \Leftarrow \left[M^{i\leftarrow\mathcal{C}}\right] = \left[M^{i\rightarrow}\right]_{\mathcal{C}} \\ & \left[M^{i\leftarrow}\right]_{\mathcal{C}} = \left[M^{i\rightarrow\mathcal{C}}\right] \Leftarrow \left[M^{i\leftarrow\mathcal{G}}\right] = \left[M^{i\rightarrow}\right]_{\mathcal{G}} \\ \text{Total belief computation:}\quad & b^i = \left[M^{i\leftarrow\mathcal{G}}\right]\left[M^{i\leftarrow\mathcal{C}}\right] = \left[M^{i\rightarrow}\right]_{\mathcal{G}}\left[M^{i\rightarrow}\right]_{\mathcal{C}} \end{aligned} \tag{3.13}$$

Table 3.2: Node computations of the BP algorithm for the various components in a decoder for the pairwise model. Component external port connections are made in the controller.

The only additional point worth belaboring is the schedule of passing messages, which a modular decoder greatly simplifies. The fact that we compute external messages in one pass along with all the messages within a component means, sans further communication solely for scheduling purposes, the decision is focused at the level of component interactions. Here there are only a few obvious choices. A parallel macro-schedule would have each component compute internal messages, then exchange external messages at the same time. A serial macro-schedule would have one component active at a time, and present its latest external messages to the next component. This serial macro-schedule, alternating between source message passing and code message passing, is what we use in this work. Within a component, we use a parallel schedule.

For the decompression problem, BP is run until convergence at the controller or declaration of failure. In the first case, we put

$$\hat{s}_i = \arg\max_{s_i} b^i(s_i) \tag{3.14}$$

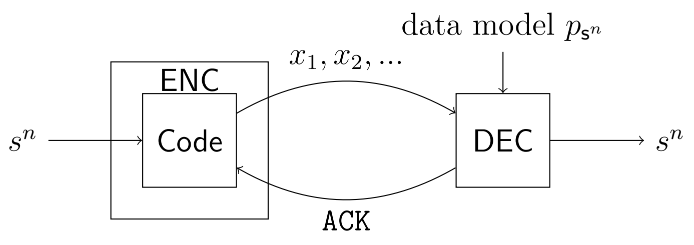


Figure 3.3: A rateless system with minimal decoder acknowledgement on the feedback.

as the decompressed output.[2]

3.2.4 Doping symbols

Depending on the source model, the decoding process may not begin without some non-trivial initial beliefs (trivial belief being the *identity message* $1(s) = 1/|\mathbf{S}|$). Therefore the encoder randomly selects a fraction r_{dope} of source nodes $\mathcal{D} \subseteq \mathcal{S}$ to describe directly to the decoder. They are presented as deterministic *controller messages*

$$d^{\mathcal{D}}(s_{\mathcal{D}}) = \mathbb{1}\{s_{\mathcal{D}} = \mathrm{s}_{\mathcal{D}}\}(s_{\mathcal{D}}) \tag{3.15}$$

and accumulated to all external messages passing through the controller nodes (Table 3.2 modified accordingly). These known "doping" symbols anchor the decoding process, and only a small amount — which can be optimized — is necessary.

In addition to considering the doping process as belonging to the controller, we can also consider it as a part of the data model or the code. If the latter, we can consider it as augmenting H with additional unit-weight checksum rows to make the true encoding matrix $H^{\triangle}$ with *total coding rate* $r = r_{\text{code}} + r_{\text{dope}}$, and encoding as $\mathrm{x} = H^{\triangle}\mathrm{s}$.[3]

Doping is discussed in more detail, with experimental results, in Section 5.2.

3.2.5 Rate selection

The question of rate selection is nuanced. The system presented can be used in many ways. As a direct no-feedback, *fixed-rate system*, the model-free encoder needs the compression rate r to be supplied (Fig. 3.1). This may come from entropy estimates or upstream hints, analogous to capacity estimates in channel coding, and comes with the possibility of vanishingly small but non-zero decoding error even when r is set above the entropy rate.

[2]The desired maximization is actually $\hat{s}^n = \arg\max_{s^n} u(s^n)$ as in ML. However, the *approximate* version of the max-product algorithm that would compute this also applies marginal maximizations for graphs with loops, without guaranteeing inter-symbol consistency [32].

[3]We do not need to communicate the actual content of the matrix H or the locations of the selected doping symbols, if the encoder/decoder pair synchronize on a single random seed. This can be done out-of-band, or the random seed can be included in an initial uncoded header.

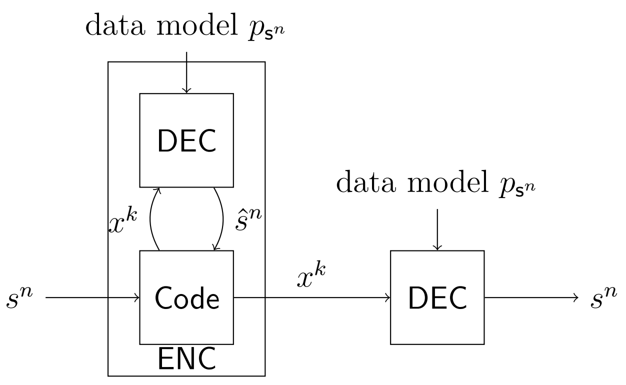


Figure 3.4: A zero-error, variable-rate system featuring a decoder simulation within the encoder and internal feedback.

On the other hand, if feedback is available, the decoder can acknowledge sufficiency as x^k is sent letter-by-letter. This immediately obtains a *rateless system* (Fig. 3.3). In a static broadcast setting, even feedback is not necessary [33]. Likewise in a storage setting, we can begin with a high rate (e.g. uncompressed), and truncate the compressed sequence x^k if we later discover we can decode at a lower rate. These are all examples of late-binding systems, for which no compression solution currently exists.

However, nothing precludes building a traditional early-binding, zero-error *variable-rate system* if we wish to (and can) bind the data model in the encoder. The presented encoder can simply be augmented with a decoder simulation and choose a rate at which the simulated decoding succeeds with the output matching the original source sequence (Fig. 3.4). Note that the Code block is still model-free, though the encoder is not. Model-code separation is fundamentally not about how to design the encoder, but how to design the coding machinery.

Thus, it is important to emphasize that rate selection in a model-code separation architecture does not present a drawback with respect to joint design. On the contrary, it enables additional flexibility such as late binding for situations where the encoder truly does not have access to the data model, or in more extreme cases, does not have access to the unprocessed source from which to form an entropy estimate (Section 10.2 gives an application).

3.3 Discussion

We discuss the high-level interpretation, performance, and complexity of the proposed architecture. Here we assume the large n regime so finite-length effects can be neglected.

3.3.1 Architecture

The model-code separation architecture achieves the cleavage of the two aspects of lossless compression — modeling and coding — allowing them to be completed in different pipeline stages, or at different times and locations. Save for the requirement to agree on a rate and a random seed, the encoder has no need for any interaction with the decoder, nor need any other prior information about the source at all, even its exact original content (i.e., the source can be pre-processed in any way, as long as it can be modeled). This is very liberating for practical system design.

Furthermore, because the decoder is an inferential decoder embodying a generative source model, it is a replica of the entire process from the (putative) generation of the source from stochastic origins before it reaches the encoder until the point the compressed data leaves the encoder. The separation of coding and modeling is therefore reflected in the decoder structure, whose modularity opens the largest seam at the interface between the source subgraph that models what happens before the encoding process, and the code subgraph that models what happens during the encoding process. The model-free nature of the presented encoder is precisely what allows this to be realized.

Practically, the model-code separation and the modularity of the decoder allow us to separately design or procure the data model and the code, and to swap them at will either during codec design or, in the case of the data model, at any time even long afterwards. As long as the components are able to communicate sensibly with the defined controller interface, as is the case with LDPC codes and PGM's, this is possible. This degree of flexibility is a great asset during system design when more daring trials can be performed, or afterwards when deployed systems can be upgraded to incorporate new understanding of data. It also allows a division of labor where coding theorists and model analysts can apply their greatest expertise and import the farthest-reaching results from their respective fields, without overwhelming concern about compatibility with other parts of the system.

3.3.2 Performance

Practical performance is presented in detail later (Chapters 4 and 5). Here we discuss performance conceptually.

In the random binning scheme (Section 2.1) the encoder produces a bin label (i.e., *hash*) for s^n according to a binning assignment (i.e., *hash function*), while the decoder seeks to identify an s^n sequence that both maps to the correct bin and is statistically typical. Usually, ML decoding is used to identify the most probable s^n among those in the correct bin. This provably achieves entropy rate by standard theory. Thus the system is *architecturally* sound — with optimal components, a powerful controller, and rich interfaces, model-code separation incurs no penalty. Performance loss can however be incurred from various additional system implementation issues.

Suboptimal components — The presented scheme implements a hash function in the form of a randomly chosen linear projection. Linear source codes achieve entropy rate (Section 2.2) with ML. However, if H is chosen from among a suboptimal coding ensemble, then any decoding may be penalized by *bad code loss*. We will return to this loss in Section 5.1. Likewise, while PGM's have full modeling power over distributions and can compute optimal inferences by ML (Section 2.3), modeling with a restricted subset like the pairwise models may result in *model mismatch loss*. We will discuss this in Section 6.3. Finally, components running BP instead of ML when their internal graphical representation is not tree-like may incur *algorithmic loss.*

Suboptimal interfaces — If non-controller components are implemented optimally (e.g., optimal codes and representations, internally running ML), and exchange *joint messages*, for instance, $M_{\mathcal{C}}(\cdot) = c(\cdot, s_{\mathcal{D}} = \mathsf{s}_{\mathcal{D}})$ and $M_{\mathcal{G}'}(\cdot) = p_{\mathsf{s}^n}(\cdot, s_{\mathcal{D}} = \mathsf{s}_{\mathcal{D}})$, then the controller can compute the optimal combined inference

$$\begin{aligned} \hat{s}^n_{\text{ML}} &= \arg\max_{s^n} c(s^n) p_{\mathsf{s}^n}(s^n) \\ &= \arg\max_{s^n} M_{\mathcal{C}}(s_{\mathcal{S}\setminus\mathcal{D}}) M_{\mathcal{G}'}(s_{\mathcal{S}\setminus\mathcal{D}}) \mathbb{1}\{s_D = \mathsf{s}_{\mathcal{D}}\}(s_{\mathcal{D}}) \end{aligned} \tag{3.16}$$

If instead interfaces are constrained to exchange less than the full joint, there is generally *interface loss*, unless it is the case that Eq. 3.16 is satisfied anyway. Such condition exists if (1) the received messages are factors of $c(s^n)$ and $p_{\mathsf{s}^n}(s^n)$ so the exact objective $u(s^n) = c(s^n)p_{\mathsf{s}^n}(s^n)$ is reconstructible by the controller, or somewhat trivially (2) $\mathcal{S}$ is partitionable into three disjoint sets, $\mathcal{D}$, $\mathsf{s}_{\mathcal{C}}$, and $\mathsf{s}_{\mathcal{G}'}$, with $\mathsf{s}_{\mathcal{C}} \perp\!\!\!\perp \mathsf{s}_{\mathcal{G}'} \,\Big|\, \mathcal{D}$, so that

$$\hat{s}^n_{\text{ML}} = (\hat{s}_{\mathcal{C},\text{ML}}, \hat{s}_{\mathcal{G}',\text{ML}}, \mathsf{s}_{\mathcal{D}}) = (\arg\max_{s_{\mathcal{C}}} c(\cdot, s_{\mathcal{D}} = \mathsf{s}_{\mathcal{D}}), \arg\max_{s_{\mathcal{G}'}} p_{\mathsf{s}^n}(\cdot, s_{\mathcal{D}} = \mathsf{s}_{\mathcal{D}}), \mathsf{s}_{\mathcal{D}})$$

which can be verified by graph separation tests on $\mathcal{U}$. Without such guarantees, e.g. when the controller receives *marginal messages* as in the presented decoder, it can at best resort to alternating optimization methods which amounts to running BP.

Suboptimal inference — When running BP in parts or all of the system, the interaction between losses is complex. Compared to the losses incurred by running BP in individual components, there may be additional loss to combining them. There may be intersecting factors in $u(s^n)$, which — if the intersection is over two or more source nodes — adds graph cycles of very short length (4). For sparse $\mathcal{G}'$ and sparse and random $\mathcal{C}$, the occurrence should be negligible, i.e. cycles of all lengths increase in number but not catastrophically [34]. In some cases, combining graphs may help overall, as a sparse code subgraph with hard constraints helps alleviate over-confident messages in a dense source subgraph by presenting its own stronger and more accurate beliefs.

3.3.3 Complexity

For this scheme, the encoding complexity is inherited from linear coding (Section 2.2), which is $\mathcal{O}(n^2)$ for general linear codes and $\mathcal{O}(n)$ for LDPC codes.

Decoding complexity inherits from both coding and modeling results. If the LDPC code has largest row weight ρ, and the PGM has κ unfactorizable cliques and maximum unfactorizable clique size η, then the combined graph $\mathcal{U}$ has BP decoding complexity $\mathcal{O}((k\,|\mathbf{S}|^{\rho} + \kappa\,|\mathbf{S}|^{\eta})I)$, where I is the number of iterations. This compares favorably with ML decoding at $\mathcal{O}(n\,|\mathbf{S}|^{k})$, if $\max\{\rho, \eta, \log\kappa\} \ll k$.

For pairwise models (Section 2.3.3) and LDPC codes, BP decoding has complexity $\mathcal{O}((k\,|\mathbf{S}|^{\rho} + (n + |\mathcal{E}|)\,|\mathbf{S}|^{2})I)$, which is at most $\mathcal{O}(n^2)$ for very dense source subgraphs where complexity is dominated by the number of source subgraph edges, and $\mathcal{O}(n)$ in most other cases especially at lower rates where the first term can have a large constant due to $\rho \geq 1/r$.

3.4 Related ideas

Different aspects of the scheme in this chapter have similarities in prior research. They generally come from several major areas.

3.4.1 Side information problems

In distributed coding, there is interest in using "data-like" side information for compression. As in [35], key frames supply side information for scalability in Wyner-Ziv video coding. As in [36, 37], the secret key is the side information in the decoding of encrypted compressed data. More broadly, works such as [38] and references therein testify to a wide range of compression problems that take advantage of "data-like" side information.

In compressed sensing, "structure-like" side information such as sparsity is used for reconstruction with fewer measurements [39]. In works such as [40], structures additional to sparsity are exploited — in this case, wavelet-domain structures for the sensing of images.

The common thread in these works is this: the encoder decimates the data to a lower rate by a relatively simple process (e.g. projection, sub-sampling); and the decoder, which possesses side information about the source, chooses the correct sample from among the numerous otherwise ambiguous possibilities. Upon this suggestive research milieu born out of applications requiring partial model-code separation (though not stated as such), this work builds in several important ways, noting that:

1. the decimation process can be any code;
2. the side information can be any data model;
3. the choice of code can be completely agnostic to the data model, i.e. model-free.

3.4.2 Graphical algorithms

In terms of algorithms, graphical methods are a well studied and algorithmically rich general inferential toolkit [27, 41], and they have also become useful to compression or signal reconstruction problems, see for instance [42, 43, 44]. More closely relevant, graphical data models are used for direct image reconstruction given some cutset pixels reminiscent of trivial coding [45], and graphical codes show good performance for compressing structureless (i.e. memoryless) sources [24].

These works begin to stake out a role for graphical algorithms in applications involving both data and constraint inference. This work takes a step forward by treating data knowledge and constraint information (i.e. model and code) on an equal footing and showing that graphical algorithms are eminently suitable as a common language for practical general compression.

3.4.3 Dual problems

Finally, architecturally related works include those on the dual problem of coding for channels with memory, where either a combination of LDPC coding and graphical channel modeling is suggested [46, 47], or equalization for inter-symbol interference is viewed as a part of turbo decoding [48].

It is fortuitous that channel coding necessitates late binding and therefore model-code separation, thus giving us an unencumbered look at a pure coding machinery — both encoding and decoding. This work takes such construction as cue and obtains something similar for compression.

4 Compressing Binary Sources

We apply the model-code separation scheme of Chapter 3 to some relatively simple sources over $\mathbf{S} = \mathbf{GF}(2)$ using codes over the same. We call this class of systems `SEP`.

For each type of source, we identify factors from its distribution p_{s^n} to build the graphical representation, compute its entropy rate, and compare compression performance under `SEP` vs. other common codecs in simulated experiments. The excellent performance of `SEP` shows that model-code separation comes at little cost in practice.

For an explanation of the experimental setup and tested systems, refer to the next section.

4.1 Experimental setup

A representative range of parameter values for each source type is selected, and 20 random samples are drawn at each parameter value by, e.g. Gibbs sampling (Section 2.3.2.1). The average rate performance (output bits per input bit) over the samples is reported for the following compressors:[1]

- `SEP-prot`: An instance of the proposed system. An off-the-shelf library of quasi-regular-($\bar{\lambda} = 3, \bar{\rho}$) binary LDPC codes is used.[2] The doping rate r_{dope} is fixed and noted. Rate performance denotes the minimal total rate $r = r^*_{\text{code}} + r_{\text{dope}}$, where

$$r^*_{\text{code}} \triangleq \min_{r} r_{\text{code}} \tag{4.1}$$

  for which decoding converges (in this case, within 150 iterations) to the correct result.

- `ARITH`: Standard arithmetic coding [49] for coding biased i.i.d. symbol streams, with symbol probabilities supplied. (This compressor is excluded for sources with memory.)

- `GZIP`: A Lempel-Ziv class universal compressor [50, 51] that learns a dictionary from serial streams. Input data is provided to it after flattening to a bit-stream. Output

[1] `ARITH` is implemented in MATLAB's Communications System Toolbox (v5.5) as `arithenco`; `GZIP` is implemented in standard GNU distributions as `gzip` (v1.5); `CTW` is found at `http://www.ele.tue.nl/ctw/download.html` (v0.1); `JBIG2` is found at `http://github.com/agl/jbig2enc` (v0.28).

[2] "Quasi-regular" denotes degree distributions taking only two adjacent values, viz., checksum degrees (row weights of H) are $\lfloor \bar{\rho} \rfloor$ and $\lceil \bar{\rho} \rceil$.

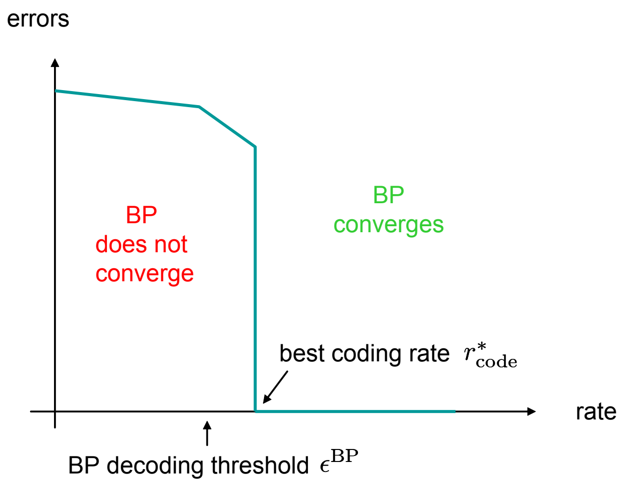

errors
BP does not converge
BP converges
best coding rate r^*_{code}
rate
BP decoding threshold ϵ^{BP}

Figure 4.1: The BP decoding threshold ϵ^{BP} is a lower rate that more accurately describes the useful rate of a particular code under BP.

length is the compressed file size, less the compressed file size of a zero-length file to account for headers.

- `CTW`: The context-tree weighting universal compressor [52] learns a Markovian tree model from serial streams. Just as for `GZIP`, input data is provided to it after flattening.

- `JBIG2`: This state-of-the-art bi-level image compressor (ITU-T T.88; 2000) [53] is based on 2D context dictionaries. We operate the encoder in lossless mode. The output length is the file size of raw-stream compression, less the compressed file size of a 1-pixel image to account for headers. (This compressor is excluded for non-2D sources.)

Universal compressors are included for reference, in particular for when there are no better data-matched codecs. The meaning of comparisons with them is briefly broached in Section 4.1.2, but a more detailed discussion of universality for model-code separation awaits in Chapter 6.

4.1.1 Threshold rate

In `SEP-prot`, for each minimal rate LDPC code found, we can identify its *BP decoding threshold*, ϵ^{BP} (Fig. 4.1). This threshold rate serves as a more accurate proxy for the "utilizable" rate of that code than its coding rate r_{code}, in that the gap between r_{code} and ϵ^{BP} is not primarily an architectural loss, but that associated with code selection and decoding method. Better code selection such as through degree-distribution optimization or other methods can be expected to close the gap (see Section 5.1).

The *total threshold rates* $r^* \triangleq \epsilon^{\text{BP}} + r_{\text{dope}}$ are reported under `SEP-thresh`, which can be viewed as another (idealized) system instance where codes are well chosen. These are the more important quantities when we speak of `SEP` performance.

4.1.2 Expected performance

We expect compressors that correctly assume a greater amount of data model to achieve better compression. `SEP` assumes the exact data model, so it is at an inherent advantage. However, the other compressors, even universal ones, also make some assumptions to be effective at finite data lengths.[3] `JBIG2` assumes recurring 2D symbol motifs, `GZIP` assumes recurring 1D symbol motifs, `CTW` assumes tree-structured symbol occurrence frequencies, and `ARITH` assumes fixed symbol occurrence frequencies. Some assumptions clearly subsume others, thus we expect for sources that demand the stronger of the assumptions, a rate performance ordering like

$$\texttt{SEP} \prec \texttt{JBIG2} \prec \texttt{GZIP}$$

$$\texttt{SEP} \prec \texttt{CTW} \prec \texttt{ARITH}$$

where "smaller" under$\prec$ refers to lower or better rate. If we distinguish the `SEP` instances and also include the ultimate lower bound in the entropy rate, we add

$$\mathbb{H}(\mathsf{s}) \prec \texttt{SEP-thresh} \prec \texttt{SEP-prot}$$

For less demanding sources, multiple systems may converge toward similar performance because they sufficiently capture the same aspect of the data model, e.g., 1D Markovianity; and the same is expected as universal compressors pick up more of the data model from learning over longer data lengths.

These are borne out in the results, which follow.

4.2 Bernoulli i.i.d. sources

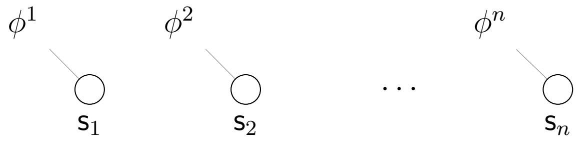


Figure 4.2: Source subgraph for $\mathbf{Bern}(p)$, with $\phi^i = [1-p;p]$.

A Bernoulli i.i.d source $\mathsf{s} \sim \mathbf{Bern}(p)$ has distribution [4]

$$p_{\mathsf{s}^n}(s^n) = \prod_{i=1}^{n} [1-p;p](s_i) \tag{4.2}$$

[3]More precisely, universal compressors operating at data lengths (or effective data lengths, due to state memory) where AEP is not effective depend on the peculiarities of their construction to determine the range of data models they are capable of capturing.

[4]$[\alpha;\beta](x) \triangleq \alpha \mathbb{1}_{\{x=0\}}(x) + \beta \mathbb{1}_{\{x=1\}}(x)$.

and entropy rate [5]

$$\mathbb{H}(\mathsf{s}) = h(p) \tag{4.3}$$

The source is trivially pairwise, has no edges, and only has ϕ^i factors, known as *node potentials*.

Compressing a $\mathbf{Bern}(p)$ source is dual to the problem of channel coding for the binary symmetric channel (BSC) with cross-over probability p. Indeed, the inferential decoder for `SEP` has graphical structure identical to the LDPC decoder for the BSC:

- (Decoder A.) The $\mathbf{Bern}(p)$ decoder solves for the source data sequence $s \sim \mathbf{Bern}(p)$ satisfying $Hs = x$ by setting

  $$\phi^i_{\mathrm{A}} \quad \Leftarrow \quad [1-p; p]$$

  and running BP in accordance with Section 3.2.3.3.

- (Decoder B.) The BSC decoder solves for the codeword c satisfying $Hc = 0$ from the received word $y = c + w$, $w \sim \mathbf{Bern}(p)$, by setting

  $$\phi^i_{\mathrm{B}} \Leftarrow \begin{cases} [1-p; p] & \text{if } y_i = 0 \\ [p; 1-p] & \text{if } y_i = 1 \end{cases}$$

  and running BP as if $x = 0$.

Due to the nature of the factors f_a in the code subgraph, where the effect of inverting any input message can be nullified by the inversion of the parity value x_a (Section 4.6), Decoder B can be transformed into Decoder A, by a series of steps inverting each ϕ^i_{B} for which $y_i = 1$ and simultaneously inverting its attached x_a. This gives an equivalent decoder for w, in exactly the same form as Decoder A.

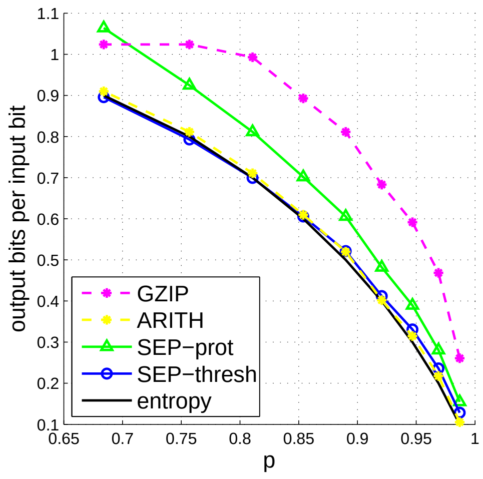
Given that LDPC decoding for the BSC is well understood, we appeal to those results in addition to what follows for the behavior of compressing $\mathbf{Bern}(p)$ sources using `SEP`.

4.2.1 Results

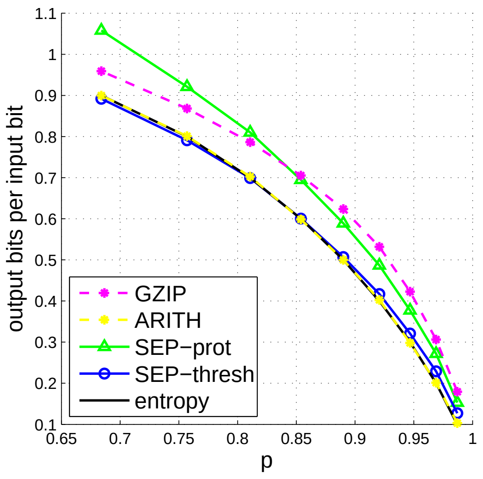
In Fig. 4.3, we observe that `SEP-thresh` can reach the entropy rate of $\mathbf{Bern}(p)$, even at fairly short data lengths. The coding rate itself (`SEP-prot`) is within 16-20% of the threshold rate of the code (`SEP-thresh`), and better in absolute terms at lower rates than at higher rates. These characteristics are well known from channel coding analogs, and good codes selected already for channel coding (cf. Section 2.2) can indeed be productively applied to this source.

Of the other systems, `ARITH` performs better than `GZIP` because it captures the full data model for this source without learning.

[5] $h(x) \triangleq -x \log x - (1-x)\log(1-x)$.



(a) $n = 1000$



(b) $n = 10000$

Figure 4.3: Compression performance for the $\mathbf{Bern}(p)$ source family over a range of p values. $r_{\text{dope}} = 0$.

4.3 Binary Markov sources

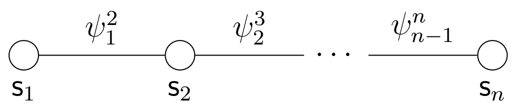


Figure 4.4: Source subgraph for $\mathbf{Markov}(q)$, with $\psi_i^j = \begin{bmatrix} q & 1-q \\ 1-q & q \end{bmatrix}$.

Markov sources are perhaps the simplest sources with memory. A binary homogeneous symmetric Markov chain $\mathsf{s}^n \sim \mathbf{Markov}(q)$ has distribution [6]

$$p_{\mathsf{s}^n}(s^n) = \frac{1}{2}\prod_{i=2}^{n} \begin{bmatrix} q & 1-q \\ 1-q & q \end{bmatrix} (s_{i-1}, s_i) \tag{4.4}$$

and entropy rate

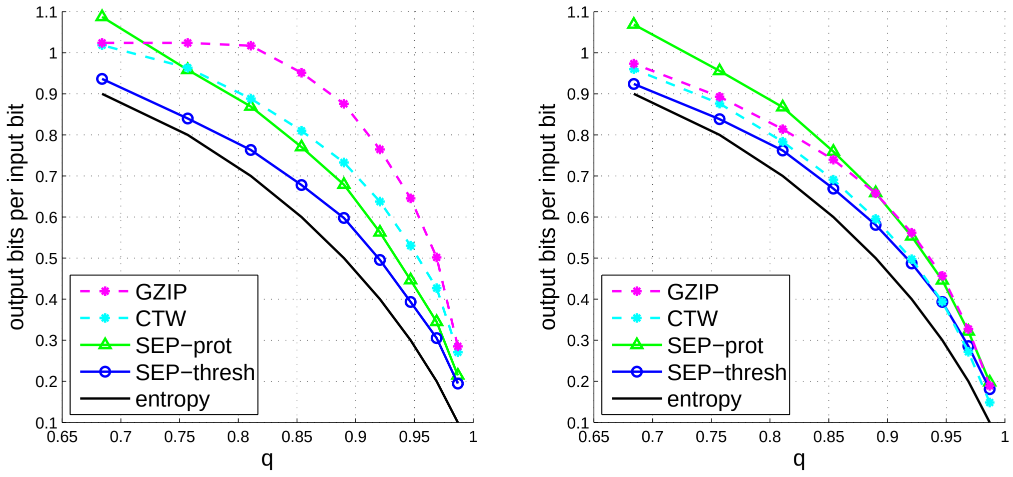
$$\mathbb{H}(\mathsf{s}) = h(q) \tag{4.5}$$

This source is also pairwise, has $\mathcal{O}(n)$ edges and only ψ_i^j factors on them, called *edge potentials*.

Compressing a larger-alphabet version of this source is the subject of Section 8.1.

[6] $\begin{bmatrix} \alpha & \gamma \\ \beta & \delta \end{bmatrix}(x, y) \triangleq [\alpha;\beta](y)\mathbb{1}_{\{x=0\}}(x) + [\gamma;\delta](y)\mathbb{1}_{\{x=1\}}(x)$.

4.3.1 Results



(a) $n = 1000$ (b) $n = 10000$

Figure 4.5: Compression performance for the $\textbf{Markov}(q)$ source family over a range of q values. $r_{\text{dope}} = 0.1$.

Fig. 4.5 shows that $\textbf{Markov}(q)$ has more difficulty to reach entropy than i.i.d. sources. For one, decoding of $\textbf{Markov}(q)$ requires non-trivial initialization such as doping, as there are no biased node potentials to begin the iterative process. In this case, $\sim 10\%$ doping gives good results. The doping rate is the primary residual that `SEP-thresh` maintains over the entropy rate at lower rates, whereas at higher rates doping acts much more in concert with the code.

Of the other systems, `CTW` is able to capture the Markovian data structure better than `GZIP` at the same data length, but only reaches performance similar to `SEP-thresh` at longer data lengths.

4.4 Binary Ising model

The Ising model is a 2D extension of the Markov chain, with neighbor affinity properties. It is common in vision research as an image model. The homogeneous Ising model $\mathsf{s}^{h\times w} \sim \textbf{Ising}(p,q)$ is defined over the $h \times w$ lattice graph $\mathcal{G} = (\mathcal{S}, \mathcal{E})$ by

$$\begin{aligned} p_{\mathsf{s}^n}(s^n) &= \frac{1}{\mathcal{Z}} \prod_{i\in\mathcal{S}} \phi(s_i) \prod_{(i,j)\in\mathcal{E}} \psi(s_i, s_j) \\ &= \frac{1}{\mathcal{Z}} \prod_{i\in\mathcal{S}} [1-p; p](s_i) \prod_{(i,j)\in\mathcal{E}} \begin{bmatrix} q & 1-q \\ 1-q & q \end{bmatrix} (s_i, s_j) \end{aligned} \tag{4.6}$$

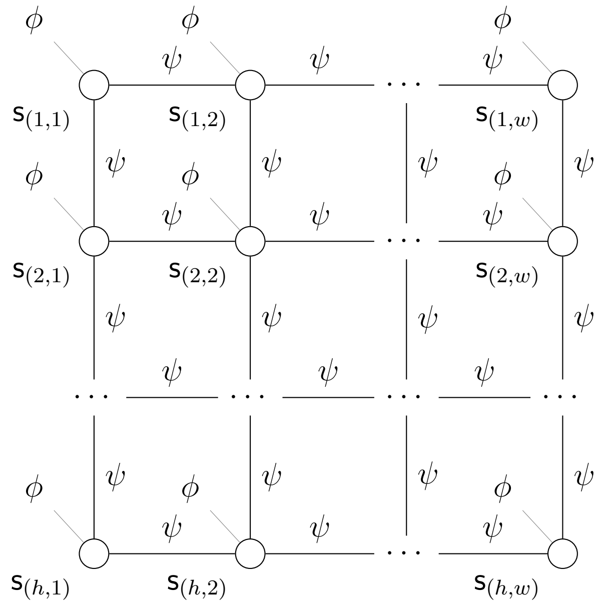


Figure 4.6: Source subgraph for **Ising**(p, q), with $\phi = [1 - p; p]$, $\psi = \begin{bmatrix} q & 1-q \\ 1-q & q \end{bmatrix}$.

The source is again pairwise, having $n = hw$ nodes (resp. node potentials), and an edge between every neighbor pair in the four cardinal directions for a total of $2hw - (h + w)$ edges (resp. edge potentials), which scales as $\mathcal{O}(n)$.

In the following, we are only interested in the *symmetric Ising model* with $p = 1/2$. Some samples are shown in Fig. 4.7.

There is a phase transition around the parameter threshold $q = \sqrt{2}/2$ (exact for n large), such that long-range correlations are low (resp. high) below (resp. above) the threshold. A significant amount of compression can be achieved above the threshold, where the source begins to resemble shape-like natural images.

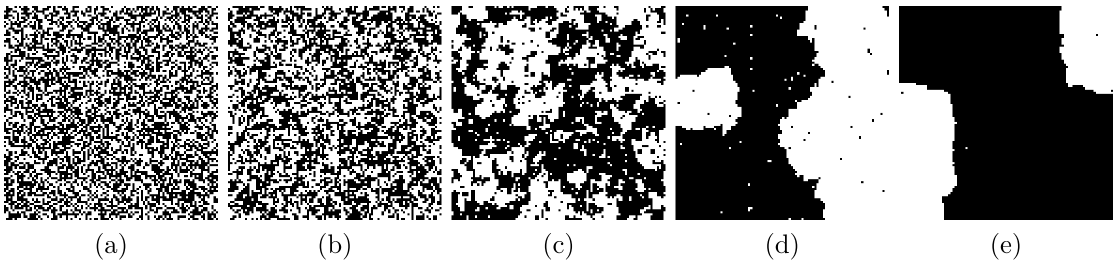


Figure 4.7: 100×100 Gibbs sampled images according to Eq. 4.6 of **Ising**$(\frac{1}{2}, q)$. From left to right, $q = 0.5, 0.6, 0.7, 0.8, 0.9$.

4.4.1 Entropy rate

In order to have a lower bound on the rate, we derive an entropy rate for the finite-size and infinite-size Ising models.

Putting $p = 1/2$, we have two calculable entropy rates for the symmetric Ising model. Letting $p(s)$ denote the distribution of an arbitrarily sized model, we note that

$$\begin{aligned}
p(s) &\propto \prod_{(i,j)\in\mathcal{E}} \psi(s_i, s_j) \\
&= q^{\#\mathrm{SE}(s)}(1-q)^{\#\mathrm{DE}(s)} \\
&\propto \exp\left\{\#\mathrm{DE}(s)\log\left(\frac{1-q}{q}\right)\right\} \\
&= \exp\{-2\theta\#\mathrm{DE}(s)\} \\
&\triangleq \pi(s;\theta)
\end{aligned}$$

where $\theta = \tanh^{-1}(2q-1)$ is a parameter of $p(s)$ when written in exponential family form, and $\#\mathrm{SE}(s)$ (resp. $\#\mathrm{DE}(s)$) is the number of edges in $\mathcal{E}$ that connect two nodes with the same value (resp. different values) for the sample s.

Now, suppose $\log \mathcal{Z} = \log \sum_s \pi(s;\theta)$ for proper normalization, then,

$$\begin{aligned}
\frac{\partial \log \mathcal{Z}}{\partial\theta} &= \frac{1}{\mathcal{Z}}\frac{\partial \mathcal{Z}}{\partial\theta} = \frac{1}{\mathcal{Z}}\sum_s \frac{\partial \pi(s;\theta)}{\partial\theta} \\
&= \frac{1}{\mathcal{Z}}\sum_s \exp\{-2\theta\#\mathrm{DE}(s)\} \times (-2\#\mathrm{DE}(s)) \\
&= \sum_s p(s) \times (-2\#\mathrm{DE}(s)) \\
&= \sum_s p(s) \times \frac{1}{\theta}\log\pi(s;\theta)
\end{aligned}$$

Thus the *finite entropy rate* for a particular sized model can be extracted from $\log \mathcal{Z}$ as

$$\begin{aligned}
\mathcal{H}_{h,w} &\triangleq \frac{1}{n}\mathbb{H}(\mathsf{s}^{h\times w}) \\
&= -\frac{1}{n}\sum_s p(s) \times \{\log\pi(s;\theta) - \log\mathcal{Z}\} \\
&= -\theta\frac{\partial\{\log\mathcal{Z}/n\}}{\partial\theta} + \{\log\mathcal{Z}/n\} \qquad (4.7)
\end{aligned}$$

Empirically, the function $\partial\{\log \mathcal{Z}/n\}/\partial\theta = \mathbb{E}\{-2\#\mathrm{DE}(s)\}/n$ (and therefore $\log \mathcal{Z}$ and $\mathcal{H}_{h,w}$) can be approximated from samples of s by substituting an average for the expectation. Finite lattices have higher entropy (for $q \neq 1/2$) due to boundary values.

For $h, w \to \infty$, we quote a result from statistical physics [54, 55, 56]:

$$\log \mathcal{Z}/n = \log \frac{2}{1-\theta^2} + \frac{1}{2(2\pi)^2} \int_{-\pi}^{\pi}\int_{-\pi}^{\pi} dq_x dq_y \log\left\{(1+\theta^2)^2 - 2\theta(1-\theta^2)(\cos q_x + \cos q_y)\right\}$$

from which we get the usual (asymptotic) entropy rate $\mathcal{H}_\infty = \mathbb{H}(\mathsf{s})$ from Eq. 4.7. Both entropy bounds will be used as performance references.

4.4.2 Results

For this source, `JBIG2` is given the data as a bitmap image, with each node s_i representing a pixel. For `GZIP` and `CTW`, data is given in scanline order.

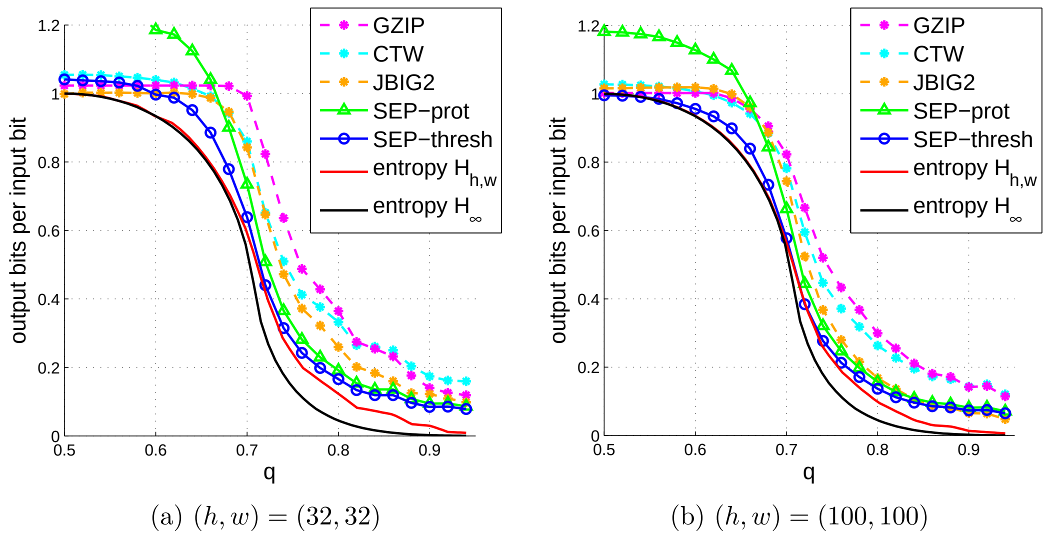


Figure 4.8: Compression performance for the **Ising**(p, q) source family for $p = \frac{1}{2}$ and a range of q values. $r_{\text{dope}} = 0.04$.

Since $p = 1/2$, a non-trivial initialization is required just like for **Markov**(q). Here, $\sim 4\%$ doping (lower than the **Markov**(q) case) works well. Referring to Fig. 4.8, `SEP-thresh` beats `GZIP` and `CTW`, rivals `JBIG2`, and is close to the finite entropy rate $\mathcal{H}_{h,w}$ over most values of q. Particularly in the low-rate regime with shape-like images (q nearer to 1), the coding-to-threshold rate gap is small enough that coding choices are of lesser concern. However, the residual attributed to the doping rate may still benefit from optimizations.

Of the other systems, `JBIG2` only begins to approach `SEP` performance at longer data lengths, while `GZIP` and `CTW` have substantially similar performance, being able to capture only the 1D features of **Ising**(p, q) well, and less able to exploit the 2D correlations as `JBIG2` does. Furthermore, traditional context-based compressors always encode causally, thus they refer to a portion of the data already encoded, thereby losing the performance attributable to the true unrestricted context, which `SEP` naturally has.

4.5 Summary

`SEP` shows good performance over several source families with distinctive model structures and a range of parameter values. At medium to lower rates, the unoptimized example of `SEP-prot` nearly suffices, but indeed, it is also important that `SEP-thresh` is close to the entropy rate in many cases, suggesting such nearly optimal performance is realizable with coding and system optimizations. For example, starting from the basic `SEP-prot` design, additional gains can be made on the coding-to-threshold rate gap at higher rates and the doping residual at lower rates.

Beyond absolute performance merits, `SEP` also benefits from the model-code separation architecture in the way in which it can embed and apply a good-faith data model at any data length. This is more important especially at shorter data lengths and for data that are ill represented by causal models.

4.6 Appendix: Parity lemma*

Definition 4.1. A binary message $m = [1-p; p]$ is called inverted to $\bar{m}$ if $\bar{m} = [p; 1-p]$. A 0-1 tensor f (perhaps representing a factor node function) is called inverted to a tensor $\bar{f}$ if $f + \bar{f} = 1$, the all-1 tensor.

Lemma 4.2. *Given a $\mathcal{C}$ component as in Table 3.2, with factor nodes f_a as defined in Section 3.2.3.1, an inversion of the external message input $[M^{i\leftarrow}]_{\mathcal{C}}$ at s_i has no effect on any factor node output (and therefore any external message output), if all f_a to which s_i is connected are inverted.*

Proof. Note that the only messages that are possibly affected in one iteration of the $\mathcal{C}$ component are, $m^{i\to a}$ for all f_a to which s_i is attached, and $m^{k\leftarrow a}$ for all s_k to which each such f_a is connected in turn. Applying the indicated inversions on the affected messages:

$$\begin{aligned} m^{i\leftarrow\sim a}\left[\bar{M}^{i\leftarrow}\right]_{\mathcal{C}} &\quad\Rightarrow\quad \bar{m}^{i\to a} \\ \bar{f}^{k}_{\sim k}\bar{m}^{i\to a}m^{\sim k\backslash i\to a} = f^{k}_{\sim k}m^{\sim k\to a} &\quad\Rightarrow\quad m^{k\leftarrow a} \end{aligned} \tag{4.8}$$

where the equality of Eq. 4.8 is because

$$\bar{f}^{k}_{\sim k}\bar{m}^{i\to a} = \bar{f}^{k}_{\sim k}\left[\begin{array}{cc} 0 & 1 \\ 1 & 0 \end{array}\right]^{i}_{i} m^{i\to a}$$

and $\left[\begin{array}{cc} 0 & 1 \\ 1 & 0 \end{array}\right]^{i}_{i}$ is the inversion operator for either $\bar{f}^{k}_{\sim k}$ or $m^{i\to a}$. □

Note that inverting f_a is equivalent to inverting x_a.

5 Coding Details

Practical implementation of SEP requires designers to make tuning choices in the system. An important set of tuning values relate to the code component and its interaction with the rest of the decoder. In this chapter, we discuss details regarding selecting the code ensemble $\mathscr{H}(n,k)$ and choosing the doping rate r_{dope}. These choices ultimately are to be evaluated for compression performance. Therefore, we also identify the summary quantities in decoding that are important for identifying how the decoder behaves — whether it converges or not, how fast, and how to know. Finally we propose some analytical tools drawn from traditional LDPC channel coding analysis, but adapted here for the compression problem.

5.1 Code selection

The selection of a code ensemble $\mathscr{H}(n,k)$ bears strongly on the performance of an LDPC coding system. This problem has been examined for channel coding, and how the results are relevant for compression is the subject of this section. The basic idea is to derive a relationship between a performance metric on the coding system and some parameters defining a code ensemble, after which we can optimize for the code ensemble.

5.1.1 Degree distribution and threshold

One result is that code ensembles parameterized by their *degree distributions* are easily related to certain performance metrics called *decoding thresholds*. We alluded to a BP decoding threshold ϵ^{BP} in Section 4.1.1 and throughout Chapter 4. We define it more precisely now in relation to a channel coding analog.

5.1.1.1 Degree distribution*

Definition 5.1. As in ([41], Chapter 3), a binary linear code with coding matrix $H_{k\times n}$ having Λ_i columns of weight i and P_j rows of weight j, has *degree distributions* encoded by the generating functions $\Lambda(x) \triangleq \sum_i \Lambda_i x^i$ and $P(x) \triangleq \sum_j P_j x^j$. It has *normalized degree distributions* $L(x) \triangleq \Lambda(x)/\Lambda(1)$ and $R(x) \triangleq P(x)/P(1)$ and *normalized edge distributions* $\lambda(x) \triangleq \Lambda'(x)/\Lambda'(1)$ and $\rho(x) \triangleq P'(x)/P'(1)$.[1]

[1] $A'(x) \triangleq \frac{d}{dx}A(x)$.

Example 5.2. A quasi-regular-($\bar{\lambda} = 3, \bar{\rho}$) code as used in `SEP-prot` has degree distributions $\Lambda(x) = x^3$ and $P(x) = (n - \lfloor n \operatorname{frac}(\bar{\rho}) \rceil) x^{\lfloor \bar{\rho} \rfloor} + \lfloor n \operatorname{frac}(\bar{\rho}) \rceil \, x^{\lceil \bar{\rho} \rceil}$.[2]

5.1.1.2 Channel coding thresholds*

Channel coding thresholds are minimum channel qualities at which an ensemble $\mathscr{H}(n, k)$ can be successfully decoded by various decoders. The first useful threshold is the *erasure threshold*, i.e. the largest fraction of erasures a binary erasure channel (BEC) can have to permit successful decoding of an LDPC code with a certain degree distribution. The higher this number, the better the code/decoder pair is for the BEC.

Fact 5.3. *Given a code with $\lambda(x)$ and $\rho(x)$, let $f(\epsilon, x) = \epsilon\lambda(1 - \rho(1 - x))$. The BP erasure threshold for the BEC,*

$$\epsilon^{\mathrm{BP}} \triangleq \sup\{\epsilon : f(\epsilon, x) - x < 0, \ \forall x \in (0, 1]\} \tag{5.1}$$

is the largest fraction of erasures that can be corrected by BP decoding with this code.

Similarly, the same code under ML decoding also has a performance limit.

Fact 5.4. *Given a code with $\lambda(x)$, $\rho(x)$, $L(x)$, and $R(x)$, let $\epsilon(x) = x/(\lambda(1 - \rho(1 - x)))$, and let $x^* \in (0, 1]$ either solve*

$$0 = \epsilon(x)L(1 - \rho(1 - x)) + L'(1)x\rho(1 - x) - \frac{L'(1)}{R'(1)}(1 - R(1 - x))$$

or be set to $x^* = 1$, then the ML erasure threshold for the BEC,

$$\epsilon^{\mathrm{ML}} \triangleq \epsilon(x^*) \tag{5.2}$$

is the largest fraction of erasures that can be corrected by ML decoding with this code.

The largest erasure threshold for any ensemble $\mathscr{H}(n, k)$ of its rate can only be k/n because the information rate of the code is set at $I_{\text{code}} = 1 - k/n$. Thus for the BEC, we have generally $\epsilon^{\mathrm{BP}} \leq \epsilon^{\mathrm{ML}} \leq 1 - I_{\text{code}} = k/n$. Applying the BEC capacity relation $C = 1 - \epsilon$ gives a corresponding sequence of *capacity thresholds*, $C^{\mathrm{BP}} \geq C^{\mathrm{ML}} \geq I_{\text{code}} = 1 - k/n$. This second sequence is applicable also to other channels. It means that though we expect to code at an information rate I_{code} approaching channel capacity from below, the actual situation may be that for this ensemble the information rate needs to be lowered by $C^{\mathrm{ML}} - I_{\text{code}}$ even for the best decoder, and by $C^{\mathrm{BP}} - I_{\text{code}}$ with a suboptimal BP decoder.

5.1.1.3 Source coding thresholds

Although thresholds are first developed for channel coding, we can use them for source coding. Appealing to the duality of channel coding for a channel with additive-noise p_{s^n}

[2] $\lfloor x \rceil$ is the rounding of x; $\operatorname{frac}(x)$ is the fractional part of x.

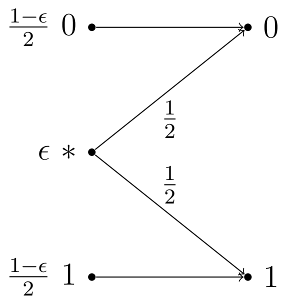


Figure 5.1: A binary doping source $\mathbf{BDS}(\epsilon)$ modeled as being generated from an observed pre-source on the left. The pre-source takes the entropic state $(*)$ with probability ϵ, and the doping states (0, 1) with overall probability $1-\epsilon$.

and source coding for a source with distribution p_{s^n}, the capacity thresholds $C^{\text{BP}} \geq C^{\text{ML}} \geq I_{\text{code}} = 1 - k/n$ correspond exactly to the *entropy thresholds* $h^{\text{BP}} \leq h^{\text{ML}} \leq r_{\text{code}} = k/n$, which are the maximum source entropies allowed for compression with an ensemble $\mathscr{H}(n,k)$ to succeed under various decoders. That is, a source code may have a rate r_{code}, but the entropy rates that are decodable may be lower. $r_{\text{code}} - h^{\text{ML}}$ measures the gap due to the code, while $r_{\text{code}} - h^{\text{BP}}$ measures the gap due to the code and BP decoding. It is in this sense that entropy thresholds are the "utilizable" rates of a source code.

Generally, there is a slightly different set of thresholds for each different source (as for channels). However, we would like to have one metric for a code/decoder pair. Just like the BEC threshold is often used to quote LDPC code performance, so consider a special source that is dual to the BEC — call it the *binary doping source (BDS)* (Fig. 5.1). In a BDS $\mathsf{s}^n \sim \mathbf{BDS}(\epsilon)$, entropy is confined within some subset ϵ of bits which are each equiprobably 0 or 1; the remaining $1-\epsilon$ bits are provided by an oracle (i.e. as if doped) without costing rate. `SEP` decoding for the $\mathbf{BDS}(\epsilon)$ then corresponds to LDPC channel decoding for the $\mathbf{BEC}(\epsilon)$, and for the same codes, the erasure thresholds ϵ^{BP}, ϵ^{ML}, etc., are also the entropy thresholds for the BDS. It is one of these entropy thresholds (ϵ^{BP}) that we use to define `SEP-thresh` performance in Section 4.1.1:

Definition 5.5. Given a code with degree distribution parameters $\lambda(x)$, $\rho(x)$, $L(x)$, $R(x)$, such that ϵ^{BP} is its BP erasure threshold (Eq. 5.1), then let ϵ^{BP} also be the *BP decoding (entropy) threshold* for compression with the code. Similarly, let ϵ^{ML} (Eq. 5.1) be the *ML decoding (entropy) threshold.*

The BDS entropy thresholds are easy to calculate and useful due to certain extremal properties (see e.g. [41], Theorem 4.141) that make them generally conservative estimates for code/decoder quality. That is to say, because the BDS is an "easy" source to compress, the excess rates on it reflect truths about the coding machinery that we can hope to make improvements on, while additional excess rates may be something source-specific — we could make improvements for each source, too, but it would be far less practical.

5.1.2 Coding improvements

The meaning of entropy decoding thresholds for coding improvements is twofold. First we can use h^{ML} or its proxy ϵ^{ML} to screen for a good code whose ML decoding performance coincides with the coding rate r_{code} (generally it means ensuring H has full rank). For LDPC ensembles this is not so difficult at sufficient data length. The greater performance gap is the coding-to-threshold gap $r_{\mathrm{code}} - h^{\mathrm{BP}}$ (or $r_{\mathrm{code}} - \epsilon^{\mathrm{BP}}$) due to suboptimal iterative decoding. No fewer than three approaches to close this gap are found in literature:

- Degree-distribution optimization: Degree distributions $\Lambda(x)$ and $P(x)$ can be adjusted through the function $f(\epsilon, x)$ to raise the BP threshold [57].
- Spatial coupling: Spatially coupled LDPC codes can have the same BP and ML thresholds [58, 18, 26, 59].
- Decoder modification: The BP decoding algorithm itself can be modified in various ways, usually with some complexity tradeoff, to bypass decoding difficulties at bottlenecks [60, 61].

Other approaches, e.g. expander graphs and related constructions, with additional graph-topological properties [44], may translate from compressed sensing problems to general compression.

The specifics of these approaches decouple with respect to the compression application, so they are less relevant to the present discussion. Suffice it to say that by identifying the threshold phenomenon of source coding with that of channel coding, we allow ourselves to gain from the still developing field of modern coding.

5.2 Doping

In Section 3.2.4, we introduced doping as part of the `SEP` decoder. Here we examine exactly what its functions are and what range of rates may be appropriate for r_{dope}.

There are two functions for doping: to initialize the decoding process, and to improve the decoding of certain sources.

5.2.1 Initialization function

Unless the source model provides biased initial marginals, such as in $\mathbf{Bern}(p)$ or compressible i.i.d. sources in general, all initial messages in the `SEP` decoder are null and BP has no dynamics. It is certainly the case that a random initialization of the messages can generate dynamics, and for some well behaved sources along with sufficient rate, this may proceed to successful decoding. However, such aggressive initialization with unfounded beliefs generally puts the decoding in one of many incorrect search spaces from which it does not easily find an escape via iterative means. Thus in `SEP`, doping is the preferred and chosen method to practically begin the decoding process in the neighborhood of the solution.

5.2.2 Coding function

That doping can make inferential decoding "easier" has been recognized in various guises, from the "degree-1 bits" of [62] to the "reference variables" of [63]. In compression too, explicit use of doping, sometimes in large quantities, are found in [24, 37], while implicit uncoded transmission of partial data upon which other correlative structures are built [45, 35] is similar to doping in function if not in conception.

As acknowledged previously, producing doping symbols in the encoder is equivalent to appending unit-weight checksum rows to the coding matrix to form $H^{\triangle}$, thus its choices are inherently part of coding optimization. In this form, unit-weight checksums also play a role within degree distribution and spatial coupling optimization methods.[3]

Since doping improves convergence but increases rate, its overall effect is a priori unclear. Absent a unified theory of code construction and optimization, we are left to consider doping separately and empirically observe its effects on coding various sources. Unsurprisingly, different amounts of doping are good for different situations.

Fig. 5.2 shows two examples from the sources in Chapter 4. Displayed are the minimal total coding rates $r = r_{\text{code}} + r_{\text{dope}}$ for `SEP-prot`, at different r_{dope} values. With the given code ensemble of `SEP-prot`, there appears to be an r^*_{dope} at which performance is best across the range of source parameters. However, the best doping rates are different for different sources and it appears that the 1D **Markov**(q) sources require relatively more doping ($\sim 10\%$) than the 2D **Ising**(p, q) sources ($\sim 4\%$). We conjecture that r^*_{dope} is model structure related, scaling inversely with the number of neighbor nodes (i.e. the source "dimension") within some radius of the source subgraph.

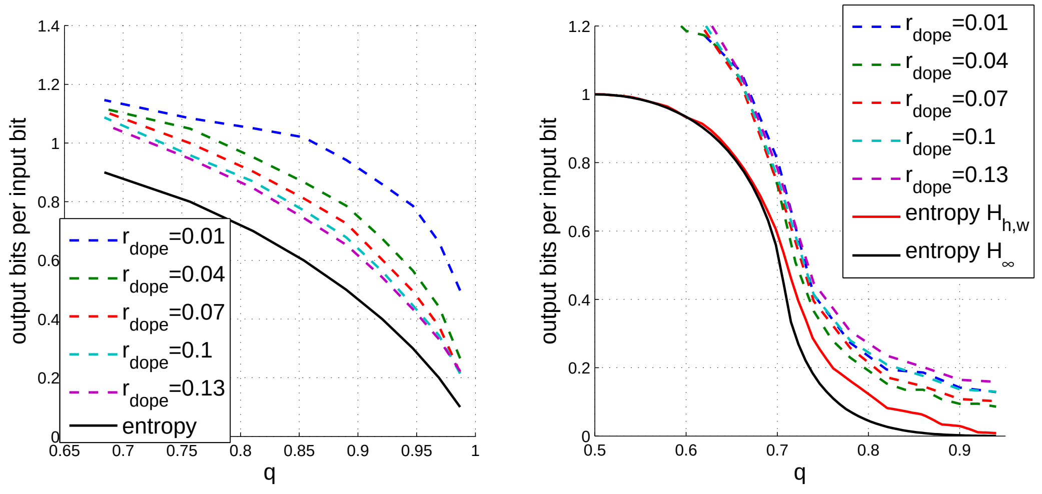
5.3 Decoding mechanics

Another way to understand the effect of coding is to observe the decoding process at finer grain. This helps us to get deeper insight about what makes a good or bad code for compression. In this section, we present two internal facets of the decoding process: the rate characteristics and the convergence dynamics.

5.3.1 Rate characteristics

By rate characteristics we mean the decoder behavior around the critical coding rate r^*_{code} separating successful and unsuccessful decoding (Eq. 4.1). Recall that in the `SEP-prot` results of Chapter 4, for each data sample, we conducted a number of trial BP decodings using LDPC codes at different coding rates r_{code}. Further recall that BP terminates in one of three ways: (1) convergence to the correct result with all constraints satisfied, (2) non-convergence

[3]Incidentally, $H^{\triangle}$ can be simplified after rows for doping have been added, by removing the other 1's in the same column as each doping symbol — say, for each doped s_i, those in the λ rows corresponding to the factors $f_{a_1}, f_{a_2}, ..., f_{a_\lambda} \in \mathcal{N}_i^{\mathcal{C}}$ — and pre-computing adjustments to $x_{a_1}, x_{a_2}, ..., x_{a_\lambda}$ by means of Lemma 4.2 (Section 4.6).



(a) **Markov**(q), $n = 1000$ (b) **Ising**$(\frac{1}{2}, q)$, $(h, w) = (32, 32)$

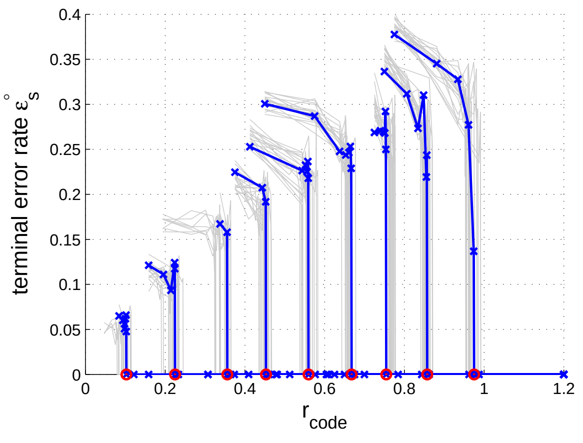
Figure 5.2: `SEP-prot` compression performance at different doping rates. No one doping rate is good for all situations.

with messages converged but some constraints unsatisfied, or (3) non-convergence with messages unconverged and reaching the maximum allowed iteration count. Upon termination, we always obtain a triplet $(r_{\text{code}}, \varepsilon_s^\circ, I)$, where r_{code} is the coding rate, ε_s° is the terminal (residual) error rate

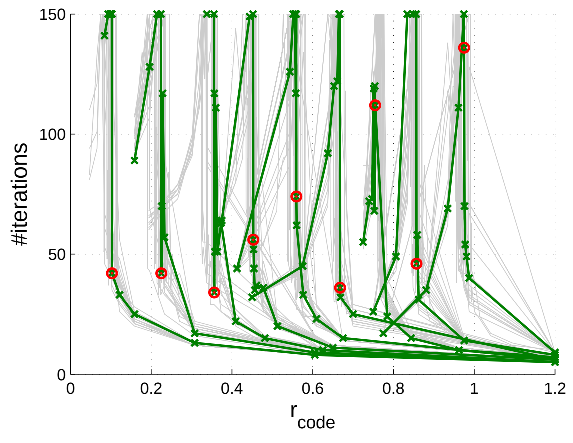
$$\varepsilon_s^\circ \triangleq \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{\hat{\mathrm{s}}_i \neq \mathrm{s}_i\} = \frac{1}{n}\left\|\hat{\mathrm{s}}^n - \mathrm{s}^n\right\|_0 \tag{5.3}$$

and I is the number of iterations elapsed. Since only the first case reports $\varepsilon_s^\circ = 0$, it is the only case termed "successful decoding." The minimal coding rate for which there is successful decoding was reported previously. That corresponds to the *critical coding point* $(r_{\text{code}}, \varepsilon_s^\circ, I) = (\mathrm{r}^*_{\text{code}}, 0, \mathrm{I}^*)$. However, it is worthwhile to look at the other points.

Fig. 5.3 plots the traces of trial decoding each source sample of the **Markov**(q) sources of Section 4.3. Each colored trace is a representative from among those trials belonging to the same source parameter value q. The terminal error rate ε_s° shows stereotypical waterfall behavior around the critical coding point, transitioning abruptly from no error above $r_{\text{code}} = \mathrm{r}^*_{\text{code}}$ to plentiful errors below it. The terminal iteration count shows a double waterfall behavior, where above or below the critical coding point, decoding either becomes exceedingly easy or it becomes quickly evident that decoding will not succeed. However, approaching the critical point closely on either side requires increasing complexity. These are well known behaviors of LDPC codes with BP decoding. Taken together, they suggest heuristics for locating the critical rate r^*_{code} with limited complexity and for designing well founded termination criteria.



(a) terminal error rate



(b) terminal iteration count

Figure 5.3: Decoding **Markov**(q) sources of various entropy rates (0.1 to 0.9). $n = 10000$. Traces each show the decoding of one sample at various coding rates. Circles mark critical coding points.

5.3.2 Convergence dynamics

To study what drives convergence in the BP decoding of `SEP`, we use **Ising**(p, q) as a model source. We first show examples, then develop a method inspired by EXIT function analysis of channel coding [64, 65] to summarize the evolution of messages on the joint source-code graph of `SEP`.

5.3.2.1 Examples

The three classes of messages passing through the controller (Table 3.2) are the source-side external message input $\left[M^{i\leftarrow\mathcal{G}}\right]$, the code-side external message input $\left[M^{i\leftarrow\mathcal{C}}\right]$, and its own doping messages d^i (Eq. 3.15). Let us choose to attach doping messages to the code-side messages in the following.

Definition 5.6. We identify $b^i_{\mathcal{G}} \triangleq \left[M^{i\leftarrow\mathcal{G}}\right]$ as the *source belief*, $b^i_{\mathcal{C}} = d^i \left[M^{i\leftarrow\mathcal{C}}\right]$ as the *code belief*, and $b^i \triangleq b^i_{\mathcal{G}} b^i_{\mathcal{C}} = d^i \left[M^{i\leftarrow\mathcal{G}}\right] \left[M^{i\leftarrow\mathcal{C}}\right]$ as the *total belief*. For each belief, we define an entropy representing its equivocation, thus we have [4] $\mathbb{h}(b^i_{\mathcal{G}})$, $\mathbb{h}(b^i_{\mathcal{C}})$, and $\mathbb{h}(b^i)$ at the node s_i. Finally, define $\bar{h}_{\mathcal{G}} \triangleq (1/n)\sum_{i=1}^n \mathbb{h}(b^i_{\mathcal{G}})$, $\bar{h}_{\mathcal{C}} \triangleq (1/n)\sum_{i=1}^n \mathbb{h}(b^i_{\mathcal{C}})$, and $\bar{h}_{\mathcal{U}} \triangleq (1/n)\sum_{i=1}^n \mathbb{h}(b^i)$ respectively as the *source uncertainty*, *code uncertainty*, and *total uncertainty*.

Taking the uncertainty $\bar{h}_{\mathcal{U}}$ as an example, it is 1 if and only if its associated marginal beliefs b^i are the identity message $1(s)$ indicating total ignorance, and 0 if and only if b^i are deterministic messages indicating full knowledge. The two cases correspond to initial and

[4] $\mathbb{h}(b) = -\sum_{s\in\mathbf{S}} \tilde{b}(s)\log\tilde{b}(s)$, where $\tilde{b}(s) = b(s)/\sum_{s\in\mathbf{S}} b(s)$.

final decoder states, respectively. Beyond these, uncertainty values are less meaningful. Particularly, intermediate uncertainty values are best interpreted as marginal entropy averages as defined, and not as entropies of the joint beliefs unless strong conditional independence properties are satisfied such as for long LDPC codes and i.i.d. sources.

Fig. 5.4 shows the convergence process of decoding an **Ising**(p, q) sample (Fig. 5.5(a)) at three coding rates. Plotted are the belief uncertainties $\bar{h}_{\mathcal{G}}$, $\bar{h}_{\mathcal{C}}$, $\bar{h}_{\mathcal{U}}$, the source error rate ε_s (cf. Eq. 5.3), and the checksum error rate

$$\varepsilon_x \triangleq \frac{1}{k}\sum_{a=1}^{k} \mathbb{1}\{H_{a,\cdot}\hat{\mathrm{s}}^n \neq \mathrm{x}_a\} = \frac{1}{k}\left\| H\hat{\mathrm{s}}^n - \mathrm{x}^k \right\|_0 \tag{5.4}$$

Also plotted in Fig. 5.4 are gray curves representing $\mathbb{h}(b^i_{\mathcal{G}})$, $\mathbb{h}(b^i_{\mathcal{C}})$, and $\mathbb{h}(b^i)$ at one undoped s_i. Fig. 5.5(b)-(e) further show the initial and terminal total beliefs as grayscale images.

The `SEP` performance of this source was reported in Fig. 4.8(a) previously, where the mean critical coding rate was found to be $r^*_{\text{code}} \approx 0.6949$. (For individual samples it may be higher or lower — e.g., higher on this sample.) Different behaviors are observed for below ($r_{\text{code}} = 0.6$), around ($r_{\text{code}} = 0.7$), and above ($r_{\text{code}} = 0.8$) the critical rate. In particular, these three regimes correspond to the three cases of BP termination outlined in Section 5.3.1, respectively $r_{\text{code}} = 0.8$ is case 1, $r_{\text{code}} = 0.6$ is case 2, and $r_{\text{code}} = 0.7$ is case 3.[5]

In all cases, there is a general decoding trajectory that begins with a doping initialization where $\bar{h}_{\mathcal{C}} \leq 1 - r_{\text{dope}}$ and $\bar{h}_{\mathcal{G}} = (1/n)\sum_{i=1}^{n} \mathbb{H}(\mathsf{s}_i|\mathcal{D}\backslash\mathsf{s}_i)$, and proceeds toward more certain source and code beliefs. With sufficient coding rate, the process reaches a point where $\bar{h}_{\mathcal{C}} = 0$ and $\bar{h}_{\mathcal{G}} = (1/n)\sum_{i=1}^{n} \mathbb{H}(\mathsf{s}_i|\mathsf{s}_{\sim i}) \leq \mathbb{H}(\mathsf{s})$, representing full decoding. If coding rate is not sufficient, however, the process is "stuck" along the way, and either reaches a stable false solution (very rare) or oscillates in regions with many plausible solutions concordant with the source model but far away from the true solution.

5.3.2.2 Message ensemble evolution

In Section 5.4, we develop an EXIT function analysis for the evolution of message ensembles on the `SEP` decoder. Using it, we can learn about the microstructure of the convergence dynamics of `SEP` without the complexity of actually running it.

In particular, we can examine the entropy update quantities $h_{\mathcal{S}\leftarrow\mathcal{X}}$ and $h_{\mathcal{S}\leftarrow\mathcal{S}}$, which are EXIT analogs of the `SEP` decoder quantities

$$\bar{h}_{i\leftarrow a} \triangleq \frac{\sum_{i=1}^{n}\sum_{a\in\mathcal{N}_i^{\mathcal{C}}} \mathbb{h}(m^{i\leftarrow a})}{\sum_{i=1}^{n} |\mathcal{N}_i^{\mathcal{C}}|}$$

[5]Cases 2 and 3 are truly distinct only for idealized sources, e.g., those with disjoint correlation neighborhoods around doped bits — roughly, sources with short-range correlation. Here, with the Ising model operating above its parameter threshold (Section 4.4), correlation is long-range, so $r_{\text{code}} = 0.6$ and $r_{\text{code}} = 0.7$ differ only in the degree of message stability.

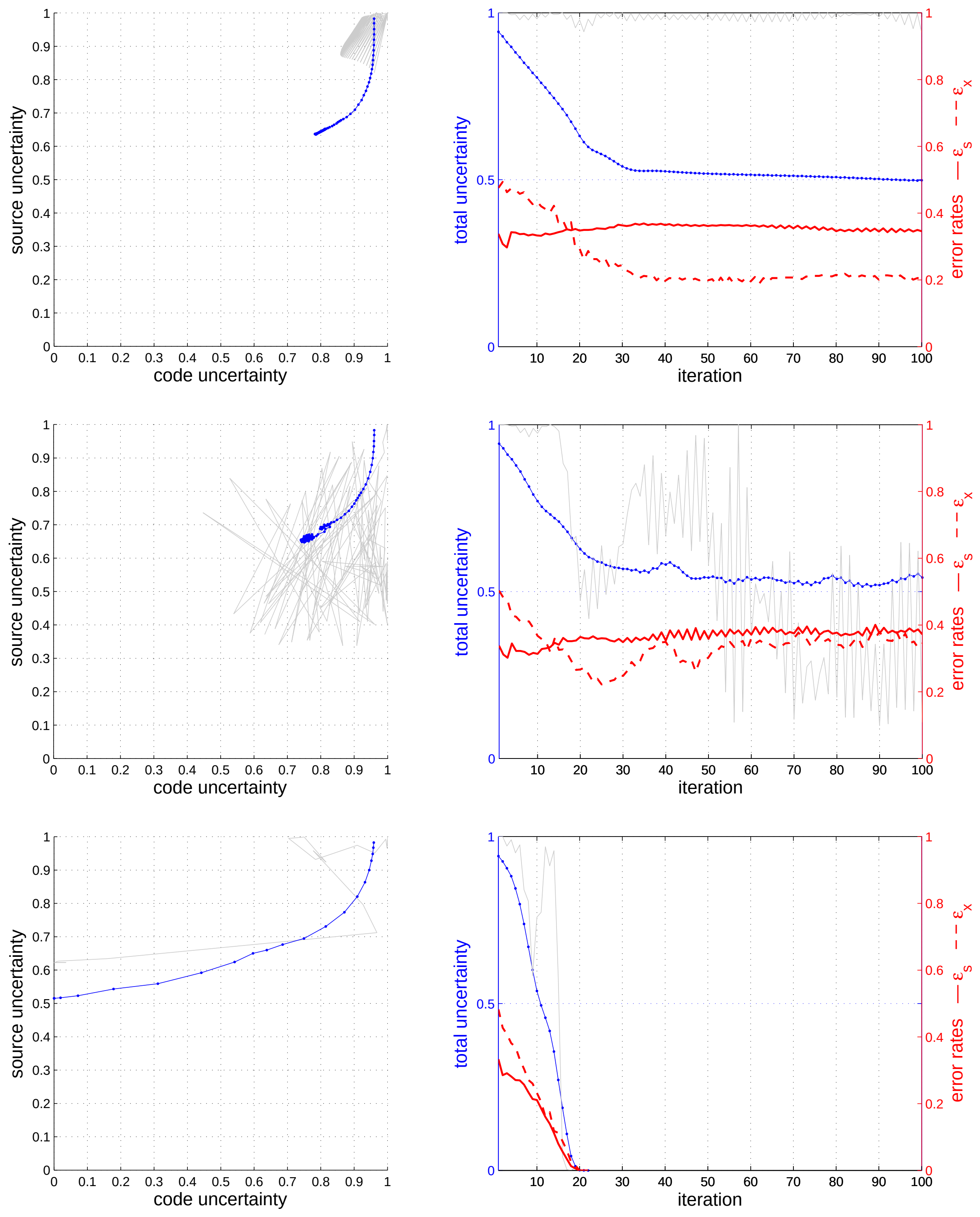


Figure 5.4: Convergence process of decoding a sample from **Ising**$(\frac{1}{2}, 0.7)$, $(h, w) = (32, 32)$, using `SEP-prot`, $r_{\text{dope}} = 0.04$, and (top to bottom) $r_{\text{code}} = 0.6, 0.7, 0.8$. Plotted are source uncertainty $\bar{h}_{\mathcal{G}}$ vs. code uncertainty $\bar{h}_{\mathcal{C}}$ (left), and total uncertainty $\bar{h}_{\mathcal{U}}$ along with source and checksum error rates ε_s and ϵ_x vs. iteration (right).

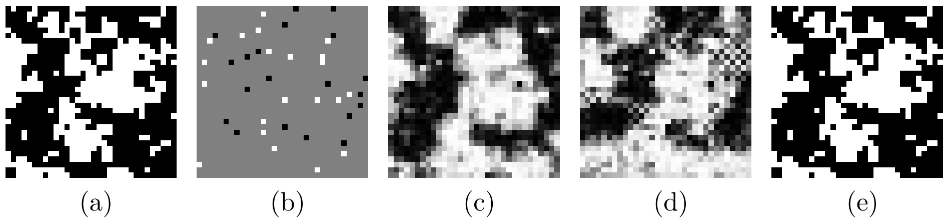


Figure 5.5: Decoding a sample from $\mathbf{Ising}(\frac{1}{2}, \frac{7}{10})$, $(h, w) = (32, 32)$, using `SEP-prot`, $r_{\text{dope}} = 0.04$: (a) ground truth $\mathrm{s}^{h\times w}$; (b) doping initialization $d(s^{h\times w} = 1)$; (c)-(e) total beliefs $b(s^{h\times w} = 1)$ at termination for $r_{\text{code}} = 0.6, 0.7, 0.8$, respectively.

and

$$\bar{h}_{i\leftarrow j} \triangleq \frac{\sum_{i=1}^{n} \sum_{j\in\mathcal{N}_i^{\mathcal{G}'}} \mathbb{h}(\mu^{i\leftarrow j})}{\sum_{i=1}^{n} \left|\mathcal{N}_i^{\mathcal{G}'}\right|}$$

These are related to the belief uncertainties of Section 5.3.2.1 — since beliefs accumulate from messages — but more fine-grained.

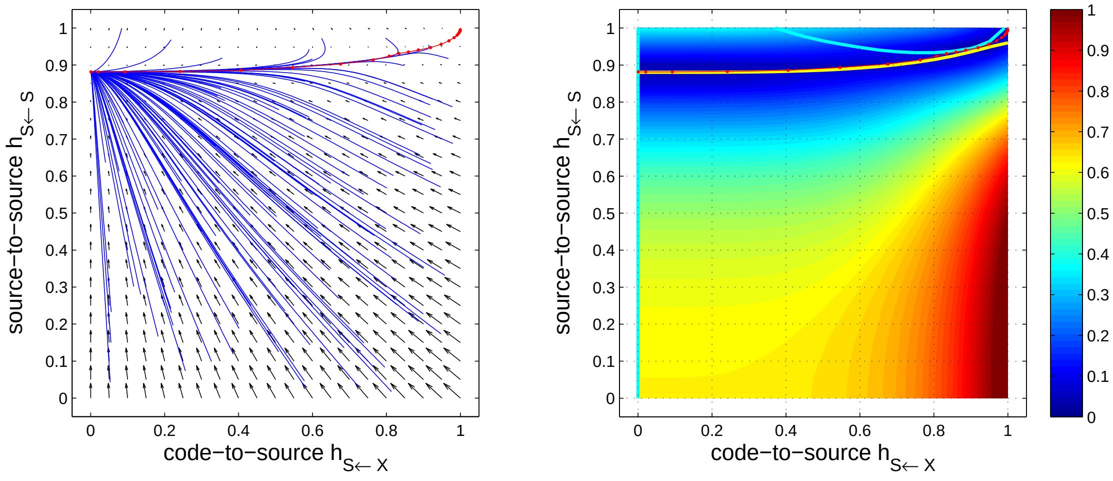
Fig. 5.6 and 5.7 demonstrate the EXIT quantities with actual `SEP` simulations overlaid. On the left, the gradients of the EXIT quantities $h_{\mathcal{S}\leftarrow\mathcal{X}}$ and $h_{\mathcal{S}\leftarrow\mathcal{S}}$ under update rules are shown as a vector field in black with sample flows in blue. The trajectory of the analogous empirical quantities $\bar{h}_{i\leftarrow a}$ and $\bar{h}_{i\leftarrow j}$ from decoding a sample in `SEP` is overlaid in red. On the right is a contour plot of the magnitude change in the uncertainty vector $(h_{\mathcal{C}}, h_{\mathcal{G}})$ during one EXIT evolution step; it indicates how dynamic a state space region is. Also plotted are the zero gradient curves of $h_{\mathcal{C}}$ (cyan) and $h_{\mathcal{G}}$ (yellow), and the same `SEP` simulation trace.

The main feature to note is that successful decoding proceeds in regions where both $h_{\mathcal{C}}$ and $h_{\mathcal{G}}$ (likewise $h_{\mathcal{S}\leftarrow\mathcal{X}}$, $h_{\mathcal{S}\leftarrow\mathcal{S}}$) have negative gradients. Such region is above the yellow curve (source model active) and below the cyan curve (coding constraints active). Thus we see the general behavior of `SEP` decoding is (1) to be activated initially by the gap in the top-right between the two curves granted by any compressible source, (2) to move toward the yellow curve via the source model and, in doing so, (3) activating the coding constraints and, if the coding rate is sufficient to allow a path, (4) to finally be driven to successful decoding.

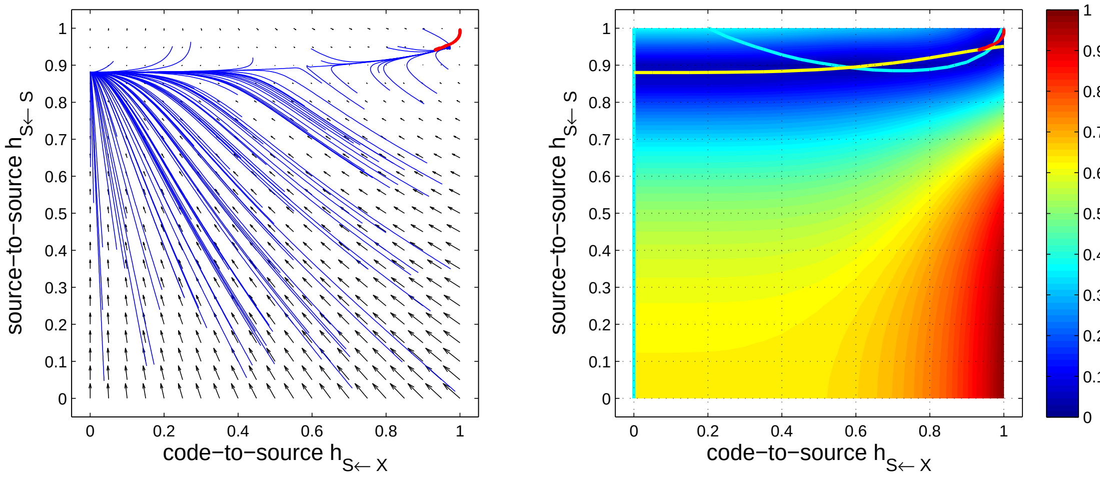
There is much to be learned from these analyses: about initialization, about the interaction between code and model, and code design and doping choices. This tool provides a starting point for further inquiries.

5.4 Appendix: EXIT analysis*

Traditionally, the extrinsic information transfer (EXIT) function $h(\mathrm{h})$ is defined for the decoders of binary symmetric DMC's ([41], Chapter 4). Namely, provided a virtual DMC W ($\mathsf{y} = \mathsf{W}(c)$) from an h-parameterized collection $\mathcal{W}(\mathrm{h})$, a received word y, and a *marginal*

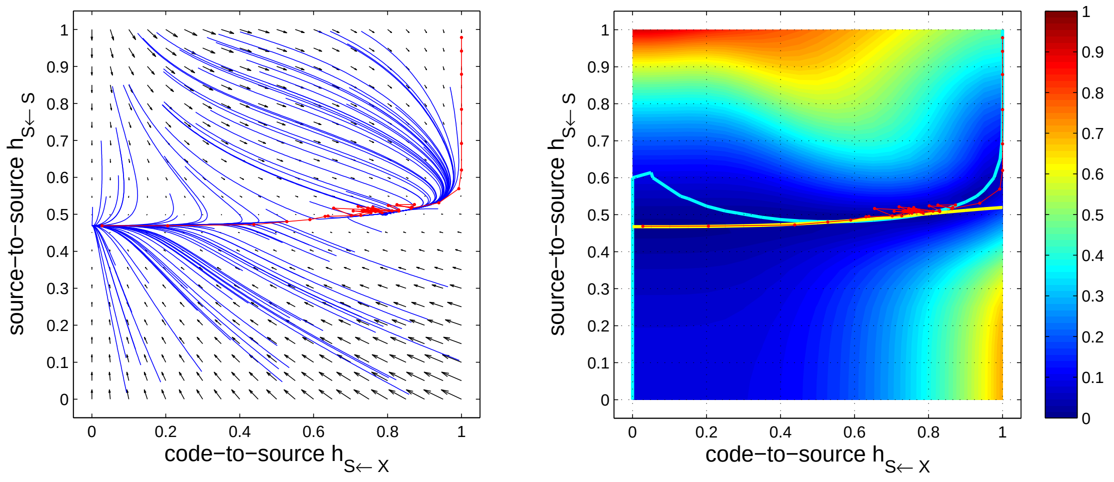


(a) $r_{\text{code}} = 0.8$

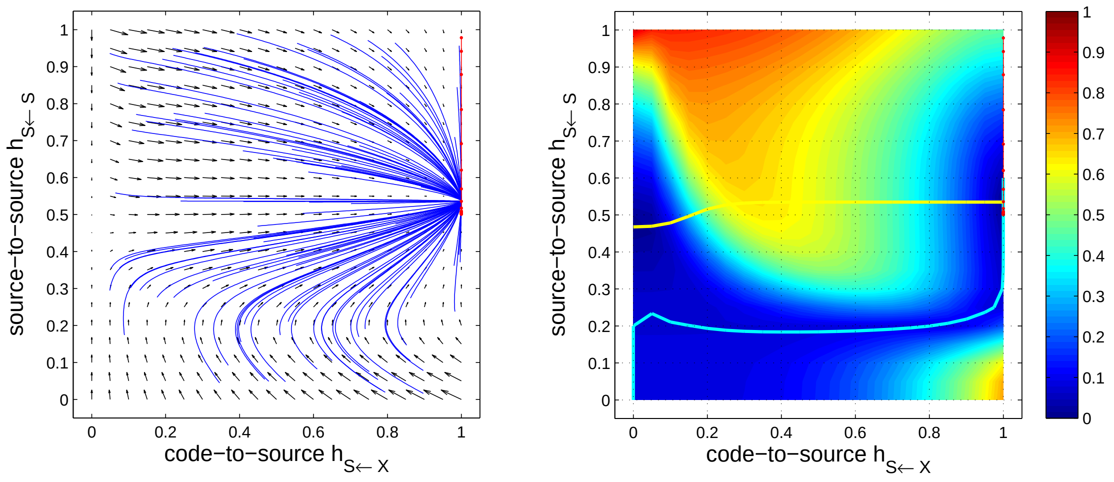


(b) $r_{\text{code}} = 0.6$

Figure 5.6: Message entropy evolution for **Ising**$(\frac{1}{2}, 0.7)$. EXIT vs. `SEP`.



(a) $r_{\text{code}} = 0.1$



(b) $r_{\text{code}} = 0.01$

Figure 5.7: Message entropy evolution for $\textbf{Ising}(\frac{1}{2}, 0.9)$. EXIT vs. `SEP`.

extrinsic estimator that produces $\hat{\mathsf{c}}_i(\mathrm{y};\mathrm{h}) = \mathsf{Estimate}(c_i|\mathrm{y}_{\sim i};\mathsf{W}\in\mathcal{W}(\mathrm{h}))$,[6] we have

$$h_i(\mathrm{h}) \triangleq \mathbb{H}(\hat{\mathsf{c}}_i(\mathsf{y};\mathrm{h}))$$

$$h(\mathrm{h}) \triangleq \frac{1}{n}\sum_{i=1}^{n} h_i(\mathrm{h})$$

$\mathsf{W}\in\mathcal{W}(\mathrm{h})$ is defined to have capacity $1-\mathrm{h}$; after applying $\mathsf{Estimate}$, a new virtual channel W' is formed between $\hat{\mathsf{c}}$ and y, with a different (larger) capacity. The attention of EXIT function analysis is then focused on considering whether a particular decoder provides a path from $\mathsf{y}^{(0)} = \mathsf{W}^{(0)}(c)$ of the initial (physical) DMC $\mathsf{W}^{(0)}\in\mathcal{W}(\mathrm{h}^{(0)} = 1-C)$ to eventually arrive at $\mathsf{y}^{(I)} = c$ of the noiseless virtual DMC $\mathsf{W}^{(I)}\in\mathcal{W}(\mathrm{h}^{(I)} = 0)$ representing complete decoding after some I iterations.

Let us now specialize to the additive-noise DMC's, whose duals are the DMS's. Let $\mathbf{W}(\mathrm{h})$ denote instead channel noise distributions, e.g. $\mathsf{y} = c+\mathsf{w}$, $\mathsf{w}\sim\mathbf{W}(\mathrm{h})$. Such a channel having capacity $1-\mathrm{h}$ means $\mathbb{H}(\mathsf{w}) = \mathrm{h}$.

For iterative decoders whose action can be modeled as repeatedly applying the marginal extrinsic estimator, a convergence dynamics is generated. If we rewrite the estimator as $\hat{\mathsf{c}}_i(\mathrm{y};\mathrm{h}) = \mathrm{y}_i - \hat{\mathsf{w}}_i(w;\mathrm{h})$ with $\hat{\mathsf{w}}_i(w;\mathrm{h}) \triangleq \mathsf{Estimate}(w_i|w_{\sim i};\mathsf{w}\sim\mathbf{W}(\mathrm{h}))$, then after each application, the entropy of w can be considered as updated to $(1/n)\sum_{i=1}^{n}\mathbb{H}(\hat{\mathsf{w}}_i(w;h)) = h(\mathrm{h})$, under some independence approximations. Thus we interpret $\mathrm{h}\mapsto h(\mathrm{h})$ as tracking the stepwise evolution of $\mathbb{H}(\mathsf{w})$ under the application of the estimator. So far as DMS's are concerned, EXIT function analysis can be borrowed nearly verbatim, substituting total belief $b(s)$ for $p_{\mathsf{w}}(w)$ and total uncertainty $\bar{h}$ for $\mathbb{H}(\mathsf{w})$.[7]

5.4.1 EXIT with sources

For sources with memory, let us view EXIT function analysis not as an exact method for asymptotic regimes, but as a set of rules that *stylize* full BP decoding. Instead of passing individual messages, we pass average entropies representing *message ensembles* between *stylized nodes*. The following is a treatment of extending the method to the `SEP` decoder with a homogeneous pairwise model having edge potential ψ.

Definition 5.7. Let $\mathbf{a},\mathbf{b}$ be mixture binary distributions.[8] Let $\mathbf{BEC}(h)$ be the mixture binary distribution $\{(h,[\frac{1}{2};\frac{1}{2}]),(1-h,[0;1])\}$, called the *BDS mixture*. Let $\mathbf{BSC}(h)$ be the

[6]W is a stochastic function implementing a conditional law $p_{\mathsf{y}|\mathsf{c}}$. $\mathsf{Estimate}$ produces a marginal random variable (n.b., not deterministic) from a specific distribution. For a uniform input distribution on c, $\mathbb{H}(\mathsf{c}|\mathsf{y}) = 1-\mathbb{I}(\mathsf{c};\mathsf{y})$ characterizes the equivocation introduced by the channel. If such c is capacity (C) achieving, then $\mathbb{H}(\mathsf{c}|\mathsf{y}) = 1-C$.

[7]For $\mathbf{Bern}(p)$ sources for example, the initial beliefs $b^{(0)}$ are the node potentials, which give $\bar{h}^{(0)} = h(p)$; at the termination of decoding with `SEP`, $\bar{h}^{(I)}$ is a combination of the node potentials and wholly satisfied checksum constraints, thus equals zero.

[8]A mixture binary distribution $\mathbf{a}$ is an object representing a collection of pairs $\{(l_i,[1-u_i;u_i])\}_i$ where l is a function summing to 1.

mixture binary distribution $\{(p, [1; 0]), (1 - p, [0; 1])\}$ where $h(p) = h$, called the *Bernoulli mixture.*

Remark 5.8. For convenient data structures and a calculus on mixture binary distributions, along with reasons for choosing $\mathbf{BEC}(h)$ and $\mathbf{BSC}(h)$, we refer to [41].

Definition 5.9. Let $\mathcal{S}$ and $\mathcal{X}$ be stylized source and factor nodes. Let λ be the code-side degree distribution of $\mathcal{S}$,[9] let γ be its source-side degree distribution. Let ρ be the degree distribution of $\mathcal{X}$. Let $\mathbf{a} \circ \mathbf{b}$ be the accumulation of distributions, $f(\otimes ...)$ be filtration by the checksum constraint, and $\psi(\mathbf{a})$ be filtration by the pairwise edge potential.

Now, define as shorthand

$$[\mathbf{M}(h_{\mathcal{S}\leftarrow\mathcal{X}})] \triangleq \sum_i \lambda_i \mathbf{BSC}(h_{\mathcal{S}\leftarrow\mathcal{X}})^{\circ i}$$

$$\mathbf{M}^-(h_{\mathcal{S}\leftarrow\mathcal{X}}) \triangleq \sum_i \lambda_i \mathbf{BSC}(h_{\mathcal{S}\leftarrow\mathcal{X}})^{\circ(i-1)}$$

$$[\mathbf{M}(h_{\mathcal{S}\leftarrow\mathcal{S}})] \triangleq \sum_i \gamma_i \mathbf{BSC}(h_{\mathcal{S}\leftarrow\mathcal{S}})^{\circ i}$$

$$\mathbf{M}^-(h_{\mathcal{S}\leftarrow\mathcal{S}}) \triangleq \sum_i \gamma_i \mathbf{BSC}(h_{\mathcal{S}\leftarrow\mathcal{S}})^{\circ(i-1)}$$

$$\mathbf{d} \triangleq \mathbf{BEC}(1 - r_{\text{dope}})$$

Finally, define the entropy update operations:

- *$\mathcal{X}$-node output computation*:

$$h_{\mathcal{S}\leftarrow\mathcal{X}} \Leftarrow \mathbb{h}\left(\sum_i \rho_i f(\otimes_{i-1}\mathbf{BEC}(h_{\mathcal{S}\rightarrow\mathcal{X}}))\right)$$

- *$\mathcal{S}$-node output computation — source side*:

$$h_{\mathcal{S}\leftarrow\mathcal{S}} \Leftarrow \mathbb{h}\left(\psi\left(\mathbf{d} \circ \mathbf{M}^-(h_{\mathcal{S}\leftarrow\mathcal{S}}) \circ [\mathbf{M}(h_{\mathcal{S}\leftarrow\mathcal{X}})]\right)\right)$$

- *$\mathcal{S}$-node output computation — code side*:

$$h_{\mathcal{S}\rightarrow\mathcal{X}} \Leftarrow \mathbb{h}\left(\mathbf{d} \circ \mathbf{M}^-(h_{\mathcal{S}\leftarrow\mathcal{X}}) \circ [\mathbf{M}(h_{\mathcal{S}\leftarrow\mathcal{S}})]\right)$$

We can run these with the same schedule as the `SEP` decoder and obtain the evolution of these quantities. If we wish, we can also compute the EXIT versions of the belief uncertainties of Section 5.3.2.1 from message ensembles,

[9]A degree distribution λ is a function summing to 1, with each term λ_i indicating the proportion of edges in a collection that belongs to degree-i nodes.

$$h_{\mathcal{C}} = \mathbb{h}\left(\mathbf{d} \circ [\mathbf{M}(h_{\mathcal{C}})]\right)$$

$$h_{\mathcal{G}} = \mathbb{h}\left([\mathbf{M}(h_{\mathcal{G}})]\right)$$

$$h_{\mathcal{U}} = \mathbb{h}\left(\mathbf{d} \circ [\mathbf{M}(h_{\mathcal{G}})] \circ [\mathbf{M}(h_{\mathcal{C}})]\right)$$

6 Modeling Details

Data compression rests on the assumption that we have a good model to capture the data characteristics of interest, but sometimes the data model is not as perfectly known as we would like. Traditionally, this is where separate universal compression systems are called for (Domain II of Table 1.1). In contrast, in model-code separation we consider uncertainty as extended data model questions. We begin this chapter by giving an interpretation of the codebook as a form of model representation, and argue why strong universality implies impractical compression. Then we distinguish two practical cases relevant to `SEP` and find that: (1) when there is a mismatch between the "true" model and the assumed model, practical performance operationalizes theory; and (2) when the "true" model is "partially known," i.e., as belonging to a parametric family among which one model is true, uncertainty can be expressed within graphical model representation.

6.1 Model representation revisited

So far we have seen the algebraic form p_{s^n} and the source graph $\mathcal{G}$ as representations of a data model. They are "representations" because they can express the same underlying prior knowledge. For the purpose of compression, the codebook $\mathscr{C}_n$ that p_{s^n} or $\mathcal{G}$ generates is also a representation.

6.1.1 Enumerative representation

To see this, consider that in the entropy-achieving schemes of source coding theory (Section 2.1), p_{s^n} only plays a role in the generation of $\mathscr{C}_n$, then p_{s^n} is itself discarded. Since the codebook remains the only piece of prior knowledge shared, assumed, or otherwise unconveyed within the system, it is, *ipso facto*, what holds the data model. More to the point, where AEP holds, $\mathscr{C}_n$ is an explicit enumeration of some suitably defined notion of a typical set, thus it is nothing more than an *enumerative representation* of "all" the typical examples of data that may come up for compression. We have alluded to such an enumerative representation in Section 2.3.

Because $\mathscr{C}_n$ is exponentially large in n, it is not usually the native representation used in practical compression. However, having this representation in mind helps us to understand situations when $\mathscr{C}_n$ does not hold exactly the typical sequences of the true model.

6.1.1.1 Codebook information

An important question about representation is the cost of describing a model within it. The cost in the enumerative representation is the information value of a codebook, which we compute next.

Definition 6.1. Without loss of generality, assume $\log |\mathbf{S}| = 1$ (i.e., binary alphabet). Let $N \triangleq 2^n$ be the total number of n-bit sequences. Let $N_{\mathscr{C}} \triangleq 2^{nH}$ be the size of a codebook $\mathscr{C}_n$ for a model compressing to rate H.

In the manner of type classes on strings [66], let us assume the size of a codebook, $N_{\mathscr{C}}$, is also a "free" parameter that is sub-exponential in N and costs "nothing" to describe. To construct $\mathscr{C}_n$ of size $N_{\mathscr{C}}$, we choose $N_{\mathscr{C}}$ sequences from among N. Thus there are [1]

$$\binom{N}{N_{\mathscr{C}}} \doteq 2^{Nh(N_{\mathscr{C}}/N)} \tag{6.1}$$

codebooks of size $N_{\mathscr{C}}$. Letting $R_{\mathscr{C}} \triangleq 1 - (\log N_{\mathscr{C}})/n = 1 - H$, such a codebook requires

$$\begin{aligned}
\lim_{n\to\infty} \frac{1}{nN_{\mathscr{C}}} Nh(N_{\mathscr{C}}/N) &= \lim_{n\to\infty} \frac{1}{nN_{\mathscr{C}}} 2^n \Big\{ nR_{\mathscr{C}} 2^{-R_{\mathscr{C}}} \\
&\quad -(1-2^{-nR_{\mathscr{C}}}) \log(1-2^{-nR_{\mathscr{C}}}) \Big\} \\
&= \lim_{n\to\infty} \frac{1}{N_{\mathscr{C}}} 2^n R_{\mathscr{C}} 2^{-nR_{\mathscr{C}}} \\
&= R_{\mathscr{C}}
\end{aligned} \tag{6.2}$$

bits per codebook symbol to specify. We interpret this to mean that an arbitrary codebook $\mathscr{C}_n$ of size $N_{\mathscr{C}}$ representing a model holds roughly $nN_{\mathscr{C}}R_{\mathscr{C}}$ bits of information.

6.1.1.2 Compression

With an enumerative model, we can obtain data compression by conveying a codebook index rather than the full sequence. Let us neglect the computational complexity, and merely observe the rate performance implications.

In Eq. 6.2, the codebook information rate $R_{\mathscr{C}} = 1 - H$ and the compression rate H are complementary in the following sense: the data model $\mathscr{C}_n$ has just the amount of information to equal the rate reduction obtained by compressing all the strings in $\mathscr{C}_n$, using $\mathscr{C}_n$. This immediately implies a breakeven point between two obvious approaches to compressing n-bit strings s^n:

- *Fully uncoded*: Spend n bits to describe each s^n directly; and

[1]With n (or N) understood, we write $a(n) \doteq b(n)$ to mean $\lim_{n\to\infty} \log a(n)/\log b(n) = 1 + o(\epsilon)$. $h(x) \triangleq -x\log x - (1-x)\log(1-x)$.

- *Fully modeled*: Spend $nN_{\mathscr{C}}R_{\mathscr{C}} = n2^{nH}(1-H)$ bits of one-time overhead to specify a codebook of size $N_{\mathscr{C}}$, then spend nH bits to describe each s^n.

The breakeven occurs when $\mathscr{C}_n$ is used 2^{nH} times, i.e., the total number of typical sequences. By the time we begin to make compression gain with the second (fully modeled) method, we will have described all useful sequences at full rate at least once. We could have equally well used the first (fully uncoded) method.

What about an intermediate method with a partial codebook? The situation is not improved:

Definition 6.2. A partial codebook of $\mathscr{C}_n$ of 2^{nR} entries, where $R \leq H$, is denoted by $\mathscr{C}_n^-(R)$. Define also $N_{\mathscr{C}}^-(R) \triangleq 2^{nR}$.

- *Partially modeled*: Spend $n2^{nR}(1-R)$ bits of one-time overhead to specify a partial codebook $\mathscr{C}_n^-(R)$ of size $N_{\mathscr{C}}^-(R)$, then spend nR bits to describe each s^n if found in it, and spend n bits to describe s^n uncoded if not found.

Proposition 6.3. *The partially modeled method breaks even in total rate with the uncoded method (and therefore the fully modeled method) when the codebook is used on average 2^{nH} times.*

Proof. With probability $2^{n(R-H)}$, s^n is found in $\mathscr{C}_n^-(R)$, so it is coded with nR bits. With probability $1-2^{n(R-H)}$, s^n is not found in the codebook, so it is coded with n bits. After K usages, the uncoded method expends Kn bits, while using $\mathscr{C}_n^-(R)$ expends on average $K2^{n(R-H)}nR + K(1-2^{n(R-H)})n + n2^{nR}(1-R)$ bits. Setting $K = 2^{nH}$ gives $2^{nH}n$ bits for both methods. □

Thus all three methods only produce net compression after $N_{\mathscr{C}} = 2^{nH}$ usages. Clearly this can only be useful if $N_{\mathscr{C}}$ is small.

6.1.1.3 Learning

The partially modeled method of Section 6.1.1.2 can be converted to a learning scheme:

- *Enumerative model learning*: Begin with $\mathscr{C}_n^-(0)$. Whenever a sequence s^n is not found in the codebook $\mathscr{C}_n^-(R_i)$, describe it uncoded with n bits and add it to form the new codebook $\mathscr{C}_n^-(R_{i+1})$. If s^n is found in $\mathscr{C}_n^-(R_i)$, then describe it with nR_i bits.

We can compute the compression rate arising from it. Put $N_{\mathscr{C}}^-(R_i)/N_{\mathscr{C}}$ as the probability of finding a source sequence in the partial codebook $\mathscr{C}_n^-(R_i)$ of size $N_{\mathscr{C}}^-(R_i) \leq N_{\mathscr{C}}$. The expected coding rate with $\mathscr{C}_n^-(R_i)$ is

$$r(i) = \frac{N_{\mathscr{C}}^-(R_i)}{N_{\mathscr{C}}} \log N_{\mathscr{C}}^-(R_i) + \left(1 - \frac{N_{\mathscr{C}}^-(R_i)}{N_{\mathscr{C}}}\right) n \tag{6.3}$$

bits. However, $N^-_{\mathscr{C}}(R_i) \Rightarrow N^-_{\mathscr{C}}(R_{i+1}) = N^-_{\mathscr{C}}(R_i) + 1$ with probability $1 - N^-_{\mathscr{C}}(R_i)/N_{\mathscr{C}}$. The average waiting time at i is $N_{\mathscr{C}}/(N_{\mathscr{C}} - N^-_{\mathscr{C}}(R_i))$, thus the expected total rate spent while learning the full codebook is

$$r_{\text{incremental}} = \sum_{i=0}^{N_{\mathscr{C}}-1} \frac{N_{\mathscr{C}}}{N_{\mathscr{C}} - i} r(i) \tag{6.4}$$

We can compare this to the cases of Section 6.1.1.2 where the best total rate is

$$r_{\text{fixed}} = nN_{\mathscr{C}}R_{\mathscr{C}} + \left(\left(\sum_{i=0}^{N_{\mathscr{C}}-1} \frac{N_{\mathscr{C}}}{N_{\mathscr{C}} - i} \right) - N_{\mathscr{C}} \right) nH \tag{6.5}$$

over the same number of uses, and find that the learning method is a negligible improvement over them. In other words, learning for an enumerative data model is no different from uncoded description, i.e. no compression, until all the typical sequences have been seen once.

Notice that enumerative model learning describes a process essentially identical to dictionary-learning universal compression such as Lempel-Ziv.

6.1.2 Choice of representation

It should be pointed out that even for the same data model, each representation may use a different (nominal) amount of information to convey the model (e.g. [67, 68] for certain graphical models). The reason is that the choice of representation itself has information value. It implies a model on sequences that puts a prior on which data models are likely and thus are given succinct descriptions.

For example, in Kolmogorov complexity theory, an area of computational complexity theory intersecting with information theory [69, 70, 71], the boundary between data model and compressed data is fluid. In this theory, the compressed output of s^n is the shortest program x that generates s^n on a universal Turing machine U with some constant amount of pre-stored data. This pre-stored data in fact is a data model (a complexity model) that allows certain classes of strings to be generated with shorter programs. We can always add to or remove from this store to change assumptions. If the store is a program that interprets graphical models, then it is no different from choosing graphical models as representation.

This theory also says that almost all strings are incompressible, thus almost all models are not describable succinctly besides by listing the content of their enumerative representations, i.e. codebooks. Otherwise, those codebooks, once flattened into strings, can be compressed. Thus the enumerative representation is optimal for almost all models.

Finally, it is known that universal learning is impossible, because the "universal" source model

$$p_{\mathsf{U}}(s^n) = \sum_{x:\mathsf{U}(x)=s^n} 2^{-|x|} \tag{6.6}$$

cannot be rejected by learning ([72], Chapter 7). There is always a risk of model mismatch by assuming any other model to be the true model.

Taken together, it means nothing magical happens simply by switching representations, and that whatever gains made on model description cost and model learning over the enumerative representation come from additional informative assumptions, amounting also to a data model. This puts uncertainty in data model on the same footing as uncertainty in data.

6.2 Unknown model

Section 6.1 tells us that we cannot get practical compression with very little data model, a regime we call *strong universality*.

Indeed, such systems as Lempel-Ziv (LZ) [51, 50], context-tree weighting (CTW) [52], prediction by partial matching (PPM) [73], dynamic Markov coding (DMC) [74], and a variety of other switching, learning, and adaptive compression algorithms [75] operate far from strong universality [76]. The assurance that they approach the entropy rate at asymptotically long data length by learning cannot be relied upon, because the operative "length" is on the order of $n2^{nH}$ for general models with entropy rate H (Section 6.1.1.3). In reality, they achieve compression only for models preferred by their representation and even among those, only for the least complex ones, e.g. finite-state-machine sources with *few* states for LZ, context-tree sources with *small* tree size for CTW, parametric family or switched collection of sources with a *small* parameter space or collection size in some other cases. To be at all practical, they must assume enough prior knowledge so that the remaining data model ambiguity is small and learnable at reasonable data length.

We conclude that the only models that can be compressed practically are those with explicit low-complexity representation, either a small codebook that can be enumerated, or a large codebook generated by parametric representation over a small parameter space. In that sense, practical (non-strongly) universal compression is not vastly different from non-universal compression after all. The only distinction is that the data model assumed, $\mathscr{C}_{\text{known}}$, may differ from the true data model $\mathscr{C}_{\text{true}}$, a form of mismatch. There are three scenarios:

1. $\mathscr{C}_{\text{known}} \subset \mathscr{C}_{\text{true}}$, the case of partial model that is generally impractical, so we must accept mismatch, addressed in Section 6.3;
2. $\mathscr{C}_{\text{known}} \supset \mathscr{C}_{\text{true}}$, the case of parameter estimation, addressed in Section 6.4; and
3. $\mathscr{C}_{\text{known}} \cap \mathscr{C}_{\text{true}} \neq \mathscr{C}_{\text{true}}$ and $\mathscr{C}_{\text{known}} \cap (\mathbf{S}^n \backslash \mathscr{C}_{\text{true}}) \neq \emptyset$, the case that can be split into the first two cases.

Next we address the two practical cases for compression with model uncertainty using `SEP`.

6.3 Model mismatch

The simplest approach to compression with uncertain data models is to assume a specific one, and find out the cost of being wrong. We show that putting the incorrect model into our

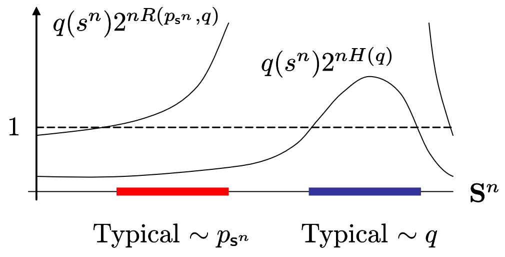

$q(s^n)2^{nR(p_{\mathsf{s}^n},q)}$
$q(s^n)2^{nH(q)}$
1
$\mathbf{S}^n$
Typical $\sim p_{\mathsf{s}^n}$
Typical $\sim q$

Figure 6.1: In this sketch, the vertical axis is the expected number of times an s^n occurs in a codebook. The curves are $q(s^n)\,|\mathscr{D}_n|$. A codebook generated according to q needs a much larger size to cover the red set than the blue set; many duplicate entries are also generated.

decoder of Section 3.2.3 results in the performance degradation suggested by large deviation theory.

Given a source s^n distributed as $p_{\mathsf{s}^n}(s^n)$ but compressing using a random codebook $\mathscr{D}_n$ for a source distributed as $q(s^n)$, theory [72] suggests coding takes place at rate no less than [2]

$$R(p_{\mathsf{s}^n}, q) \triangleq \lim_{n\to\infty} \frac{1}{n}\mathbb{E}(-\log q(\mathsf{s}^n)) = \mathbb{H}(\mathsf{s}) + \frac{1}{n}D(p_{\mathsf{s}^n}||q) \tag{6.7}$$

Briefly, the codebook $\mathscr{D}_n$ provides the sequences drawn according to q, and therefore for some s^n to occur in $\mathscr{D}_n$ incurs an informational surprise of $-\log q(s^n)$ instead of $-\log p_{\mathsf{s}^n}(s^n)$, thus requiring the much larger codebook of rate $R(p_{\mathsf{s}^n}, q)$ to include s^n with high probability (Fig. 6.1).

Note that this characterization of rate allows redundant entries to occur in $\mathscr{D}_n$, each with a unique index, as it is necessary when the codebook grows beyond the size of the typical set according to q. This does not make sense from an efficient codebook construction point of view, but it has concrete meaning in our decoder — the redundant entries represent more likely words in the hash constraint set than the true realization (Fig. 2.1) that needs a larger hash size to eliminate.

Specifically, in our decoder, the analog to coding with $\mathscr{D}_n$ is using the source subgraph modeling $q(s^n)$ for decoding. BP decoding is equivalent to approximate maximization of

$$u'(s^n) \triangleq c(s^n)q(s^n) \tag{6.8}$$

Compared to Eq. 3.4, the only difference is in the relative probabilities of the words in the hash constraint set. In that sense, $R(p_{\mathsf{s}^n}, q)$ plays the same role in mismatch as $\mathbb{H}(\mathsf{s})$ does in no mismatch for sizing the hash constraint set, and we expect similar relative performance in practice.

[2] $D(p||q) \triangleq \mathbb{E}_p \log(p(s^n)/q(s^n))$. Absolute continuity requirements need to be satisfied, i.e., $q(s^n) = 0 \Rightarrow p_{\mathsf{s}^n}(s^n) = 0$, or else no zero-error coding is possible.

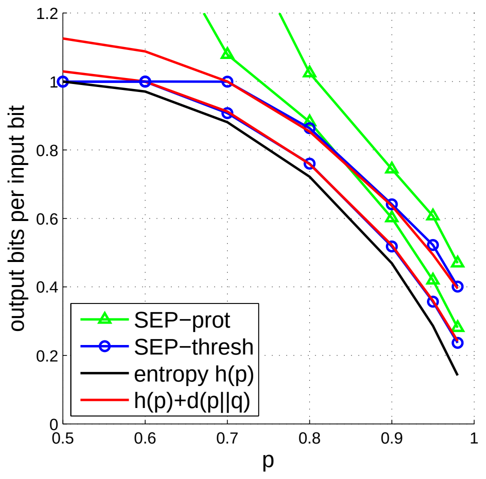


Figure 6.2: Performance of SEP on compressing a **Bern**(p) source with a **Bern**(q) source graph. $r_{\text{dope}} = 0$, $n = 1000$. For each value p, $q = p + 0.1, p + 0.2$ (bottom to top) are shown.

6.3.1 Results

Applying the foregoing, a **Bern**(p) source compressed using a **Bern**(q) random codebook requires a nominal rate of [3]

$$h(p) + d(p||q) = -p \log q - (1 - p) \log(1 - q) \tag{6.9}$$

Note that for $h(p) + d(p||q) > 1$, we no longer need to be concerned with redundant codebook entries because each hash constraint set is expected to have at most one source word, therefore, in practice the rate required should never be greater than for direct description, i.e. $\min\{1, h(p) + d(p||q)\}$.

Fig. 6.2 shows the result of using the source graph of a **Bern**(q) source in the decoder to decode the compressed bits from **Bern**(p) samples. The performance of `SEP-thresh` matches the theoretical best rate, just as happened with the matched model in Section 4.2.

6.4 Parameter estimation

Partially specified models may take myriad forms, but one type are parametric models that provide prior knowledge in the form of a family of fully specified models — equivalently, a collection of codebooks $\{\mathscr{C}_n\}_\theta$ — from which the true model is but one. The reason that parameterization is useful is that some codebooks in the collection $\{\mathscr{C}_n\}_\theta$ may be exponentially smaller than others, implying we should not use an undistinguished model of the entire family for compression.

[3] $d(p||q) \triangleq p \log(p/q) + (1 - p) \log((1 - p)/(1 - q))$.

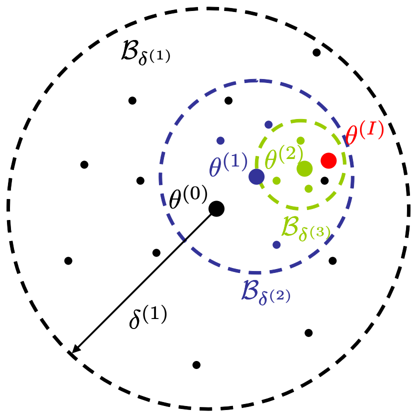

$\mathcal{B}_{\delta^{(1)}}$
$\theta^{(I)}$
$\theta^{(1)}$
$\theta^{(2)}$
$\theta^{(0)}$
$\mathcal{B}_{\delta^{(3)}}$
$\mathcal{B}_{\delta^{(2)}}$
$\delta^{(1)}$

Figure 6.3: Parametric decoding begins with an initial guess of the parameter at θ^0 and searches a neighborhood $\mathcal{B}_{\delta^{(1)}}(\theta^{(0)})$ within the parameter space. The scope of the search narrows in subsequent iterations, finally converging on $\theta^{(I)}$.

Compressing for these models requires a degree of learning. We can estimate parameters (if they do not change) by training on multiple data samples — this is a standard machine learning problem. However, for a single sample for which the parameters are fixed, what we need is a way to compress in the absence of offline training. Given that additional effort is expended to estimate parameters, and there may be uncertainty in that estimate, there could well be a cost in rate.

6.4.1 Parametric decoding

We first give a heuristic, depicted in Fig. 6.3, that jointly estimates parameters within the decoding loop itself.

Suppose s is drawn from a distribution $p(\mathsf{s};\theta)$ where θ denotes unknown parameters. Define the checksum-correctness objective

$$F(\hat{s}) = \frac{1}{k}\left\|H\hat{s} - x\right\| \tag{6.10}$$

where $\|\cdot\|$ is some appropriate norm (e.g. ℓ_0). At each iteration t of the decoding algorithm (Section 3.2.3.3), evaluate $F(\cdot)$ on the contemporary source estimate $\hat{s}^{(t)}(\theta)$ obtained from the total belief update (Eq. 3.13). The value θ^* that minimizes $F(\hat{s}^{(t)}(\theta))$ among choices in a neighborhood $\mathcal{B}_{\delta^{(t)}}(\theta^{(t-1)})$, for some diminishing sequence $\delta^{(1)} > \delta^{(2)} > \cdots > 0$ of bounded sum (e.g. $\delta^{(t)} = \delta/\alpha^t$ for $\delta > 0, \alpha > 1$), is chosen as the parameter estimate $\theta^{(t)}$ for that iteration. At the end of decoding, the sequence of estimates for θ are also converged within some neighborhood. This works well in many cases, but it requires sampling the neighborhood $\mathcal{B}_{\delta^{(t)}}$ well and experimenting with its annealing.

6.4.2 Augmented graphical model

The parametric decoding heuristic makes a hard parameter estimation decision from a restricted set at each step of decoding. This heuristic points to a more general way to deal with parametric models, which is to augment the graphical source model with parameter nodes.

Remark 6.4. Suppose we have a $\mathbf{Bern}(\theta)$ source with unknown $\theta \in [0,1]$. How many additional bits does it take to compress? Is the excess related to specifying the value θ? Note that we cannot specify the real number θ with a finite number of bits even if we have it. However, one can easily be convinced that for finite n, the compression of s^n should not need the full precision of θ, because a sequence of that length can only distinguish between $n+1$ models corresponding to $n+1$ type classes; in other words, the model specification — and the estimation of θ — should be able to have error without consequence, so it would be wrong to make a hard parameter decision.

Without loss of generality, let us now consider compressing for this $\mathbf{Bern}(\theta)$ source using the model-code separation architecture.[4]

It turns out explicit estimation of θ is not necessary, because compression is itself a way to "determine" and "specify" an unknown parameter θ up to the required precision. The main thing to realize is that, while θ is deterministic, we can construct a Bayesian belief $p_\Theta(\theta)$ for it. We also do not know what $p_\Theta(\theta)$ is, but we assume the true θ is drawn from a random variable $\Theta \sim p_\Theta$. Thus we can write the joint distribution (s^n, Θ) as

$$p_{\mathsf{s}^n\Theta}(s^n,\theta) = p_\Theta(\theta)\prod_{i=1}^{n} p_{\mathsf{s}_i|\Theta}(s_i|\theta) \tag{6.11}$$

whereby using the same notation as in Section 4.2, we have

$$p_{\mathsf{s}_i|\Theta}(s_i|\theta) = [1-\theta;\theta](s_i,\theta)$$

If we compare this to the original Eq. 4.2,

$$p_{\mathsf{s}^n}(s^n) = \prod_{i=1}^{n}\phi(s_i) = \prod_{i=1}^{n}[1-p;p](s_i)$$

we notice that the unary function $\phi(s_i) = [1-p;p](s_i)$ at each s_i has been replaced by a binary function

$$\pi(s_i,\theta) \triangleq [1-\theta;\theta](s_i,\theta) = (1-\theta)\mathbb{1}_{\{s_i=0\}}(s_i) + \theta\mathbb{1}_{\{s_i=1)}(s_i)$$

and an additional term $p_\Theta(\theta)$ appears.

Recall the graph $\mathcal{G}$ for this source (Fig. 4.2). If we simply augment the nodes $\mathcal{S} = \{\mathsf{s}_1,...,\mathsf{s}_n\}$ by a node Θ, connected to all of them via $\pi(s_i,\theta)$ as edge potential as replacement

[4]Using a mixture ML construction, [77] has shown that LDPC codes achieve the entropy universally for stationary memoryless sources in the limit of large n, so it hints at a lower complexity method using BP.

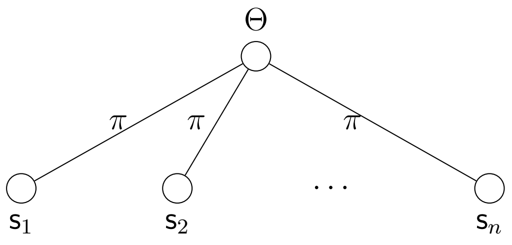


Figure 6.4: Augmented source subgraph $\mathcal{G}^{\triangle}$ for $\mathbf{Bern}(\theta)$, with $\pi = [1-\theta; \theta]$.

for node potential $\phi(s_i)$, then we have the *augmented graphical model* $\mathcal{G}^{\triangle}$ for (s^n, Θ) (Fig. 6.4). If we apply the same trick of combining $\mathcal{G}^{\triangle}$ with the code graph $\mathcal{C}$, we can run BP to optimize over the new objective

$$u^{\triangle}(s^n, \theta) \triangleq c(s^n) p_{\mathsf{s}^n\Theta}(s^n, \theta) \tag{6.12}$$

We will marginalize over both each s_i and Θ, but of course we only care about the former — the latter is a side effect.

Let us now write the update equations. Since Θ and s_i are different node types and message types differ (over $\mathbf{GF}(2)$ for s_i, potentially over $\mathbf{R}$ for Θ), we must use the edge potential π to couple them. The best way to understand it is to take the pairwise potentials as factor nodes explicitly (not drawn). We will also call these factor nodes $\pi \in \Pi$ and index them by their neighbors (i, θ). Thus, new messages $\mu^{i\leftarrow(i,\theta)}$, $\mu^{i\rightarrow(i,\theta)}$ and $\mu^{\theta\leftarrow(i,\theta)}$, $\mu^{\theta\rightarrow(i,\theta)}$ will pass along edges connecting Θ and s_i. $\mu^{i\leftarrow(i,\theta)}$ merely replaces ϕ^i, so its function is clear. $\mu^{i\rightarrow(i,\theta)}$ is trivial. So what remains is to state the conversion between s_i-type and Θ-type messages and the update at Θ. For completeness, we also list the remaining identities for this particular source:

Π node output (conversion):	$\mu^{\theta\leftarrow(i,\theta)} \Leftarrow \pi_i^\theta \mu^{i\rightarrow(i,\theta)}$
	$\mu^{i\leftarrow(i,\theta)} \Leftarrow \pi_\theta^i \mu^{\theta\rightarrow(i,\theta)}$
Θ node output:	$\mu^{\theta\rightarrow(i,\theta)} \Leftarrow \mu^{\theta\leftarrow(\sim i,\theta)}$
Θ belief computation (diagnostic):	$b^\theta = \mu^{\theta\leftarrow(*,\theta)}$
$\mathcal{S}$ node output:	$\mu^{i\rightarrow(i,\theta)} \Leftarrow \left[M^{i\leftarrow}\right]_{\mathcal{G}^{\triangle}}$
$\mathcal{S}$ external message output:	$\left[M^{i\rightarrow}\right]_{\mathcal{G}^{\triangle}} \Leftarrow \mu^{i\leftarrow(i,\theta)}$

Table 6.1: Node computations on the extended (top) and existing (bottom) parts of the augmented graph $\mathcal{G}^{\triangle}$.

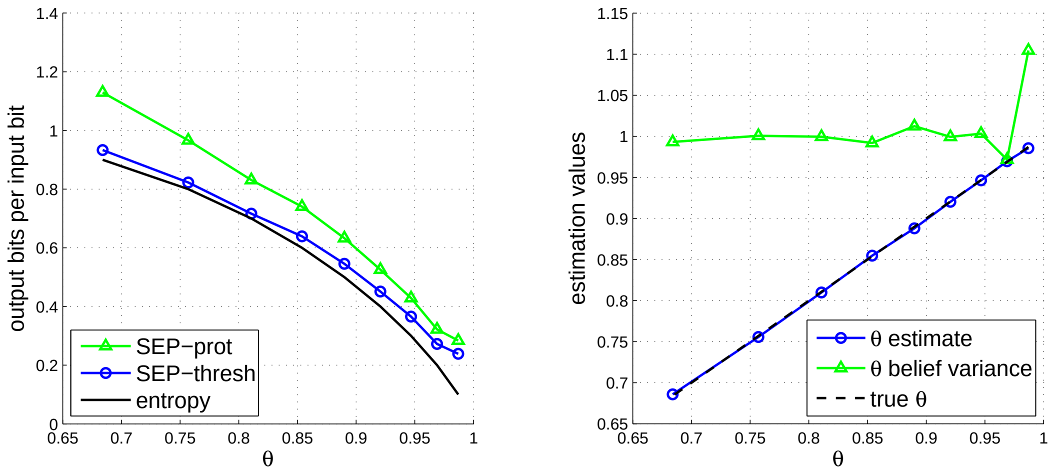


Figure 6.5: Performance of compressing $\mathbf{Bern}(\theta)$ with `SEP` and the augmented graphical model $\mathcal{G}^{\triangle}$ (left). Estimation diagnostic is shown, with estimation mean and normalized variance (right). $n = 1000$, $r_{\text{dope}} = 0.001$.

Note that we can always still have other independent priors, including on Θ itself, but we do not consider those cases here.

The function $\pi(s_i, \theta)$ is linear in $\theta \in [0, 1]$ and can be considered as a pair of kernels for estimating θ based on whether $s_i = 0$ or $s_i = 1$. The Θ update accumulates these estimates. Messages over θ can be represented in various ways, but since no more than a precision of $1/n$ can be obtained on θ, one possibility is to discretize the space, in which case $\pi(s_i, \theta)$ has tensor form,

$$\pi_i^\theta = \begin{bmatrix} 1 & 0 \\ \frac{n-1}{n} & \frac{1}{n} \\ \vdots & \frac{2}{n} \\ \frac{1}{n} & \vdots \\ 0 & 1 \end{bmatrix} (s_i, \theta) \tag{6.13}$$

6.4.3 Results

Fig. 6.5 shows the performance of compressing the $\mathbf{Bern}(\theta)$ source with this decoder. Since no initial beliefs are available, we must use doping, but we use the least possible — just 1 bit. We see a loss compared to Fig. 4.3, the case with a fully specified model, but the loss is slight except at the lowest rates.

If we are interested in the estimation of θ, we also have it, since we obtain a marginal belief b^θ on Θ for free. One should be convinced that the θ propagated in $\mu^{i \leftarrow (i,\theta)}$ messages

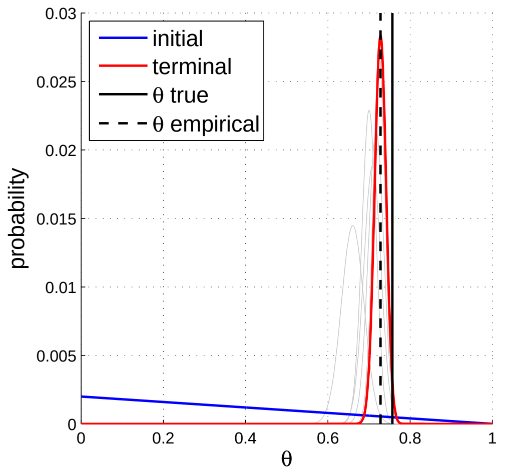


Figure 6.6: Convergence of Θ beliefs from the 1-bit doping initialization to the final form at decoding success (every 5 iterations plotted). The true $\theta \approx 0.7570$ differs from the empirical type-class of the sample (728 1's out of 1000).

to each s_i converges to the mean belief

$$E[b^\theta] \triangleq \int_{[0,1]} d\theta \; b^\theta(\theta)\theta$$

and that $E[b^\theta]$ and

$$Var[b^\theta] \triangleq \int_{[0,1]} d\theta \; b^\theta(\theta)(\theta - E[b^\theta])^2$$

are respectively the θ estimation and estimation error variance. Referring to Fig. 6.5 for the estimation diagnostic, the belief means appear highly accurate (n.b., when rate is insufficient, wild estimates are seen). The belief variance, once normalized by the theoretical error variance

$$Var[\hat{\theta}_{\text{unbiased}}] = \frac{1}{n}\theta(1-\theta)$$

of the unbiased estimator

$$\hat{\theta}_{\text{unbiased}} = \frac{1}{n}\sum_{i=1}^{n} s_i$$

is also nearly 1 for the entire range of θ, except at the lowest rates. Thus the joint estimation of θ with compression is optimal given the observed data, as expected.

Finally, we show the convergence process of b^θ in Fig. 6.6. It is notable that the estimate converges to the empirical θ of the sample type-class, not to the true θ. In some sense, the true θ is irrelevant, and this differs from what we would do when we had θ.

6.4.4 Discussion

The method of the augmented model can be extended to other parameter types in a source model. For example, we can make the same message updates with a parameter in an edge potential. Suppose we have a binary $\mathbf{Markov}(\theta)$ source with unknown transition probability θ. Then we can augment each edge potential $\psi_i(s_i, s_{i+1})$ by connecting to a shared parameter node Θ via a ternary factor $\pi(s_i, s_{i+1}, \theta) = p_{\mathsf{s}_i \mathsf{s}_{i+1}|\Theta}(s_i, s_{i+1}|\theta)$, i.e. $\pi(a, a, \theta) = 1 - \theta$ and $\pi(a, \bar{a}, \theta) = \theta$.

It can be extended also to offline or online learning from multiple samples. Suppose model parameters are stable across multiple samples (e.g. ergodic sources), then we can retain the final belief for parameters θ of a decoding run, and initialize the next run using those as prior. The excess rate required to code each subsequent sample is reduced.

Finally, this method is an adaptation of a variety of parameter learning methods over graphical models, e.g. [78, 79, 80] — which form a rich field in their own right — to the problem of joint compression and estimation. The added insight in the latter context is for the estimation to serve the specific interest of minimizing rate. In this sense, a deeper connection runs between compression and model identification.

Also related are more complex hierarchical data models (e.g. hidden Markov models), because parameters are indistinguishable from generic hidden variables, except for the fact that the parameter space is typically sub-exponential in n, while the space of hidden variables is not.

7 Architectural Generalization

While we have presented the model-code separation architecture and its archetypal `SEP` system as applied to binary lossless compression (Chapters 3 and 4), realistic compression scenarios often involve additional processes or constraints that make the system presented so far not immediately applicable. One such scenario is source data symbols belonging to a larger-than-binary alphabet. In this chapter, we extend `SEP` by means of an architectural generalization to handle this and potentially other elaborate situations that will be seen in Chapters 9 and 10.

7.1 Large-alphabet sources

The basic compression system of Chapter 3 assumes the data s^n and the code as represented by H are over the same field $\mathbf{S} = \mathbf{GF}(q)$. While the study of non-binary LDPC coding is progressing [81, 82], LDPC codes and decoding algorithms are still most well developed over $\mathbf{GF}(2)$. Furthermore, it may not be ideal to need to match the alphabet of each source with a tailored code. If we work with $\mathbf{GF}(2)$ codes, however, we need to handle large-alphabet data with care.

One traditional method is bit-planing, i.e. treating the bits representing a letter in a larger alphabet as independent $\mathbf{GF}(2)$ sources. However, this is inherently suboptimal, because it neglects the correlation that may exist between the bit planes:

Example 7.1. Suppose the alphabet is $\mathbf{S} = \{0, 1, 2, 3\}$, and we compress a 3-vector $s = (s_1, s_2, s_3)$ where $s_0 = 0$, and $s_{i+1} = s_i$ or $s_{i+1} = s_i + 1$ each with probability $1/2$. There are only 8 (equiprobable) sequences, namely $(0, 0, 0)$, $(0, 0, 1)$, $(0, 1, 1)$, $(0, 1, 2)$, $(1, 1, 1)$, $(1, 1, 2)$, $(1, 2, 2)$, $(1, 2, 3)$, thus the entropy is 3 bits. The first (most significant) bit plane has 4 sequences of $(0, 0, 0)$, 2 sequences of $(0, 0, 1)$, and 2 sequences of $(0, 1, 1)$, for an entropy of 1.5 bits. The second (least significant) bit plane has all 8 possible length-3 sequences, for an entropy of 3 bits. The total of coding the bit planes separately is at best 4.5 bits.

Are there ways to obtain better compression even when only using binary codes? It turns out there is:

Example 7.2. If instead we look at the three level sets of s, we find that each has a single $0 \to 1$ transition, the locations of all of which have a nesting relationship. The $l_0 = \{s > 0\}$

level set has 1 sequence of $(0,0,0)$, 1 sequence of $(0,0,1)$, 2 sequences of $(0,1,1)$, and 4 sequences of $(1,1,1)$. It can be coded with 1.75 bits. Without going through the details, conditioned on l_0, the $l_1 = \{s > 1\}$ level set can be coded with 1 bit, and conditioned on both l_0 and l_1, the $l_2 = \{s > 2\}$ level set can be coded with 0.25 bits. The total is 3 bits, which is the entropy of the source and cannot be beat.

We do not pursue this level-set idea further in this work, because there is an even greater philosophical motivation for dealing with other than $\mathbf{GF}(2)$ sources in a general way: sources may have a great deal of complexity in their internal symbol structure, and source models are defined over those symbols. Consider, for example, floating point numbers or relational database entries as source symbols. In real systems, even though s may be highly structured internally, it is often presented to the compression system after it has already been serialized into an opaque bit-stream z. If coding machinery is to be truly model-free, perhaps it should not assume the field structure or even the alphabet size of s, and simply work with z.

Next we describe how to compress for large-alphabet sources with model-code separation by inserting into the `SEP` decoder an additional subgraph that models how symbols of s are represented as z.

7.2 Representation and translation

Suppose $s^n = \{s_1, ..., s_n\}$ is an abstract n-symbol data sequence *serialized* by symbol-level *representational maps*. For ease of discussion, we assume all s_i belong to the same alphabet $\mathbf{S}$ of size M, and so there is one map for all n symbols, though this need not be the case. The representation map is a bijective function $t_{M\to q} : \mathbf{S} \to \mathbf{GF}(q)^B$ where $B \geq \log_q M$. For integer symbols s_i serialized into $\mathbf{GF}(2)$, this can be as simple as their machine representations, or other binary expansions like Gray-coding. Likewise, let $t_{M\to q} : \mathbf{S}^n \to \mathbf{GF}(q)^{nB}$ operate on an n-tuple symbol-by-symbol in the obvious way.

In probabilistic graphical models, symbols are inherently abstract, so we are also interested in some related functions for BP. When messages are passed to or from source nodes, there are related messages on their serialized representations. Define a pair of *message translation functions* $T_{M\to q} : (\mathbf{S} \to \mathbf{R}^+) \to (\mathbf{GF}(q) \to \mathbf{R}^+)^B$ and $T_{q\to M} : (\mathbf{GF}(q) \to \mathbf{R}^+)^B \to (\mathbf{S} \to \mathbf{R}^+)$ that convert between a message $m^{(M)}$ over $\mathbf{S}$ and a B-tuple of messages $m^{(q)} = m_1^{(q)}, ..., m_B^{(q)}$ over $\mathbf{GF}(q)$. Assuming messages are properly normalized probabilities, then for $\omega \in \{1, ..., B\}$ and $\beta \in \mathbf{GF}(q)$,

$$T_{M\to q}(m^{(M)})_\omega(\beta) = \sum_{\alpha \in \mathcal{S}} m^{(M)}(\alpha) \mathbf{1}\left\{t_{M\to q}(\alpha)_\omega = \beta\right\} \tag{7.1}$$

and for $\alpha \in \mathbf{S}$,

$$T_{q\to M}(m^{(q)})(\alpha) = \prod_{\omega=1}^{B} m_\omega^{(q)}(t_{M\to q}(\alpha)_\omega) \tag{7.2}$$

These conversions just state that $m^{(M)}$ is a product of marginals $m_1^{(q)}, ..., m_B^{(q)}$, which is not true in general, but we can still use them to do inference.

7.3 System construction

Now we describe the encoder and decoder constructions to compress abstract sources in the model-code separation architecture by using representation and translation.

7.3.1 Required inputs

Let $\mathrm{s}^n \in \mathbf{S}^n$ be an n-vector *source data* sequence presumed to be a random sample of s^n. To compress it, we require

- A code: an collection $\mathscr{H}(nB, k)$ of $k \times nB$ parity matrices of a rate k/nB LDPC code over $\mathbf{GF}(q)$;
- A data model: p_{s^n} in PGM representation;
- A representational map: $t_{M\to q}$.

7.3.2 Represented encoding

If the data is presented to the encoder as s^n, then serialize it first by

$$\mathrm{z}^{nB} = t_{M\to q}(\mathrm{s}^n) \tag{7.3}$$

Otherwise, assume z^{nB} is presented to the encoder.

Model-free coding takes place in the represented alphabet of $\mathbf{GF}(q)$, which can be naturally called the *coding domain*. Setting k to target $r_{\text{code}} = k/nB$, choose a random $H \in \mathscr{H}(nB, k)$, the compressed output is

$$\mathrm{x}^k = H\mathrm{z}^{nB} = Ht_{M\to q}(\mathrm{s}^n) \tag{7.4}$$

7.3.3 Translated decoding

Decoding takes into account the additional serialization step in (or prior to) the encoder.

Let $\mathcal{S} = \{\mathsf{s}_1, ..., \mathsf{s}_n\}$. By an abuse of notation, we will also relate graphical node identities by $t_{M\to q}$, so that

$$\{\mathsf{z}_{i,1}, \mathsf{z}_{i,2}, ..., \mathsf{z}_{i,B}\} \equiv t_{M\to q}(\mathsf{s}_i) \tag{7.5}$$

$$\mathcal{Z} \equiv t_{M\to q}(\mathcal{S}) = \bigcup_{\mathsf{s}\in\mathcal{S}} t_{M\to q}(\mathsf{s}) \tag{7.6}$$

are all well defined.

Recall in Section 3.2.3, the code subgraph $\mathcal{C} = (\mathcal{S} \cup \mathcal{X}, \mathcal{F})$ and the source subgraph $\mathcal{G}' = (\mathcal{S}, \Psi, \mathcal{E}')$ share the n source nodes $\mathcal{S}$ in $\mathcal{U}$. This is no longer the case here. Instead, the code subgraph is $\mathcal{C} = (\mathcal{Z}, \mathcal{X}, \mathcal{F})$, where $\mathcal{Z} = t_{M\to q}(\mathcal{S})$. The combined source-code graph $\mathcal{U} \triangleq (\mathcal{S} \cup \mathcal{Z}, \Psi \cup \mathcal{X}, \mathcal{E}' \cup \mathcal{F})$ has an additional layer (Fig. 7.1). The message passing strictly within each subgraph is still unchanged from Section 3.2.3.3, but whenever messages cross alphabet/representation boundaries they are translated. Refer to the inset labeled Algorithm 7.1 on specific modifications to the BP computations for a pairwise model.

Algorithm 7.1 Decoding with message translation for a pairwise model. The highlighted computations compute message translation.

Let $\mathcal{N}^{\mathcal{C}}_{i,\omega}$ denote the neighbors of $\mathsf{z}_{i,\omega} \equiv t_{M\to q}(\mathsf{s}_i)_\omega$ in $\mathcal{X}$,

- Source message update:

$$m^{(M)}_{i\to j}(s_j) \Leftarrow \sum_{s_i} \left[m^{(M)}_{\mathcal{C}\to i}(s_i)\right] \phi_i(s_i)\psi_{ij}(s_i, s_j) \prod_{\mathsf{s}_k \in \mathcal{N}^{\mathcal{G}}_i \setminus \mathsf{s}_j} m^{(M)}_{k\to i}(s_i) \tag{7.7}$$

  where

$$\boxed{m^{(M)}_{\mathcal{C}\to i}(s_i) = T_{q\to M}\left(\prod_{f_a \in \mathcal{N}^{\mathcal{C}}_{i,1}} m^{(q)}_{a\to i,1}(z_{i,1}), \ldots, \prod_{f_a \in \mathcal{N}^{\mathcal{C}}_{i,B}} m^{(q)}_{a\to i,B}(z_{i,B})\right)(s_i)} \tag{7.8}$$

- Code message update (source to factor):

$$m^{(q)}_{i,\omega\to a}(z_{i,\omega}) \Leftarrow \left[m^{(q)}_{\mathcal{G}\to i,\omega}(z_{i,\omega})\right] \prod_{f_b \in \mathcal{N}^{\mathcal{C}}_{i,\omega} \setminus f_a} m^{(q)}_{b\to i,\omega}(z_{i,\omega}) \tag{7.9}$$

  where

$$\boxed{m^{(q)}_{\mathcal{G}\to i,\omega}(z_{i,\omega}) = T_{M\to q}\left(\prod_{s_j \in \mathcal{N}^{\mathcal{G}}_i} m^{(M)}_{j\to i}(s_i)\phi_i(s_i)\right)_\omega (z_{i,\omega})} \tag{7.10}$$

  and (factor to source):

$$m^{(q)}_{a\to i,\omega}(z_{i,\omega}) \Leftarrow \sum_{\mathcal{N}^{\mathcal{C}}_a \setminus \mathsf{z}_{i,\omega}} f_a(\mathcal{N}^{\mathcal{C}}_a) \prod_{\mathsf{z}_{j,\omega'} \in \mathcal{N}^{\mathcal{C}}_a \setminus \mathsf{z}_{i,\omega}} m^{(q)}_{j,\omega'\to a}(z_{j,\omega'}) \tag{7.11}$$

- Belief update:

$$\hat{s}_i = \arg\max_{s_i} \left[m^{(M)}_{\mathcal{C}\to i}(s_i)\right] \left[\prod_{\mathsf{s}_j \in \mathcal{N}^{\mathcal{G}}_i} m^{(M)}_{j\to i}(s_i)\phi_i(s_i)\right] \tag{7.12}$$

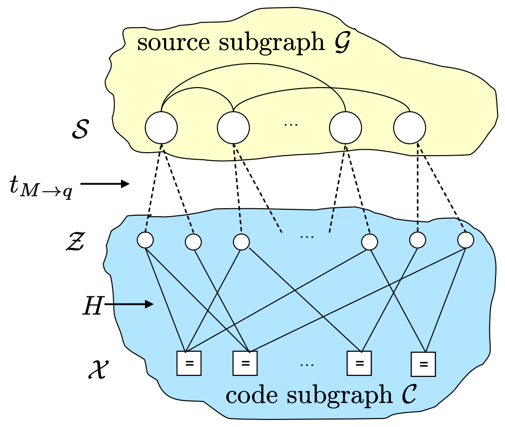


Figure 7.1: The structure of the decoder inference graph $\mathcal{U}$ for the system in Section 7. The illustrated model is pairwise, with factor nodes elided.

7.3.4 Doping symbols

The doping process can occur in either domain, but more naturally it occurs in the code domain unless the representational map is known to the encoder, e.g. it is the encoder that applies serialization. Whichever domain this takes place in, some nodes in $\mathcal{U}$ become observed variables with deterministic messages, e.g.

$$d^{(q)}(z_{\mathcal{D}'}) = \mathbb{1}\{z_{\mathcal{D}'} = \mathrm{z}_{\mathcal{D}'}\}(z_{\mathcal{D}'}) \tag{7.13}$$

and we obtain the equivalent effect in the other domain by message translation, e.g.

$$d^{(M)}(s_{\mathcal{D}}) = T_{q\to M}(d^{(q)}(z_{\mathcal{D}'} = 1)) \tag{7.14}$$

where $\mathcal{D}' = t_{M\to q}(\mathcal{D})$.

7.4 Modular decoder

The decoder with translation (Fig. 7.1) extends the architectural design of the decoder of Section 3.2.3.3. In the latter decoder (Fig. 3.2), we consider the source and code subgraphs each as modularly independent components exchanging external messages on virtual ports. This remains the case here. Indeed, the $\mathcal{G}$ and $\mathcal{C}$ components are unaware of what is on the other side of their interfaces.

Instead, the controller must be expanded to encompass both $\mathcal{S}$ and $\mathcal{Z}$, and a new layer representing message translation is inserted between them. An enlarged controller is one way to view the extension to the generalized decoder, but the message translation function too can be modularized into its own component (Fig. 7.2). Each representational map $t_{M\to q}$ defines a translator component, and it represents some additional prior knowledge about the relationship between the source symbols ($\mathcal{S}$) on which the $\mathcal{G}$ component is defined, and the

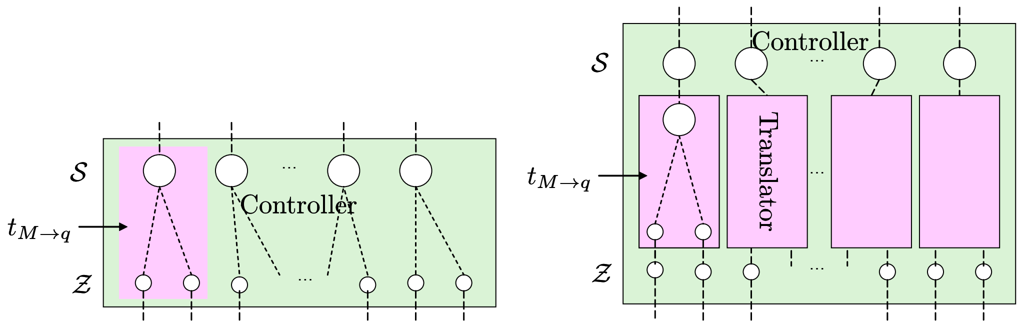


Figure 7.2: Message translation is functionally distinct (left), and it can be encapsulated into a separate component (i.e., not part of the controller), but interfacing with the controller (right).

"source symbols" ($\mathcal{Z}$) that the $\mathcal{C}$ component works with.

In some sense, we can begin to think of the controller as a central manager that holds all the relevant *state nodes* of importance to decoding, and that connects to a variety of components representing independent prior knowledge about the relationship between those nodes. This design is extensible to a greater number of processing situations than just for translating between alphabets. For instance,

- Multiple partial models representing independent insights about the data to be compressed can be applied in parallel as source components;
- Multiple codes representing independent constraints on the input can be applied in parallel as code components;
- Additional components can be applied in serial to represent a chain of processing stages from input data to compressed output, including e.g. concatenated codes and hierarchical data models.

Thus we have built a decoder architecture that is truly "process-replicating" and "inferential," terms which are first propounded in a narrower sense in the opening of Chapter 3.

7.5 Related ideas

Conversion of messages among different alphabets in a graphical setting is realized in a turbo coding system [83]. Interpretation of correlation and code constraints as functional blocks is found in the setting of compressing correlated i.i.d. streams [84].

8 Compressing Large-Alphabet Sources

Using the architectural generalization of Chapter 7, we can compress sources with larger-than-binary alphabets using binary codes.

8.1 Markov sources

Homogenous irreducible aperiodic Markov chains over a cyclic group $\mathbf{Z}_M$ are among the simplest large-alphabet sources with memory. Such sources $\mathsf{s}^n \in \mathbf{Z}_M^n$ have distribution

$$p_{\mathsf{s}^n}(s^n) = \tau(s_1) \prod_{i=2}^{n} \begin{bmatrix} q_{(1,1)} & q_{(1,2)} & \cdots & q_{(1,M)} \\ q_{(2,1)} & q_{(2,2)} & & q_{(2,M)} \\ \vdots & & \ddots & \\ q_{(M,1)} & q_{(M,2)} & & q_{(M,M)} \end{bmatrix} (s_{i-1}, s_i) \tag{8.1}$$

where $q_{(u,v)} \triangleq \mathbb{P}\{\mathsf{s}_i = u | \mathsf{s}_{i-1} = v\}$, the edge potential $\psi = [q_{(u,v)}] \triangleq \begin{bmatrix} q_1 & q_2 & \cdots & q_M \end{bmatrix}$ has the form of a stochastic transition matrix (columns summing to 1), and $\tau = \max \operatorname{eig}(\psi)$ is the unique stationary distribution. The *bitwise entropy rate* (e.g. after serializing) is

$$\mathbb{H}(\mathsf{z}) = \mathbb{H}(\mathsf{s}) / \log M = \sum_{s=1}^{M} \tau(s) \mathbb{h}(q_s) / \log M \tag{8.2}$$

If ψ is further doubly stochastic, then the chain is symmetric with $\tau = 1/M$.

8.1.1 MarkovZ

Here we specialize to a family of Wiener-process-like, smoothly transitioning Markov chains over $\mathbf{Z}_M$ by defining transition probabilities of ψ as discretized Gaussian densities of certain variances centered around the current state, i.e the self transition has the highest probability, followed by "nearer" neighbors:

$$q_{(m_{i+1}, m_i)} \propto \mathbb{P}\left\{ |m_{i+1} - m_i| - \frac{1}{2} < \mathsf{Z} < |m_{i+1} - m_i| + \frac{1}{2} \right\} \tag{8.3}$$

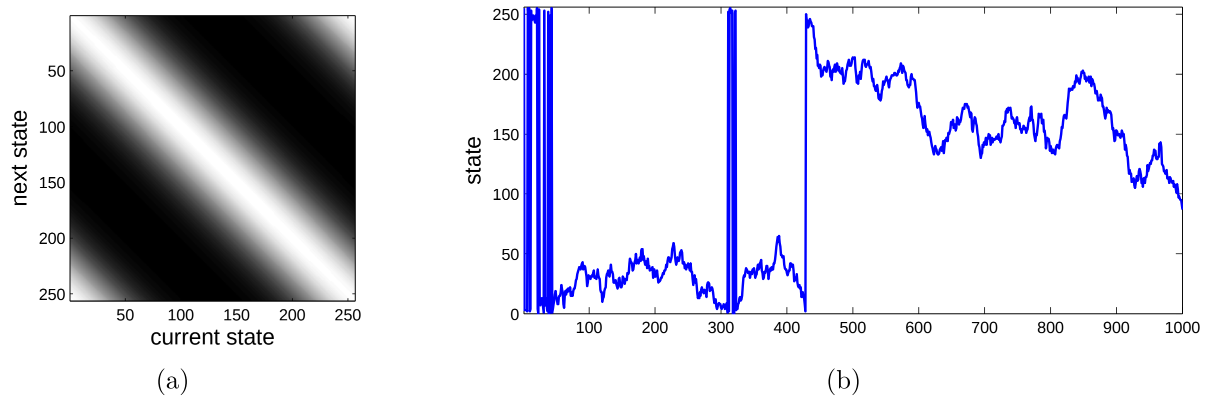


Figure 8.1: **MarkovZ**$[256](h)$ sources: (a) A transition matrix for $h = 0.9$ (brighter value is higher probability; the bright band would be narrower at lower entropies); (b) a sample path for $h = 0.5$.

Here, $|m_{i+1} - m_i| \triangleq \min\{|m_{i+1} - m_i|, |M - (m_{i+1} - m_i)|\}$ is understood cyclically (so ψ is symmetric circulant), and $\mathsf{Z} \sim \mathbf{N}(0, \Sigma)$ for some Σ.

We denote these sources by **MarkovZ**$[M](h)$, where h parameterizes the bitwise entropy rate of the source as adjusted via Σ (Fig. 8.1).

8.1.2 Results

We generate $\mathsf{s}^n \sim$ **MarkovZ**$[256](h)$ with length 1000, beginning at steady state. They are compressed by the system in Section 7, using the Gray-coding representational map. A range of entropy rates are targeted. There are 20 trials for each entropy rate. We compare `SEP` with `GZIP` and `CTW`, with the same compressor setup as in Section 4; here, `GZIP` and `CTW` compress the $\mathbf{Z}_{256}$ symbols as a stream of bytes in their canonical representation.

We see in Fig. 8.2 that the `SEP` performance is very close to the entropy lower bound, especially when accounting for the BP threshold rates in `SEP-thresh`. `GZIP` performance is poor, and does not improve substantially even when sample length is increased to 100,000. `CTW` is somewhat better, and achieves fairly good performance at the longer sample length.

Unlike in Section 4.3, there is relatively little performance loss at lower entropies compared to at higher entropies. To further demonstrate this, Fig. 8.3 shows the compression of this source at various bit depths, including the binary case ($B = 1$) which can also be found in Fig. 5.2, and the $\mathbf{Z}_{256}$ case ($B = 8$) just described. We see rapidly improved performance at moderate bit depths, even as we have not optimized for the doping rate, which remains at $r_{\text{dope}} = 0.04$ throughout.

This is also a positive result in that, it shows the non-bijective message translation between different alphabets is not an apparent hindrance to performance, at least for this type of

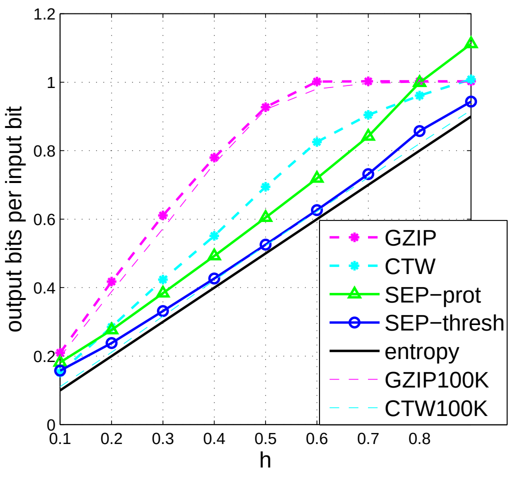


Figure 8.2: Compressing $\mathbf{MarkovZ}[256](h)$ by `SEP` and other systems, compared to source entropy. $r_{\text{dope}} = 0.04$. For all systems, $n = 1000$ are plotted; for `GZIP` and `CTW`, $n = 100,000$ are also given.

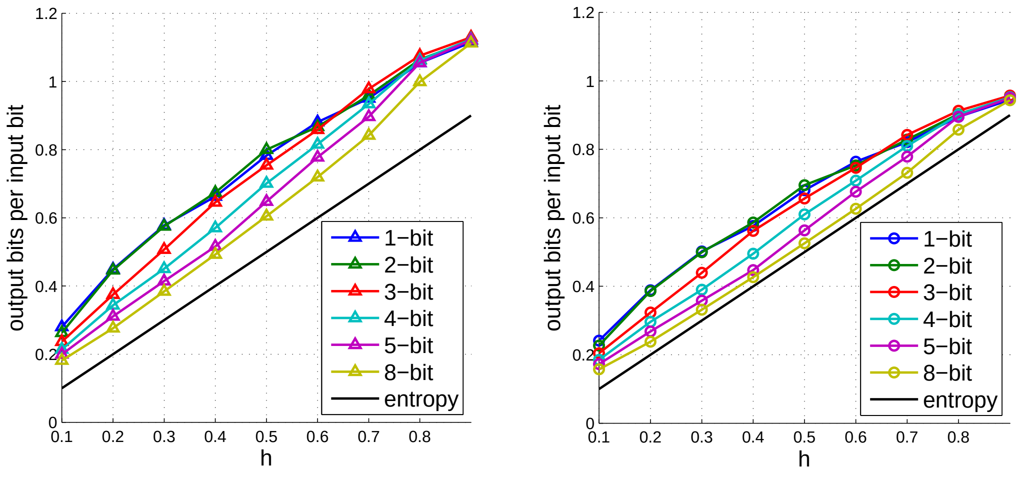


Figure 8.3: Compressing $\mathbf{MarkovZ}[2^B](h)$ for $B = 1, 2, 3, 4, 5, 8$. $r_{\text{dope}} = 0.04$, $n = 1000$. On the left is the performance of `SEP-prot`; on the right, that of `SEP-thresh`.

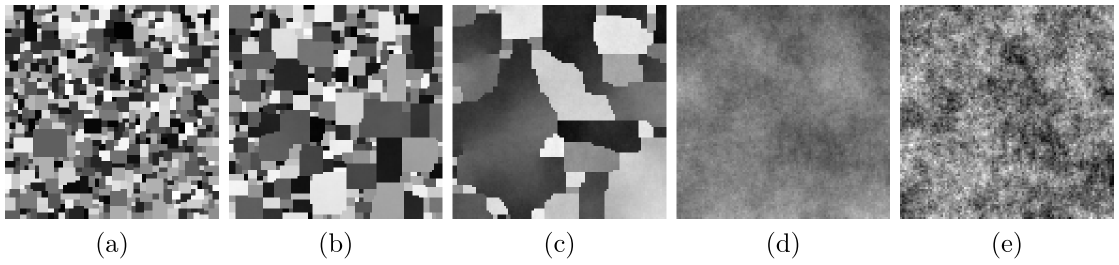


Figure 8.4: 100×100 Gibbs sampled images of $\mathbf{PottsZ}[256](h)$. From left to right, $h = 0.1, 0.3, 0.5, 0.7, 0.9$.

benign serialization. It should also be pointed out that this architecture allows the retention of the full data model at the alphabet of the source in a way that would be impossible with such methods as the independent coding of bit planes.

8.2 Potts model

The homogeneous Markov model can be extended to two dimensions, giving rise to the large-alphabet version of the homogeneous Ising model, or the Potts model [85]. Over an $h \times w$ lattice graph $\mathcal{G} = (\mathcal{S}, \mathcal{E})$, it has distribution

$$p_{\mathsf{s}^n}(s^n) = \frac{1}{Z} \prod_{i \in \mathcal{S}} [p_1; p_2; ...; p_M](s_i) \prod_{(i,j) \in \mathcal{E}} \begin{bmatrix} q_{(1,1)} & q_{(1,2)} & \cdots & q_{(1,M)} \\ q_{(2,1)} & q_{(2,2)} & & q_{(2,M)} \\ \vdots & & \ddots & \\ q_{(M,1)} & q_{(M,2)} & & q_{(M,M)} \end{bmatrix} (s_i, s_j) \tag{8.4}$$

where $\phi = [p_1; p_2; ...; p_M]$ and $\psi = [q_{(u,v)}]$ is arbitrary.

Unfortunately, we do not have an entropy rate for the Potts model.

8.2.1 PottsZ

Likewise, we can define a family of Potts models over $\mathbf{Z}_M$ with varying degrees of neighbor affinity. Put $\phi = 1$ and ψ as in Eq. 8.3, except that now $|m_{i+1} - m_i|$ is understood normally (so ψ is symmetric Toeplitz). This condition ensures that there is no wrapping around extremal values of $\mathbf{Z}_M$.

We denote these sources by $\mathbf{PottsZ}[M](h)$, where h parameterizes the bitwise entropy rate of the Markov source with the same ψ. This is just a convenient notation, and does not imply an entropy rate for this source.

Fig. 8.4 shows some examples for $\mathbf{PottsZ}[256](h)$. As h increases, the size of bounded domains of similar color, and the amount of color variation within a domain both increase. At small h, there are small patches of solid color, while at large h, there is a single noisy patch. At intermediate h, domain size is such that the color variation within them connect with neighbors so that boundaries are unclear. For values of h around this phase transition, $\mathbf{PottsZ}[256](h)$ resembles natural images, so can be used as a grayscale image model.

8.2.2 Results

We obtain Gibbs sampled images from this source and compress them with `SEP`, using the Gray-coding representational map. We compare against a few state-of-the-art image compressors:[1]

- `SPIHT`: A wavelets domain system ("Set partitioning in hierarchical trees") [86], that exploits similarities across subbands. We operate it at full rate for lossless mode.
- `J2K-LS`: The JPEG2000 compressor (ITU-T T.800; 2002) operated in lossless mode, employing reversible integer-to-integer wavelet transforms. Output length is the compressed file size, less the compressed file size of a 1-pixel image to account for headers.
- `BMF`: A proprietary lossless image coder that has consistently outperformed most compressors of its class in recent years, and is believed to be based on PPM (prediction by partial matching).

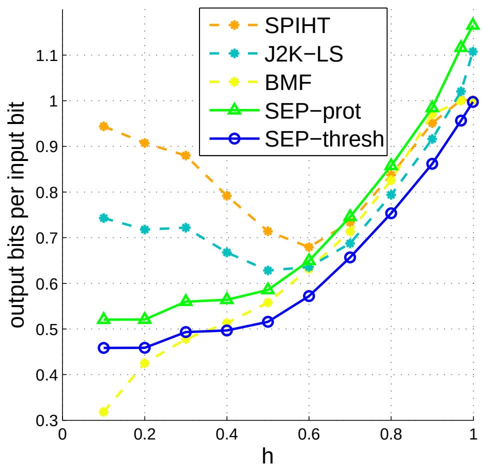


Figure 8.5: Compression performance for the $\mathbf{PottsZ}[256](h)$ source family. $(h, w) = (25, 25)$, $r_{\text{dope}} = 0.1$.

Fig. 8.5 shows the result. Since there is no known entropy rate to compare to, we do not know how close to the best performance we have come to. The Potts model is known to have positive "ground-state" entropy, as can also be observed from the samples in Fig. 8.4,

[1] `SPIHT` is found at `http://www.cipr.rpi.edu/research/SPIHT/` (v1.0); `J2K-LS` is implemented in MATLAB's Image Processing Toolbox (v8.3) as `imwrite`; `BMF` is found at `http://www.compression.ru/ds/` (v2.0).

thus we also do not have exhibits of images of very low rate. Nevertheless, the competitive performance of all systems across h is intriguing. While all compressors perform well above the phase threshold with edgeless images, `SPIHT` and `J2K-LS` are unable to compress solid-color blocky samples below the threshold well.

On the other hand, `SEP` performs fairly well across the entire range of h values. It is worthy to note that even at the higher rates, `SEP` is able to distinguish between the subtle model features of $\mathbf{PottsZ}[M](h)$ images and truly random images, allowing it to obtain compression gain even as the other compressors saturate at full rate.

Another example using this model is discussed in Section 10.2.3.

8.3 Summary

The model-code separation architecture embodying a message translator component can compress large-alphabet sources well, despite the additional layer within the message-passing path in the decoder. This is very encouraging because it means even as we model our data fully in its native alphabet, for purposes of coding, we only need to keep a library of good binary codes.

9 Lossy Compression

When the typical set of s^n is too large to represent losslessly (e.g. non-finite in size), or we are unwilling to expend enough bits to represent each of its elements distinctly, we resort to a non-invertible (e.g. many-to-one) mapping from each source data $s^n \in \mathbf{S}^n$ to a reproduction output $\hat{s}^n \in \mathbf{S}^n$, where generally $s^n \neq \hat{s}^n$. The mapping is evaluated under a *distortion measure* $d_n : \mathbf{S}^n \times \mathbf{S}^n \to \mathbf{R}_{\geq 0}$ to assess its average *distortion level*

$$\bar{d}_n \triangleq \mathbb{E} d_n(\mathsf{s}^n, \hat{\mathsf{s}}^n) \tag{9.1}$$

For a given rate budget, we want to keep the distortion level low, and vice versa.

Systems that fall under Domain III of Table 1.1 implement for this scenario. They are traditionally designed using a joint architecture (Fig. 9.1) just like their lossless counterparts (Fig. 1.3). With the latter, we know that a model-code separation architecture enables systemic flexibility with negligible effect on performance. Can an analogous separation architecture be feasible for lossy compression? We show that this is so, if we can leave geometry information implied by the distortion measure in the encoder.

Now let us again begin by seeking architectural insight from source coding theory.

9.1 Rate-distortion theory*

In *rate-distortion theory [87, 72]*, as in lossless compression (Section 2.1), a string $s^n = s_1, ..., s_n$ is considered as drawn from a probabilistic source $\mathsf{s}^n \sim p_{\mathsf{s}^n}$, but now the alphabet $\mathbf{S}$ is arbitrary.

The theory defines the *rate-distortion function* as

$$\mathbb{R}(\Delta; \mathsf{s}) \triangleq \lim_{n\to\infty} \frac{1}{n} \mathbb{R}(\Delta; \mathsf{s}^n) = \lim_{n\to\infty} \frac{1}{n} \left(\inf_{p_{\hat{\mathsf{s}}^n|\mathsf{s}^n} : \bar{d}_n \leq \Delta} \mathbb{I}(\mathsf{s}^n; \hat{\mathsf{s}}^n) \right) \tag{9.2}$$

whose value lower bounds the average number of bits required to represent each symbol of s^n to within average distortion level Δ, the *tolerance*.[1]

[1] $\mathbb{I}(\mathsf{x}; \mathsf{y}) = D(p_{\mathsf{xy}} || p_{\mathsf{x}} p_{\mathsf{y}}) = \mathbb{E} \log \frac{p_{\mathsf{xy}}(\mathsf{x},\mathsf{y})}{p_{\mathsf{x}}(\mathsf{x}) p_{\mathsf{y}}(\mathsf{y})}$. Notationally, $\mathbb{R}(\Delta; \mathsf{s})$ is more commonly written as $R(\Delta)$ with the source s understood.

The theory holds for (1) stationary ergodic sources of bounded total distortion, i.e. $\mathbb{E} d_n(\mathsf{s}^n, \hat{s}^n) < \infty$, and (2) finite-span distortion measures $d_n(s^n, \hat{s}^n) = (n - g + 1)^{-1} \sum_{k=0}^{n-g} d(s_{k+1,...k+g}, \hat{s}_{k+1,...,k+g})$, where

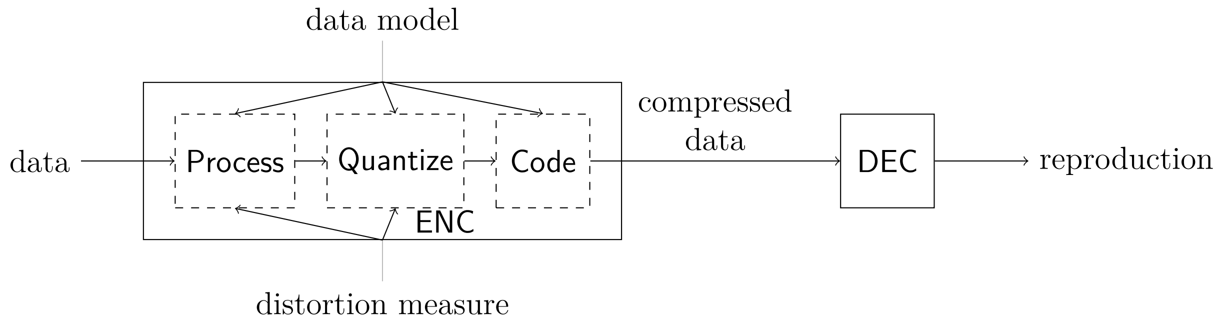


Figure 9.1: A common joint architecture for lossy compression systems. The data model along with the distortion measure are used to design Process, Quantize, and Code. (Confusingly, the latter two units of the encoder are sometimes together called the *quantizer*, but we reserve that term only for Quantize.)

The distribution $p^*_{\hat{\mathsf{s}}^n}$ that "achieves" the infimum in Eq. 9.2 is called the *optimal reproduction distribution*, while the neighborhood $\mathcal{B}_{\Delta+\epsilon}(\hat{s}^n) = \{s^n \in \mathbf{S}^n : d_n(s^n, \hat{s}^n) < \Delta + \epsilon\}$ is called the *tolerance ball* of $\hat{s}^n$.

9.1.1 Achievability

Analogously to lossless coding, a *random codebook scheme* can approach this bound arbitrarily closely, by building a reproduction codebook $\mathscr{Q}_n$ of $2^{n(\mathbb{R}(\Delta;\mathsf{s})+\epsilon)}$ entries, filled with random draws of $\hat{\mathsf{s}}^n \sim p^*_{\hat{\mathsf{s}}^n}$. For large n, all the typical sequences of s^n are found within a tolerance ball of a codebook entry, with the remaining ones contributing negligible distortion. Thus essentially any sequence from s^n can be compressed to a Δ-approximation using the $\mathbb{R}(\Delta;\mathsf{s})$ bits per symbols it takes to specify a codebook entry label.

On the other hand, there is no obvious late-binding system (with respect to the data model) analogous to the *random binning scheme* for lossless compression. The closest analog is known as Wyner-Ziv coding [88, 89], in which the decoder holds additional side information $\mathsf{y}^n \in \mathbf{S}^n$ unavailable to the encoder. In this scheme, the encoder first builds a reproduction codebook $\mathscr{Q}_n$ of $2^{\mathbb{I}(\mathsf{s}^n;\hat{\mathsf{s}}^n)+n\epsilon_1}$ entries, filled with random draws of some $\hat{\mathsf{s}}^n$. It then hashes all the entries of $\mathscr{Q}_n$ to $2^{\mathbb{I}(\mathsf{s}^n;\hat{\mathsf{s}}^n)-\mathbb{I}(\hat{\mathsf{s}}^n;\mathsf{y}^n)+n\epsilon_2}$ bins as in lossless Slepian-Wolf coding. If $\hat{\mathsf{s}}^n$ is distributed according to $p^{\mathrm{WZ}}_{\hat{\mathsf{s}}^n}$, the marginal that "achieves" the infimum in

$$\mathbb{R}^{\mathrm{WZ}}(\Delta;\mathsf{s}|\mathsf{y}) \triangleq \lim_{n\to\infty} \frac{1}{n} \left(\inf_{p_{\hat{\mathsf{s}}^n|\mathsf{s}^n}:\bar{d}_n \le \Delta} [\mathbb{I}(\mathsf{s}^n;\hat{\mathsf{s}}^n) - \mathbb{I}(\hat{\mathsf{s}}^n;\mathsf{y}^n)] \right) \tag{9.3}$$

then for large n and a typical y^n, each bin will on average contain no more than one codebook entry $\hat{s}^n \in \mathscr{Q}_n$ typical of $\hat{\mathsf{s}}^n|\mathsf{y}^n = y^n$, and each typical sequence of $\mathsf{s}^n|\mathsf{y}^n = y^n$ will in turn be within a tolerance ball of such an $\hat{s}^n$. In this way, essentially all sequences from s^n are compressed to a Δ-approximation using the $\mathbb{R}^{\mathrm{WZ}}(\Delta;\mathsf{s}|\mathsf{y})$ bits per symbol it takes to specify a

$d : \mathbf{S}^g \times \mathbf{S}^g \to \mathbf{R}_{\geq 0}$.

bin label. In significant contrast to lossless compression, $\mathbb{R}^{\mathrm{WZ}}(\Delta; \mathsf{s}|\mathsf{y}) > \mathbb{R}(\Delta; \mathsf{s}|\mathsf{y})$ generally, though equality is possible, e.g. with jointly Gaussian $(\mathsf{s}^n, \mathsf{y}^n)$ and mean-squared error (MSE) distortion measure [90].

9.1.2 Applied system design

The main system design insight of the foregoing is with respect to the role of the reproduction codebook $\mathscr{Q}_n$. In lossy compression, two competing criteria are imposed on $\mathscr{Q}_n$ — it must be, at the same time, sufficiently space-filling to obtain low distortion, and sufficiently similar to s^n in composition to save on rate. The optimal reproduction distribution $p^*_{\hat{\mathsf{s}}^n}$ is the one that fills space in such a way that the elements of $\mathscr{Q}_n$ are hit with equal probability, so that no residual entropy is left in the output.

The random codebook scheme encodes directly using such a codebook, having generated it with the full knowledge of the distortion measure and the data model in an early-binding design. Its practical realizations are the high-performance vector quantizers whose encoders are designed by optimization or sample training methods (e.g. Lloyd-Max [91, 92, 93], clustering [94, 95], neural networks [96, 97]).

In Wyner-Ziv coding, we glimpse a different design in which $\mathscr{Q}_n$ is an *intermediate codebook* that does not result in equiprobable usage if we consider y^n to be a parameter in the "full" data model for s^n, e.g. $p(\mathsf{s}; \theta) = p_{\mathsf{s}^n|\mathsf{y}^n}(\cdot|\theta)$. The random binning that follows can then be understood as an entropy coding stage. In this design, the distortion measure is still fully known at the encoder to design $\mathscr{Q}_n$, while the data model is only partially known at the encoder, and fully known at the decoder. This is not a true late-binding system, but highly suggestive of one in which the encoder entirely divests itself of the data model. However, the Wyner-Ziv result says that we should expect a performance penalty for such separation, because even a model-poor (cf. model-free) encoder results in a fundamental rate gap.

9.2 Separation architecture

The key to model-code separation in lossless compression is the model-free hashing encoder (Section 3.2.2) that uses no prior knowledge. Since lossy compression systems involve two sources of compression gain, (1) conflating multiple source sequences as one (*quantization*), and (2) exploiting the data model (*entropy removal*), naturally there are two sources of prior knowledge, respectively, (1) the distortion measure (*geometry*) and (2) the data model (*typicality*). Should we treat the two types of knowledge similarly or differently?

9.2.1 A naive hashing encoder

A first attempt at separation is to treat them similarly, in which case we use the same hashing encoder as in lossless compression to reduce s^n to $n\mathbb{R}(\Delta, \mathsf{s})$ bits, then hope to apply both the distortion measure and the data model subsequently. This turns out to be impossible for the distortion measure.

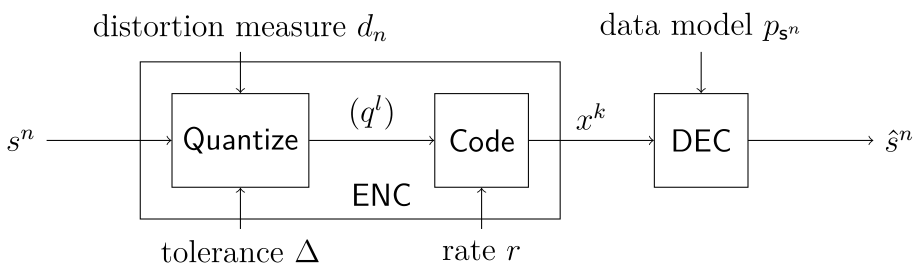


Figure 9.2: System diagram for a lossy compression architecture featuring model-quantizer separation and model-code separation. The encoder is model-free but distortion measure aware.

To see this, take the example of a finite-alphabet, bounded total distortion source s^n. Its typical set has roughly $2^{n\mathbb{H}(\mathsf{s})}$ elements. Suppose we created some $2^{n\mathbb{R}(\Delta;\mathsf{s})}$ bins according to a random binning assignment, then into each bin will fall on average $N \doteq 2^{n(\mathbb{H}(\mathsf{s})-\mathbb{R}(\Delta;\mathsf{s}))}$ random elements of the typical set. For $\Delta > 0$, $\mathbb{R}(\Delta;\mathsf{s}) < \mathbb{H}(\mathsf{s})$ by the convexity of $\mathbb{R}(\Delta;\mathsf{s})$, so this number is exponentially large in n. If we code s^n to $n\mathbb{R}(\Delta;\mathsf{s})$ bits by random binning, the best we can do at the decoder is to arbitrarily select one of the N elements as $\hat{s}^n$. Since the N elements constitute a random sampling of the typical set, the expected distortion level over all binning assignments is just the zero-rate distortion of s^n; in other words, rather than obtain a lower (and optimal) distortion level of Δ by spending $n\mathbb{R}(\Delta;\mathsf{s})$ bits, we obtain the maximum distortion level as if we spent no rate at all.

The fatal flaw of random binning as coding for lossy compression is that it is *geometry-free*: an element coded into a bin permanently loses its spatial identity (beyond what is known about the typical set as a whole), and quantization gain is lost. What we learn here is that any worthwhile reductive stage for lossy compression must be *locality sensitive*; or put another way, it must contain an element of quantization in the presence of the distortion measure.

9.2.2 Model-quantizer separation

Since the two types of knowledge are inherently different, we need a more nuanced separation design. Recall the lossless compression architecture of Fig. 3.1. In Section 9.2.1, we argue that the functional replacement for Code in lossy compression must be quantization-like, and the distortion measure must accompany it within the encoder to make geometry-aware decisions. In Section 9.1.2, we learn that the data model however may be severable from the encoder (though with a potential loss). Thus in addition to model-code separation, another separation principle emerges in lossy compression between the geometry processing using the distortion measure and the typicality processing using the data model. We call it *model-quantizer separation*.

Additionally, we learn from Wyner-Ziv coding that the encoder may be implementable (if desired) as a *model-free quantization* stage followed by a model-free (entropy) coding stage,

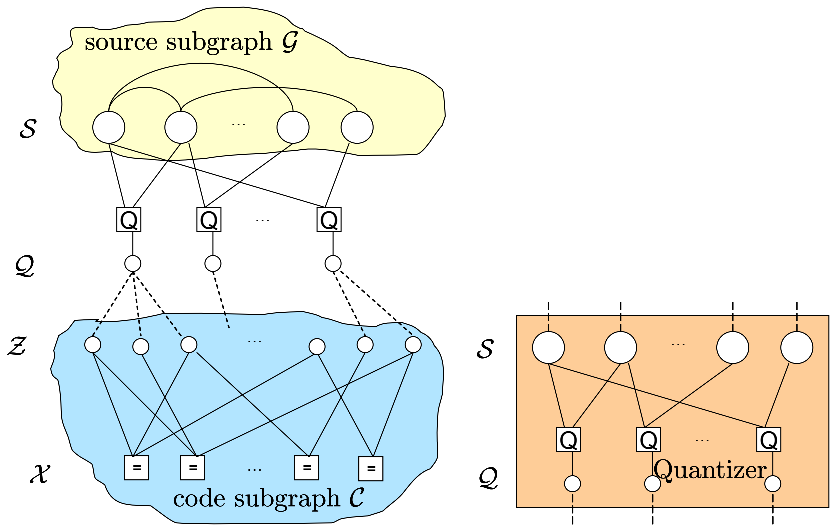


Figure 9.3: The structure of the decoder inference graph (left), and the modularized quantizer component (right) for separation architecture lossy compression. This left drawing assumes the full complement of transformations are required, including alphabet translation and entropy coding — these stages may be omitted in some settings.

the latter being the same hashing unit as for lossless coding (because it is exactly that). Let us therefore propose the architecture shown in Fig. 9.2, with the modeling, coding, and quantization aspects of lossy compression all treated separately. By design, it shares the same type of systemic flexibility as its lossless counterpart with respect to the data model, and we can even reuse the coding components from lossless compression.

Remark 9.1. The design principle of separation in lossy compression has been reflected in the lineage of "practical Wyner-Ziv" systems, particularly in video compression [98, 35, 99, 100, 101], where partial model separation from the encoder in the spirit of side-information problems (Section 3.4.1) provides scalability and other desirable systemic flexibility.

To fully address the inquiry that opens the chapter though, we need to show designs for low-complexity model-free quantization, and study how the architecture affects system performance.

9.3 Model-free quantization

The important practical question for model-free quantization design is how to represent them in an inferential decoder. Specifically, the decoder must take a form like that of Fig. 9.3, with an additional graphical layer inserted to represent encoder quantization processing. The $\boxed{\mathsf{Q}}$ factors, similar to $\boxed{=}$ factors of coding, stand for constraints, in this case relating input $\mathcal{S}$ to quantized output $\mathcal{Q}$ via the Quantize operation. We show how this may be done for a number of quantizer designs.

9.3.1 Existing quantizers

If we wish to use existing encoder quantizer components, we usually can. Because current systems (Fig. 9.1) rely heavily on Process and Code to exploit the data model,[2] leaving relatively little to bind to Quantize, their quantizers may already be model-free.[3]

Here are a few typical existing quantizers and how inferential decoders can be designed for them:

- *Traditional quantizer*: We can use an off-the-shelf traditional quantizer as a black-box component, as long as the statistics of their output $q^l \in \mathbf{Q}^l$ can be characterized, e.g.

$$p_{\mathsf{q}^l}(q^l) = \int_{\mathsf{Quantize}^{-1}(q^l)} p_{\mathsf{s}^n}(s^n)$$

  The output q^l can be taken as the input of an ordinary lossless compression problem. p_{q^l} is likely to be graphically representable with similar complexity as p_{s^n} if the quantizer is structured in a way that is spatially faithful (e.g. scalar quantizer, lattice quantizer with $l = n$). On the other hand, the output of an unstructured quantizer providing a single index $q \in \mathbf{Q}$ is less helpful; in that case, we may need to model the internals of the quantizer in the decoder, with messages crossing the quantizer processing boundary translated as appropriate. Complexity can be high.

- *Coded quantizer*: Certain low-complexity data structures such as trees or linear codes (e.g. LDGM, LDPC quantizers) with innate graphical representations are nowadays used to represent output spaces of quantizers. For such quantizers, we can incorporate their representations into the decoding graph with relatively little effort. (See Section 9.3.1.1.)

- *Geometric hashing*: Geometry-preserving hashing based on random projections is used in approximate nearest-neighbor queries and semantic clustering. They have a certain appeal for the model-quantizer separation architecture, as a direct replacement for the random binning hash. They are largely representable by graphical methods in concert with scalar quantizers. (See Section 9.3.1.2.)

9.3.1.1 Coded quantization*

A number of authors have looked at coded quantization using structured (e.g. linear) codes $\mathbf{L} = \{v : v = Gu\}$ as the reproduction set combined with a low-complexity (possibly

[2]The mirrors the lossless compression case, but with lossy compression, the pre-processing stage also exploits the distortion measure to perform a large amount of implicit quantization, i.e., various psycho-perceptual data models are in fact a data model coupled with a distortion measure.

[3]Quantizers for speech [102, 103] and to some extent, audio [104, 105, 106], bind weakly to the data model by training on model parameters, while quantizers for images [107] and videos [108] are essentially model-free, i.e. purely geometric. The reason is that model-free quantizers are easier to design and use than quantizers strongly bound to specific data models, even in joint architecture systems.

approximate) nearest-neighbor search. These are embedding methods that seek for an input s a u that generates the optimal embedding, where

$$u^* = \arg\min_u d(s, Gu)$$

is the quantized output. Of the most well developed are works using sparse linear codes and encoding with some stage of iterative message-passing:

- Using the output space of a low-density generator matrix (LDGM) directly as the reproduction set [109, 42, 110]. This achieves the rate-distortion bound for the $\mathbf{Bern}(\frac{1}{2})$ source with erasure distortion.

- Modifying the output space by additional constraints such as low-density parity-check (LDPC) constraints in compound LDGM-LDPC constructions [111, 112]. This achieves the rate-distortion bound for the $\mathbf{Bern}(\frac{1}{2})$ source with Hamming distortion.

- Non-linear modifications to weakly sparse generator-matrix embeddings [113]. This achieves the rate-distortion bound for uniform i.i.d. $\mathbf{GF}(q)$ sources with Hamming distortion.

- Non-uniform $\mathbf{Bern}(p)$ sources with Hamming distortion can be compressed with $\mathbf{GF}(q)$-quantized LDGM codes to somewhat better than the Ancheta bound (Section 9.6) [114]. With multi-level quantization [115], the rate-distortion bound can be achieved for arbitrary DMS's with separable and bounded distortion measure. However, both of these constructions require knowledge of the optimal reproduction distribution at the quantizer.

These results are promising, but direct coded quantization onto linear subspaces gives poor coverage of the space except over the typical set of uniform sources. If we strive for model-quantizer separation and must settle for certain suboptimal reproduction codebooks, we need to know the performance of such systems, which we examine in Section 9.4.

9.3.1.2 Geometric hashing*

In recent years, a number of locality-sensitive hashing (LSH) schemes have been developed for nearest-neighbor query problems, e.g. [116, 117, 118, 119]. These all suggest some form of random projection followed by low-complexity (i.e. scalar, or scalar binary) quantization, and because they are not targeted at general compression problems, they happen to be model-free.

In the case of the binary Hamming distortion, the proposed LSH is a collection of 1-bit coordinate projections, so an l-bit hash is given simply by $\mathcal{H}_l(s^n) = (s_{j_1}, ..., s_{j_l})$ for some random selection of $j_1, ..., j_l \in \{1, ..., n\}$. Using such an LSH as quantizer implies losslessly coding a subset of the input bits. This indeed is compatible with model-quantizer separation.

However, note that the processing is entirely linear, thus performance over Bernoulli i.i.d. sources is limited by the Ancheta bound (Section 9.6), though for sources with memory, this situation can be somewhat ameliorated.

A more fundamental problem with the LSH is that, due to the small alphabet size, rectilinear halfspaces of $\mathbf{GF}(2)^n$ do not have the geometry to generate a hashing region that approaches a Hamming Δ-ball as the hash collection grows. (To contrast, for the MSE distortion, the proposed LSH is a collection of scalar-quantized 1D projections, and that hashed region does approach an MSE Δ-ball as the collection size grows.) Taking this insight further, we propose a scheme to build a geometric hashing quantizer that operates on more than one dimension of $\mathbf{GF}(2)^n$ at a time. This structure both allows to take advantage of Hamming geometry and begins to approach a true vector quantizer in the limit of large hash collection.

9.3.2 Low-density hashing quantizer

On the other hand, beginning with the inferential decoder design in mind, we can propose a new quantizer design that is more directly suitable than existing quantizers in some combination of ease of implementation, complexity, and performance.

Referring to Fig. 9.3, a full vector quantizer would involve one $\boxed{\mathsf{Q}}$ factor that connects to all $\mathcal{S}$ nodes, and producing one large-alphabet output $\{\mathsf{q}\} = \mathcal{Q}$. However, since we would follow that with an alphabet translator to take $\mathcal{Q}$ to binary symbols $\mathcal{Z}$, we can consider the vector quantizer to be equivalently decomposed into a multitude of bitwise $\boxed{\mathsf{Q}}$ factors each producing one bit $\mathsf{q}_i \in \mathcal{Q}$. Each $\boxed{\mathsf{Q}}$ now acts like a one-bit quantizer or hash. For $\boxed{\mathsf{Q}}$ implementing arbitrary functions, the complexity can still be exceedingly high, given its $\mathcal{O}(n)$ neighborhood — a full vector quantizer still has high complexity even to produce 1 bit.

A natural reduction is to connect each $\boxed{\mathsf{Q}}$ not to all the $\mathcal{S}$ nodes but only to some, giving rise to a product quantizer structure [120]. If the connection is sparse enough, the collection of bitwise quantizers may be called a *low-density hashing quantizer (LDHQ)*.

For the rest of this section, we design an LDHQ-supported separation architecture lossy compression system for the binary Hamming case.

9.3.2.1 Quantization function

Each bitwise quantizer $\boxed{\mathsf{Q}}$ implements a specific quantization function, defined next.

Connect $\boxed{\mathsf{Q}}$ to a random δ nodes of $\mathcal{S}$: $\{\mathsf{s}_{j_1}, \mathsf{s}_{j_2}, ..., \mathsf{s}_{j_\delta}\} \subseteq \mathcal{S}$, so let $s = s_{j_1}, s_{j_2}, ..., s_{j_\delta}$. Let $u \in \mathbf{S}^\delta$ be a random vector called the *reference point*. Let $\bar{u}$, i.e. the inversion of u, be called the *antipodal point*. Then,

$$\mathsf{Quantize}(s;u) = \begin{cases} 0 & \text{if } d_\delta(s,u) < d_\delta(s,\bar{u}) \\ 1 & \text{if } d_\delta(s,u) > d_\delta(s,\bar{u}) \\ \mathsf{q} \sim \mathbf{Bern}(\frac{1}{2}) & \text{if } d_\delta(s,u) = d_\delta(s,\bar{u}) \end{cases}$$

defines a 1-bit stochastic quantizer (we can also use a deterministic tiebreak for the third case). Since $d_\delta(s,u) + d_\delta(s,\bar{u}) = \delta$, the quantization function can be computed trivially by

$$\mathsf{Quantize}(s;u) = \mathbb{1}\{d_\delta(s,u) \gtrless \frac{\delta}{2}\} \tag{9.4}$$

9.3.2.2 Encoding algorithm

To compress s^n, setting l to target the total number of quantization bits — this sets the target distortion — choose a random bipartite graph $\mathcal{L} = (\mathcal{S}, \mathcal{Y}, \cdot)$, where $\mathcal{Y} \triangleq \{g_1, ..., g_l\}$ denotes the l $\boxed{\mathsf{Q}}$ factor functions, and the degree on each g_b is δ_b. Write $A_b = \mathcal{N}_b^{\mathcal{L}}$ for the neighborhood of g_b on the $\mathcal{S}$ side. Choose also a collection of random reference points $\mathscr{U} = \{u_1, u_2, ..., u_l\}$, $u_b \in \mathbf{S}^{\delta_b}$. Apply

$$\begin{aligned} \mathrm{q}^l &= \mathsf{Quantize}(\mathrm{s}^n; \mathscr{U}, \mathcal{L}) \\ &\triangleq \begin{bmatrix} \mathsf{Quantize}(\mathrm{s}_{A_1}; u_1) \\ \mathsf{Quantize}(\mathrm{s}_{A_2}; u_2) \\ \vdots \\ \mathsf{Quantize}(\mathrm{s}_{A_l}; u_l) \end{bmatrix} \end{aligned} \tag{9.5}$$

This is followed by entropy coding. Setting k to target the final compression rate $r_{\text{code}} = k/n$, choose a random $H \in \mathscr{H}(l,k)$. Apply

$$\mathrm{x}^k = H\mathrm{q}^l \tag{9.6}$$

to obtain the compressed result.[4]

9.3.2.3 Decoding algorithm

Since the source and code components are independent and need not be modified in any way (cf. Section 7.4), we only describe the quantizer component. This component is much like a combination of the alphabet translator component (Fig. 7.2) and the code component (Fig. 3.2) in operation.

In similar fashion to Eq. 3.4, we can define the *quantizer constraint function*,

$$\begin{aligned} g(s^n, q^l) &\triangleq \mathbb{1}\left\{\mathrm{q}^l = \mathsf{Quantize}(s^n; \mathscr{U}, \mathcal{L})\right\}(s^n, q^l) \\ &= \prod_{b=1}^{l} g_b(s_{A_b}, q_b) \\ &= \prod_{b=1}^{l} \mathbb{1}\left\{\mathrm{q}_b = \mathsf{Quantize}(s_{A_b}; u_b)\right\}(s_{A_b}, q_b) \end{aligned} \tag{9.7}$$

[4] $\mathscr{U}$, $\mathcal{L}$, just as H (and doping, which we neglect here), are generated pseudo-randomly, and likewise need not be described explicitly to the decoder beyond synchronizing a seed.

Thus the entire decoder now attempts to marginalize

$$u(s^n, q^l) \triangleq c(q^l) g(s^n, q^l) p_{\mathsf{s}^n}(s^n) \tag{9.8}$$

for maximization over each s_i.

To obtain the messages, notice the quantizer component sends and receives external messages on the virtual ports of both sets of variable nodes ($\mathcal{S}$ and $\mathcal{Q}$). Let $[M^{i\leftarrow}]$ and $[M^{i\rightarrow}]$ denote respectively the input and output external messages on variables s_i. Let $\left[M^{b\leftarrow}\right]$ and $\left[M^{b\rightarrow}\right]$ denote respectively the input and output external messages on variables q_b. Denote by $\nu^{i\rightarrow b}$ a message passed from s_i to g_b, and by $\nu^{i\leftarrow b}$ one passed from g_b to s_i. Then, the updates are:

$\mathcal{S}$ node output:	$\nu^{i\rightarrow b} \Leftarrow \nu^{i\leftarrow\sim b}\left[M^{i\leftarrow}\right]_{\mathcal{L}}$
$\mathcal{S}$ external message output:	$\left[M^{i\rightarrow}\right]_{\mathcal{L}} \Leftarrow \nu^{i\leftarrow *}$
$\mathcal{Q}$ node ouptut:	$\nu^{i\leftarrow b} \Leftarrow g^{i}_{\sim i,b}\nu^{\sim i\rightarrow b}\left[M^{b\leftarrow}\right]_{\mathcal{L}}$
$\mathcal{Q}$ external message output:	$\left[M^{b\rightarrow}\right]_{\mathcal{L}} \Leftarrow g^{b}_{*}\nu^{*\rightarrow b}$

Table 9.1: Node computations for the LDHQ quantizer component. The wildcards are taken over neighbors in the graph $\mathcal{L}$.

If BP converges with $\hat{s}^n$ computed as usual, and the inferred values of $\hat{q}^l = \mathsf{Quantize}(\hat{s}^n; \mathscr{U}, \mathcal{L}) = \mathrm{q}^l$ and $\hat{x}^k = H\hat{q}^l = \mathrm{x}^k$ are correct, then decoding succeeds.

9.3.2.4 Exploratory results

We compress $\mathsf{s}^n \sim \mathbf{Bern}(p)$, $p = 1/2$, with Hamming distortion $d_n(s^n, \hat{s}^n) \triangleq (1/n)\sum_{i=1}^{n} \mathbb{1}\{s_i \neq \hat{s}_i\}$. The rate-distortion function for this source is, for $0 \leq \Delta \leq \min\{p, 1-p\}$,

$$\mathbb{R}(\Delta; \mathsf{s}) = h(p) - h(\Delta) \tag{9.9}$$

The linear processing (Ancheta) bound (Section 9.6) for $p = 1/2$ is, for $0 \leq \Delta \leq p$,

$$R^{\text{Ancheta}}(\Delta) = 1 - 2\Delta \tag{9.10}$$

This is indeed the time-sharing bound achieved by the type of geometric hashing methods described in Section 9.3.1.2.

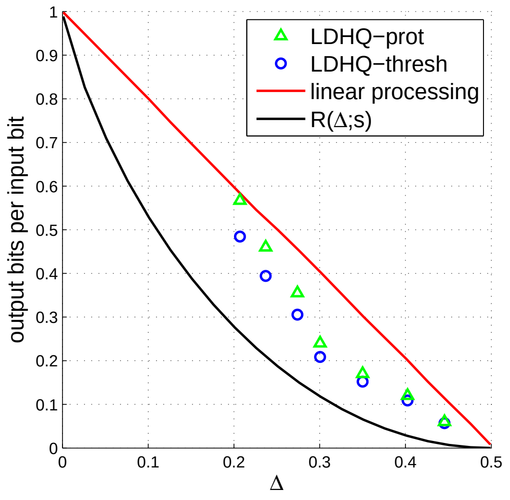


Figure 9.4: Compressing $\mathbf{Bern}(\frac{1}{2})$ under Hamming distortion with LDHQ. $n = 1000$, and $r_{\text{dope}} = 0.04$.

Our goal with LDHQ then is to demonstrate that the linear processing bound can be breached at low complexity, even within the context of model-quantizer separation. Fig. 9.4 shows a few trials where we set a constant degree on $\boxed{\mathsf{Q}}$ of only 3.[5] Note that $\lfloor nr_{\text{dope}} \rfloor$ bits of q^l (rather than s^n) are doped, and the reported rates include this, thus `LDHQ-prot` shows $r = r^*_{\text{code}} + r_{\text{dope}}$ and `LDHQ-thresh` shows $r^* = \epsilon^{\text{BP}} + r_{\text{dope}}$ (cf. Section 4.1). We see that, particularly at lower rates, decoding succeeds with demonstrably better performance than the linear processing bound.

The result suggests that forms of low-complexity, geometry-aware hashing may yet provide the desired performance in lossy compression problems for use in a separation architecture. A fuller exploration of this approach still awaits further research.

9.4 Quantization codebook mismatch

A joint model-quantizer system such as the random codebook scheme of Section 9.1.1 has the opportunity to use one codebook — perhaps the rate-distortion optimal one, $\mathscr{Q}^*_n$ — for both quantization and entropy removal. In the model-quantizer separation architecture, however, the encoder handles quantization of the space using a quantization codebook $\mathscr{Q}_n$ that does not know the distribution of the typical set within the space. Therefore, there is redundancy in the quantizer output sequences q^l, presenting as their non-uniform rates of occurrence. This section addresses the rate performance ramifications of *quantization codebook mismatch* between the optimal $\mathscr{Q}^*_n$ and the designed $\mathscr{Q}_n$, whether due to model-quantizer separation or other reasons.

[5]The quantization graph $\mathcal{L}$ is generated using the same library that generates the coding graph $\mathcal{C}$.

9.4.1 Entropy coding

Non-uniform quantizer output is removable by entropy coding. In the high-rate limit, *entropy coded quantization (EC*Q)* approaches optimal performance for space-filling model-free quantizers, because the quantization cells are small enough compared to the resolution of data distribution variation. But at lower rates, there are few quantization cells that must cover larger parts of the typical set, and optimal compression requires a specific arrangement of cells, so there is a performance loss even with entropy coding, very much analogous to the Wyner-Ziv loss seen in Section 9.1.1.

9.4.2 Main results

We want to know how much loss there is to using an arbitrary system codebook $\mathscr{Q}_n$, and how much of it is regained by entropy coding on its output $q^l \Leftrightarrow \hat{s}^n$, under the specific condition that we know the original source distribution at the decoder — not a condition usually considered.

To do this, we explicitly view the quantization codebook as a stochastic collection $\mathsf{Q}_n \triangleq \{(\hat{\mathsf{s}}^n)_1, (\hat{\mathsf{s}}^n)_2, ..., (\hat{\mathsf{s}}^n)_{|\mathscr{Q}_n|}\}$ of reproduction words drawn from a *design distribution* on the words of $\mathbf{S}^n$.

The performance of lossy compression under codebook mismatch is extensively surveyed in [121] by large deviation methods. In it, we find the following results:

Definition 9.2. Let P be the source law, i.e. p_{s^n}. Let W be an arbitrary stochastic encoding law, i.e. $p_{\hat{\mathsf{s}}^n|\mathsf{s}^n}$. Let PW be the joint distribution of the source-reproduction pair under source law P and encoding law W, i.e. $p_{\mathsf{s}^n\hat{\mathsf{s}}^n}$. Let Y be the output distribution under source law P and encoding law W, which is to say, $p_{\hat{\mathsf{s}}^n} = \int_{s^n} p_{\mathsf{s}^n}(s^n) p_{\hat{\mathsf{s}}^n|\mathsf{s}^n}(\hat{s}^n|s^n)$ or $Y = \int_P PW$ for short. (We also call Y the *effective codebook distribution*; it can differ from the design distribution.) Let Q be an arbitrary codebook design distribution. Let Q^* be the optimal codebook distribution according to rate-distortion theory, i.e. $p^*_{\hat{\mathsf{s}}^n}$.

Then, with all optimizations subject to $\mathbb{E}d_n(\mathsf{s}^n, \hat{\mathsf{s}}^n) \leq \Delta$,

- When the encoder has P, it can compute for itself the optimal codebook distribution by setting the design distribution Q to be the effective distribution Y, giving Q^*. The optimal rate of compressing a P-source with this codebook is, per rate-distortion theory,

$$\begin{aligned}\mathbb{R}(\Delta; \mathsf{s}^n) &= \inf_{W: Q=\int_P PW} D(PW||P \times Q) \\ &= \inf_W D(PW||P \times Y) \\ &= \inf_W \mathbb{I}(\mathsf{s}^n; \hat{\mathsf{s}}^n)\end{aligned}$$

- When the encoder and the decoder both do not have P, nor know the optimal codebook distribution Q^* through other means, then they must quantize using some arbitrary codebook of design distribution Q. The optimal rate of compressing a P-source now is

$$\begin{aligned} R^Q(\Delta) &= \inf_W D(PW||P \times Q) \\ & \inf_W [D(PW||P \times Y) + D(Y||Q)] \\ &= \inf_W [\mathbb{I}(\mathsf{s}^n; \hat{\mathsf{s}}^n) + D(Y||Q)] \\ &\geq \mathbb{R}(\Delta; \mathsf{s}^n) \end{aligned}$$

The difference $R^Q(\Delta) - \mathbb{R}(\Delta; \mathsf{s}^n)$ represents the loss for using the suboptimal codebook Q for quantization. Theoretically, no entropy coding is possible, because we do not "know" the distribution of the quantization output, either (without learning).

In our architecture, however, it is only the model-free encoder that does not know/use the source distribution P and therefore cannot assume the optimal codebook distribution Q^*. Instead, it quantizes using an arbitrary codebook of design distribution Q, as above. However, the decoder does have the source distribution P, as well as the codebook distribution Q that the encoder chooses via the quantizer. Thus we need a new result.

Definition 9.3. Let $W_Q \triangleq \arg\inf_W D(PW||P \times Q)$ denote the optimal encoding law when coding a P-source with an arbitrary codebook of design distribution Q. Let $Y_Q = \int_P PW_Q$ denote the associated effective distribution at the output.

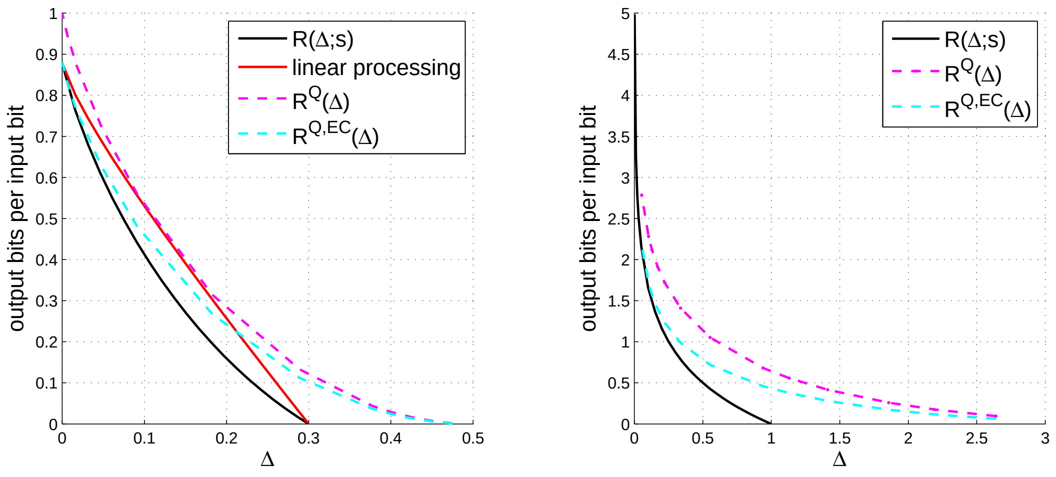
Remark 9.4. Y_Q is the output distribution of the codebook under usage, and it is as if the codebook is actually distributed as Y_Q from the perspective of the system. Indeed, it is indistinguishable from the case where the codebook is designed with distribution Y_Q instead of Q, because the encoding law Y_Q must be optimal in either case.

A concrete example is if a quantizer uses a (deterministic) codebook $\mathscr{Q}_n$ (thus Q is uniform over the entries of $\mathscr{Q}_n$), but in usage some entries are used more than others according to Y_Q, then, the opportunity to reduce the coding rate from $\log |\mathscr{Q}_n|$ to $\mathbb{h}(Y_Q)$ is exactly the role of lossless entropy coding, provided Y_Q is known.

Theorem 9.5. *The optimal rate for rate-distortion coding a P-source, using a quantizer with codebook design distribution Q at the encoder, with knowledge of P and Q at the decoder, is*

$$\begin{aligned} R^{Q,\mathrm{EC}}(\Delta) &= \inf_W D(PW||P \times Y_Q) \\ &= \inf_W [D(PW||P \times Q)] - D(Y_Q||Q) \\ &= R^Q(\Delta) - D(Y_Q||Q) \\ &\leq R^Q(\Delta) \end{aligned}$$

The term $D(Y_Q||Q)$ is the rate recovered by entropy coding the quantizer output. Furthermore, the encoder can apply model-free coding (lossless compression) at rate $R^{Q,\mathrm{EC}}(\Delta)$, knowing the decoder, with P and Q at hand, will "compute" and apply the model for Y_Q, the utilized, effective distribution of the codebook Q_n, rather than Q itself. Note that this



(a) **Bern**(0.3), Hamming distortion (b) $\mathbf{N}(0,1)$, MSE distortion

Figure 9.5: Performance comparison of compressing i.i.d. sources with mismatched codebooks: (a) $Q = \mathbf{Bern}(0.5)$ when $Q^* = \mathbf{Bern}(p \neq 0.5)$; (b) $Q = \mathbf{Unif}(-3, 3)$ when $Q^* = \mathbf{N}(0, \Sigma < 1)$.

"computation" need not be an explicit one, since the distribution emerges on the graphical nodes $\mathsf{q}^l \Leftrightarrow \mathsf{s}^n$ from the inferential decoder applying the quantizer design $\mathcal{L}$ (codebook and encoding law) and the source model $\mathcal{G}$.

9.4.3 Examples

Fig. 9.5 shows the various rate functions computed for two i.i.d. sources (codebooks are also i.i.d., thus all quantities refer now to single-letter marginals). For the **Bern**(p) Hamming case, the optimal reproduction distribution is

$$Q^* = \mathbf{Bern}\left(\frac{p - \Delta}{1 - 2\Delta}\right) \tag{9.11}$$

while for the $\mathbf{N}(0, \Sigma)$ MSE case, the optimal reproduction distribution is

$$Q^* = \mathbf{N}(0, \Sigma - \Delta) \tag{9.12}$$

We are particularly interested in the uniform (or other entropy maximizing) distribution for the quantizer, because a model-free quantizer almost certainly has to use it. In both cases, we see $R^{\mathrm{Q,EC}}(\Delta)$ converging to $\mathbb{R}(\Delta; \mathsf{s})$ at high rate and also perform well at intermediate rates. At lower rates, the performance is less satisfying, but note that time-sharing in an architecturally compatible way with the $\mathbb{R}(\Delta; \mathsf{s})$ zero-rate extremum (i.e. not coding some symbols) gives performance much closer to $\mathbb{R}(\Delta; \mathsf{s})$.

9.5 Summary

By drawing on Wyner-Ziv coding from rate-distortion theory and practical lossy compression systems that feature partial model-code separation, then linking them with entropy-coded quantizer designs, we obtain a separation architecture that keeps geometry information in the encoder while moving typicality information into the decoder. A new performance bound shows such separation architecture incurs no loss in the high-rate limit.

Practically, we show how the architecture can reuse components from lossless compression by adding a model-free quantizer. Existing quantizers, including traditional, coded, and geometric hashing quantizers, as well as the low-density hashing quantizer (LDHQ) that we construct, can be used. The LDHQ in particular gives encouraging performance.

In general, lossy compression systems are much more complex than lossless systems, but separation also lends us one more tool to manage their complexity, while giving us the systemic flexibility we expect.

9.6 Appendix: Ancheta bound*

The linear processing lower bound for lossy compression of **Bern**(p) i.i.d. sources with Hamming distortion due to Ancheta [23], quoted below, gives the most optimistic rate-distortion tradeoff for compression systems using linear processing in the encoder. This bound puts an important limitation on system performance because many lossy compression algorithms for this source turn out to be linear processing. Better performance than the Ancheta bound, especially at low complexity, is difficult.

Note that it does not apply to using linear codes, e.g. coded quantization using linear subspaces as reproduction, but solely to linear processing.

Fact 9.6. *The Ancheta bound $R^{\text{Ancheta}}(\Delta)$ is the solution for r to*

$$\Delta = (1-r)h^{-1}\left(\frac{h(p)-r}{1-r}\right) \tag{9.13}$$

This is strictly bounded away from the rate-distortion function for $p \neq 1/2$, and it is conjectured that the even more pessimistic, time-sharing bound (achieved by losslessly coding a portion of the bits, discarding the rest) is the tight lower bound for all p.

10 Toward Realistic Applications

In this chapter, we consider two compression tasks dealing with real image data to give a flavor of the type of issues one may encounter when putting the architecture into implementation. The first is a bi-level image compression task where we use the binary Ising model as the image model. The second is compressing a grayscale image in the encrypted domain. These examples are not meant to be the definitive word on these tasks, but are useful to show the beginning of any development process toward realistic applications.

10.1 Bi-level image compression

Real images have strong characteristics known as image models, but unlike the case of Section 4.4, we do not have a definitive way to create a graphical model for them. Nevertheless, we can assume the $\mathbf{Ising}(p, q)$ model family, apply them to real bi-level image sets, accept the mismatch, and see how an extremely simple image model fares. It can be considered the beginning of a development process for more complex image models. We also do not optimize the code, thus the results represent a realistic look at performance with the material already in this thesis without additional development.

10.1.1 Experimental setup

Image sets consisting of a large number of similar images are collected, scaled to size, and thresholded to bi-level images. The rate performance (output bits per input bit) for each image compressed is reported for `SEP-prot`, `JBIG2`, and `GZIP` (Section 4.1).

10.1.2 Palmprints under the Ising model

We test compression on the CASIA biometric palmprints database (Fig. 10.1) [122].[1] We assume the images are drawn from the $\mathbf{Ising}(\frac{1}{2}, q)$ model and estimate the parameter q according to Section 6.4.1. In this case, the images have large solid patches, and are thus a fairly good match to the Ising model at certain parameters. The compression performance is demonstrative of this (Fig. 10.2), where `SEP-prot` performance is comparable with `JBIG2` performance (the latter a little better), and exceeds `GZIP` performance by a larger margin.

[1] We converted the 8-bit grayscale images to 1-bit ones by thresholding at mid-level.

10.1.3 Handwritten digits under the Ising model

We test compression on the MNIST handwritten digits database (Figure 10.1) [123].[2] We again assume the images are drawn from the $\mathbf{Ising}(\frac{1}{2}, q)$ model of Section 4.4 and estimate the parameter q. This is a greater mismatch in image models than with palmprints. However, the performance is still reasonably good (Fig. 10.3) compared to `JBIG2` and `GZIP`.

10.1.4 Discussion

The results in the experiments, while encouraging, are based on a rudimentary image model that is unlikely to perform equally well under all circumstances. For more complex images, we would prefer a less mismatched image model. A possibility is a context-based model much like what is used in `JBIG2` [124]. We can easily replace the source model in the `SEP` architecture, by implementing a contextual probability table, in which each entry $p(s_i|\mathcal{N}_i^{\mathcal{G}} = c)$ is the empirical probability of the center pixel value s_i conditioned on the context state c of its neighbor pixels $\mathcal{N}_i^{\mathcal{G}}$. A flexibility not afforded to sequential coders like `JBIG2` is that we are not restricted to causal contexts, which should improve performance.

Alternatively, we can retain the Ising model, but augment it with non-homogeneous parameters across the image, then apply the learning methods of Section 6.4.4.

10.2 Encrypted compression

We now turn to an application that absolutely requires model-code separation: compression in the encrypted domain. Sometimes the owner of data wishes to retain total secrecy, and so presents to the encoder a version of the data that has already been encrypted. Refer to Fig. 10.4 for the system diagram. Upstream from the encoder, the source data s^n has been serialized to a bit-stream, then encrypted by a one-time pad system using an additive binary key k^{nB} to give the encrypted bits z^{nB}, which the encoder takes as input. The task is to compress without the encryption key at the encoder, but at the decoder.

[2]We converted the 8-bit grayscale images to 1-bit ones by thresholding at mid-level, and combined 16 digits into a single image to produce a more complex test image.

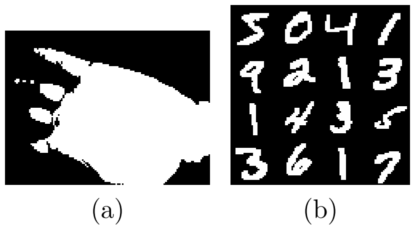

(a) (b)

Figure 10.1: Example of (a) palmprint and (b) handwritten digits images used in Section 10.1.

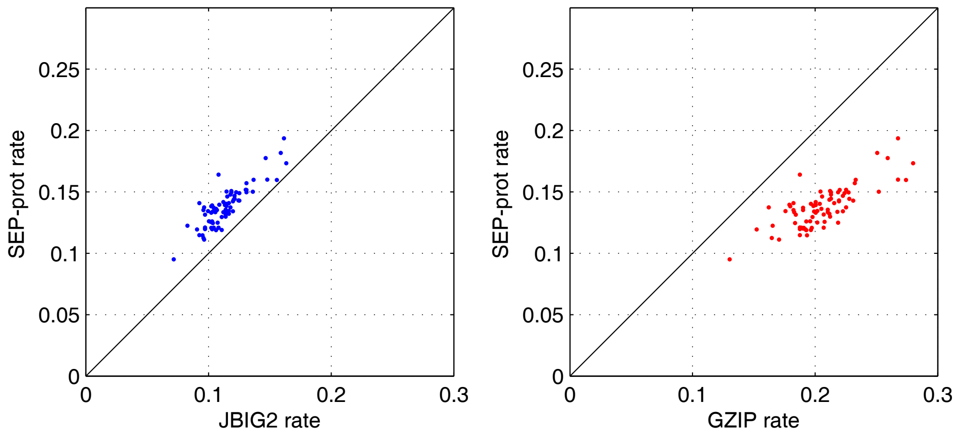


Figure 10.2: Under the **Ising**(p, q) model, performance of `SEP-prot` is comparable to `JBIG2` and better than `GZIP` on 80 images (96×128) derived from the CASIA biometric palmprints database.

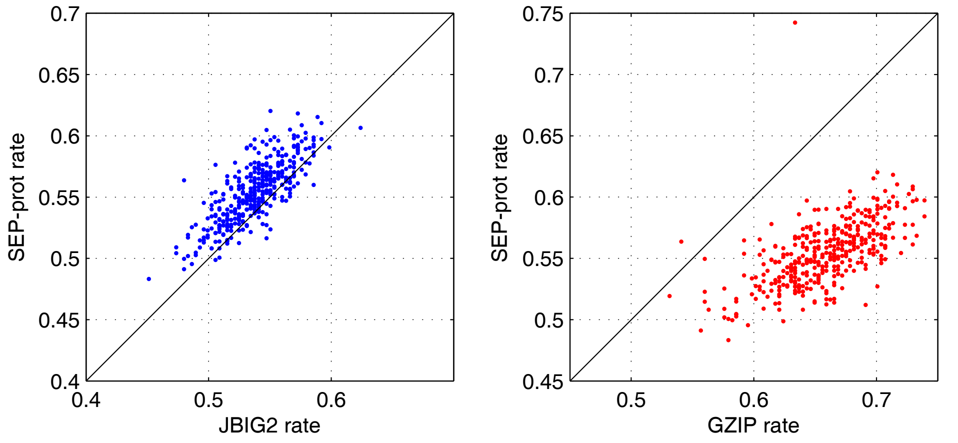


Figure 10.3: Under the **Ising**(p, q) model, performance of `SEP-prot` is comparable to `JBIG2` and better than `GZIP` on 3750 images (50×50) derived from the MNIST handwritten digits database.

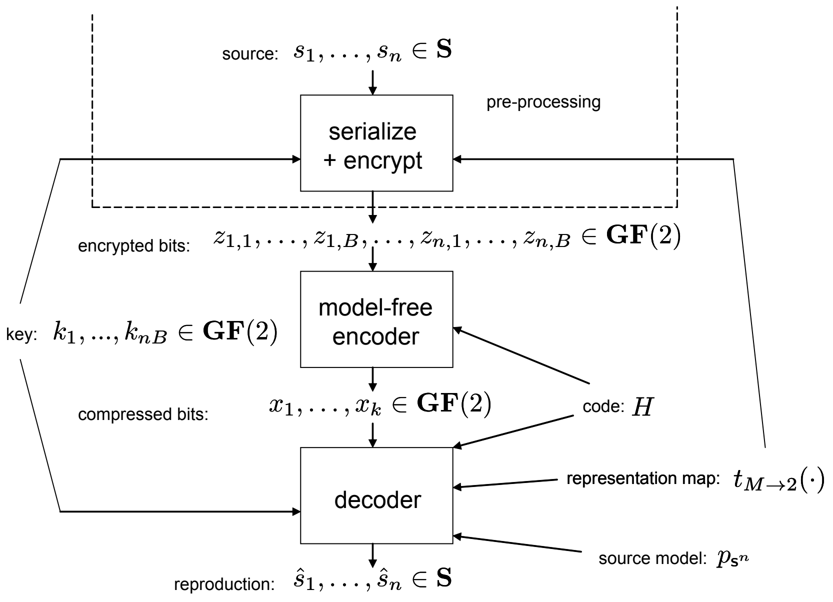

source: $s_1, \ldots, s_n \in \mathbf{S}$
pre-processing
serialize + encrypt
encrypted bits: $z_{1,1}, \ldots, z_{1,B}, \ldots, z_{n,1}, \ldots, z_{n,B} \in \mathbf{GF}(2)$
key: $k_1, ..., k_{nB} \in \mathbf{GF}(2)$
model-free encoder
code: H
compressed bits: $x_1, \ldots, x_k \in \mathbf{GF}(2)$
decoder
representation map: $t_{M\to 2}(\cdot)$
source model: p_{s^n}
reproduction: $\hat{s}_1, \ldots, \hat{s}_n \in \mathbf{S}$

Figure 10.4: Compressing an encrypted source with a model-free encoder. The already encrypted bits are presented to the encoder to compress, which without the secret key, is impossible to do traditionally.

10.2.1 Encrypted encoding

In the language of Section 7, the encoder receives

$$\mathrm{z}^{nB} = t_{M\to 2}(\mathrm{s}^n) \oplus \mathrm{k}^{nB} \tag{10.1}$$

Normally, this stream is totally uncompressible without the encoder also having the key. However, we can simply view $\oplus \mathrm{k}^{nB}$ as part of the representational map and apply coding to $t_{M\to 2}(\mathrm{s}^n) \oplus \mathrm{k}^{nB} \equiv t'_{M\to 2}(\mathrm{s}^n)$ in the same fashion as Section 7.3.2:

$$\mathrm{x}^k = H\mathrm{z}^{nB} = Ht'_{M\to 2}(\mathrm{s}^n) \tag{10.2}$$

10.2.2 Encrypted decoding

Note that in the joint graph $\mathcal{U}$ for this system (Fig. 10.5), the code subgraph is entirely in the *encrypted domain* since z^{nB} and $\mathrm{x}^k = H\mathrm{z}^{nB}$ are encrypted. Note further that the source subgraph is entirely unencrypted since s^n, the object of decoding, is unencrypted. Therefore the encryption boundary coincides with the representation boundary. Then apply the decoding of Section 7.3.3, using $t'_{M\to 2}$ and the corresponding $T'_{M\to 2}$ and $T'_{2\to M}$ pair.

In practice this means during each iteration of computations (Algorithm 7.1), $m^{(M)}_{\mathcal{C}\to i}(s_i)$ is obtained by message decryption followed by conventional translation $T_{2\to M}$, and $m^{(q)}_{\mathcal{G}\to i,\omega}(z_{i,\omega})$ is obtained by conventional translation $T_{M\to 2}$ followed by message encryption, with the rest of the decoding algorithm unchanged. With the key, message encryption and decryption

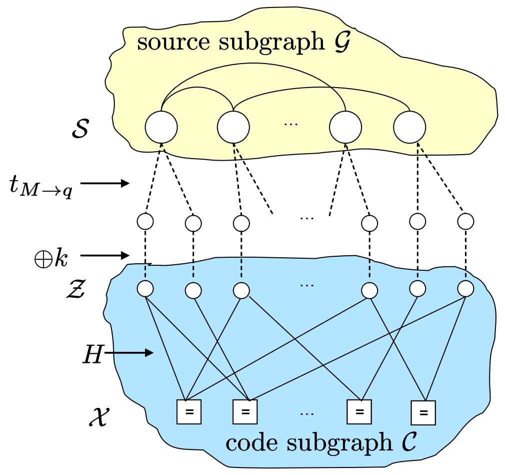


Figure 10.5: The structure of the decoder inference graph for the system in Section 10.2.

simply amount to permuting the labels of bits in binary messages.

10.2.3 Image data example

In this section, elements in the encrypted domain are underlined for clarity.

In Fig. 10.6, we show an example of compressing an encrypted 8-bit grayscale image $\mathrm{s}^{h\times w}$ (Fig. 10.6a). We apply the **PottsZ**[256](h) model of Section 8.2 as image model in the decoder. A sample drawn from this model $\mathsf{s}^{h\times w} \sim$ **PottsZ**[256](h) is displayed for comparison (Fig. 10.6b). As one can see, this is a rather crude model for the natural image.

To encrypt the image, a random encryption key k^{8hw} is applied to the internal bits of the image, i.e.,

$$\underline{\mathrm{z}}^{8hw} = t_{256\to 2}(\mathrm{s}^{h\times w}) \oplus \mathrm{k}^{8hw} \tag{10.3}$$

The encrypted "image" can be viewed by treating $\underline{\mathrm{z}}^{8hw}$ as unencrypted bits. As expected,

$$\underline{\mathrm{s}}^{h\times w} \triangleq t_{2\to 256}(\underline{\mathrm{z}}^{8hw}) \tag{10.4}$$

becomes snowflake noise (Fig. 10.6c) without the key, and traditional compression is not possible.

The data is compressed by encrypted encoding,

$$\underline{\mathrm{x}}^{k} = H\underline{\mathrm{z}}^{8hw} \tag{10.5}$$

Doping is then applied to the encrypted bits corresponding to a selected subset $s_{\mathcal{D}}$ of $s^{h\times w}$, resulting in the initialization in the encrypted domain with node potentials on $\mathcal{Z}$ of

$$\underline{d}^{(2)}(z_{\mathcal{D}'}) = \mathbb{1}\{z_{\mathcal{D}'} = \underline{\mathrm{z}}_{\mathcal{D}'}\}(z_{\mathcal{D}'}) \tag{10.6}$$

where $\mathcal{D}' \subseteq \mathcal{Z}$ are the bits corresponding to the symbols of $\mathcal{D} \subseteq \mathcal{S}$ under the representational map $t'_{256\to 2}$. This encrypted-domain initialization is displayed in image form (Fig. 10.6d) by setting the initial encrypted beliefs to

$$\underline{b}^{(256)}(s_{\mathcal{D}}) = T_{2\to 256}(\underline{d}^{(2)}(z_{\mathcal{D}'} = 1)) \tag{10.7}$$

$$\underline{\hat{s}}_{(i,j)} = \arg\max_{s_{(i,j)}} \underline{b}^{(256)}_{(i,j)}(s_{(i,j)}) \tag{10.8}$$

In this example, $r_{\text{code}} = 0.7$, and $r_{\text{dope}} = 0.04$, at which settings decoding without encryption is possible, thus it proceeds exactly the same with encryption. The only difference is the decoder graph $\mathcal{U}$ has a layer modeling encryption/decryption.

Finally, we show two diagnostic examples of missing components in the decoder. Note that at any time during decoding, we can obtain a decrypted estimate of $s^{h\times w}$ from the marginal beliefs $b^{(256)}_{(i,j)}(s_{(i,j)})$ on $\mathcal{S}$,

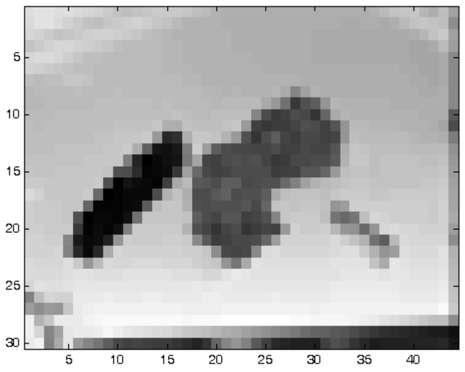
$$\hat{s}_{(i,j)} = \arg\max_{s_{(i,j)}} b^{(256)}_{(i,j)}(s_{(i,j)}) \tag{10.9}$$

If the decoder has the encryption key but does not have the code graph, it decodes relying only on the image model and doping; then we obtain one type of unstable "solution" (Fig. 10.6e) showing an image that has somewhat the layout of the original image while following strongly the generic image model applied.

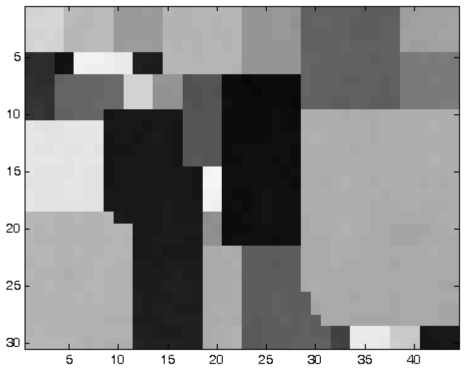
On the other hand, if the decoder has the complete coding components but not the encryption key — supplying a random key or pretending $\underline{z}^{8hw}$ is in-the-clear, we obtain another type of unstable "solution" (Fig. 10.6f) showing a nearly random image having the characteristics of the image model. Since the values supplied to the decoding, except for the locations of the doped symbols, are entirely random, decoding not only does not succeed but provides no information except what would be obtained by beginning with a set of random pixel values.

10.2.4 Discussion

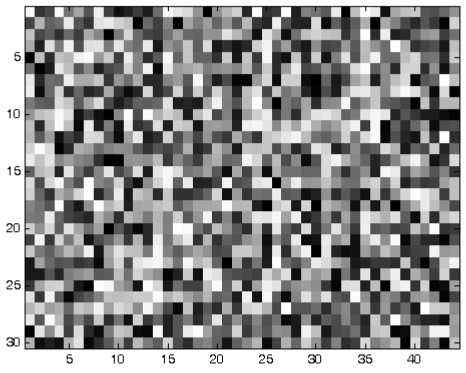
In this application, both the image model and the cryptographic scheme are basic designs. For grayscale (and color) images, though the Potts model has been used in image processing and vision research, very different models based on transform-domain priors [125] and occlusive patches [126] are also popular and require additional technology to implement them. Similarly, many existing cryptosystems [127, 128] have been deployed and applied to data, so we must be prepared to design message-passing decoder components for those systems, or develop new methods of low-complexity cryptographical inference to serve the important application of encrypted compression.



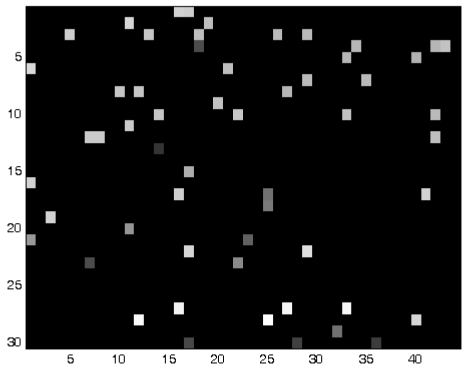
(a) ground truth $\mathrm{s}^{h \times w}$



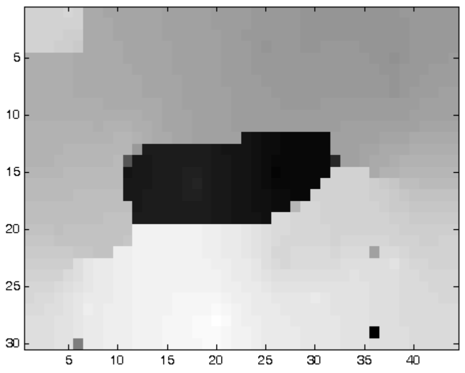
(b) random $\mathsf{s}^{h \times w}$ sample



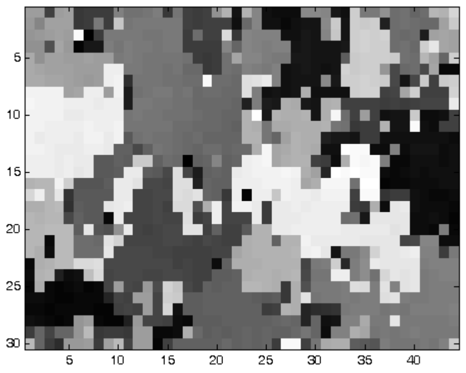
(c) encrypted image $\underline{\mathrm{s}}^{h \times w}$



(d) doping initialization $\underline{\hat{s}}^{h \times w}$



(e) no code component



(f) with wrong key supplied

Figure 10.6: Compressing an encrypted grayscale image ($\mathbf{S} = \mathbf{Z}_{256}$) of size $(h, w) = (30, 44)$, using the image model $\mathbf{PottsZ}[256](0.4)$. $r_{\text{code}} = 0.7$, and $r_{\text{dope}} = 0.04$.

11 Conclusion

Let us review the contributions of this work and look forward to what is to come.

11.1 Review

In this thesis, we proposed a model-code separation architecture for data compression that gives many advantages of systemic flexibility over current system design. We are able to incorporate prior knowledge about data and its processing into systems very easily, without affecting the other parts of the system. In turn, we obtain features in compression that we are unable to have with current systems.

11.1.1 Requirements

Of the requirements for contemporary and future compression listed in Section 1.1.2, we have addressed the first two, *design flexibility* and *upgrade flexibility*, throughout the thesis, but in particular in Sections 3.3.1 and 7.4. The utility of model-code separation for these requirements is clear. It allows us the flexibility to alter the data model and processing assumptions at any time. Systems for new data types can be designed incrementally with low barrier to entry. Agents have the means and incentives to compete to improve on components.

We addressed *security* in Section 10.2. The remaining requirements are equally evident. We have created an architecture in which the encoder is lightweight, and which for lossless compression and high-rate lossy compression, does not sacrifice performance over the joint architecture — ideal for mobile acquisition and encoding (*mobility*). Our compressed data on which systems are standardized is also a true information stream that can be re-ordered, partially lost, accumulated from multiple origins, etc., affording us many modalities of pipeline design (Section 3.2.5) and the kind of *distributivity* and *robustness* that does not require joint source-channel coding.

11.1.2 Design proposals

Throughout the thesis, we identified the joint model-code architectural style not only with common practical systems but also with the Shannon random codebook scheme from the earliest days of compression research. But we also saw the Slepian-Wolf and Wyner-Ziv random binning schemes as prototypes (Sections 2.1.2, 9.1.2) on which to build model-code separation systems like `SEP`.

Based on these prototypes, we constructed several systems and architectural generalizations to compress both losslessly and lossily (Section 9.3.2), with models both certain and uncertain (Section 6.4), and for data that can be over alphabets entirely different from that of the coding system (Section 7.3). The design choices were not only easily enabled by the architecture itself but sometimes became self-evident. The same approach can be taken for numerous other scenarios which have not been explicitly considered in this thesis.

11.2 Future work

It is by purposeful choice that this thesis focuses on architectural issues primarily. There is much more remaining to be done, and it would be unrealistic to answer every question regarding e.g. modeling, coding, graphical algorithms, which are important research fields in their own right. What we have presented is a system architecture that, by its construction, links the theory of source coding more closely to the practice of compression system design. Therefore, perching at the threshold of theory and practice, future work is easily accessible in each direction.

11.2.1 Modeling

As this work has shown, this aspect of compression is perhaps the most important of all. We seek models for different types of data that now exist or are being generated in large quantities, e.g. financial and biological data. Existing systems have had decades to accumulate expertise and trials in this area, and a somewhat more robust amount of development than what can be mustered in this work will be a boon for performance.

We also need to develop representations for complex models and data structures to work with them. We are intrigued by the possibility of models on graphs that may go beyond PGM's, including various computational, sampling, or constraint models that may prove to better describe data types.

11.2.2 Coding

We seek better code design, not just for better optimized LDPC codes or other graphical codes, but also for rate-adaptive, streaming, or sequential versions of them.

Although coding has been relegated to somewhat the role of a workhorse component in the present architecture, it too is very important, inasmuch as we require it to have robust behavior and good performance across a range of scenarios. We have shown this is the case for a number of cases, but beyond empirical evidence, a more general analysis to provide guarantees is also desired.

11.2.3 Quantizer design

Having also conceptually and architecturally separated the quantizer from code and model in separation architecture lossy compression, we are left with the problem of finding suitable geometry-aware hashes for lossy compression. The traditional conception that conflates geometric requirements with statistical requirements in designing a "code" that works for both is not without its contradictions, but the separated design of quantizer also has its own set of challenges.

The low-density hashing quantizer (LDHQ) design as outlined in this work requires further refinement. For example, we need to consider how to generate a finite number of quantization regions for spaces that have infinite extent. Though this is one design, there are other, more exotic direct mapping methods such as space-filling curves [97] that can now be put into experimentation in this modular architecture.

Lossy compression in general is a highly complex problem, of which the distortion measure is still retained in its most rudimentary form in our presentation. How to work with non-idealized distortion is a problem of equal magnitude to how to work with non-idealized data models, and it is worthwhile to further consider whether this architecture permits the use of more implicit forms of distortion as in [129].

11.2.4 Communication system design

Since all compressed bits are equally informative, this architecture allows seamless transmission over a symbol erasure channel by adjusting the compression rate. In more general transmission scenarios, we can consider incorporating channel coding — especially linear channel coding — within the same architecture, perhaps similar in spirit to e.g. [130, 131]. The inference can be taken over both the source and channel parts in the decoder, yet each part can be substituted.

We should also note that the results and designs in this work directly carry over to the important dual problem of channel coding for additive-noise channels with memory, thus opens another avenue of inquiry.

11.2.5 Algorithms and implementation

The modular design principles and interfaces uncovered in this work can directly map to functional factorization in code (indeed, we have done so to an extent). There are many other ways to improve an implementation, though, that may not be readily apparent from this treatment. For example, better numerical stability and complexity reduction are achievable through an extensive use of computations in log-likelihood and other distribution spaces.

We also seek complexity reduction in other ways, especially with regard to messages over larger alphabets, where perhaps particle or sampling methods may be required as model size grows larger. Too, the judicious removal of unused graph edges and the preservation of messages during decoding should grant BP a degree of restartability, and a faster determination of convergence.

The above are problems where the presented system design can receive and benefit from a great deal of research already taking place in each area. There are a few more expansive, higher-level ideas that are relevant.

11.2.6 Hardware acceleration of BP

BP is currently implemented on CPU's, GPU's, and some ASIC's. Since our architecture relies almost exclusively on large-scale Bayesian inference, and algorithms such as BP repeatedly apply accumulation and filtration over messages as atomic operations, it begins to seem as if the CPU or even GPU instruction sets are not the most suitable. The complexity of inference is to a large degree an artifact of counting operations over integer or floating point numbers, rather than over the atomic operations of a Bayesian calculus.

One part of a practical future for compression with the architecture here may well lie in incorporating into hardware processors computational units that store messages and apply operations over messages, much as the floating point has become embedded as a basic hardware type. All inferential algorithms will gain in a fundamental way.

11.2.7 Ecosystem

A greater goal of this work is to open compression systems to improvement in all its sundry parts. For this to occur, not only need the architecture design be modular and flexible, but the "architecture" for the sharing of expertise, design, prototype implementations, and performance evaluations also needs to be well designed.

To be concrete, what are considered as data types need to be well defined, perhaps through representative training and testing data sets as is sometimes done, so that two competing parties can agree that they are designing data models for the same type of data. As new data types become available for which no compression systems exist, such an ecosystem for introducing and modifying designs quickly in a backward compatible way begins to show its value.

11.3 Epilogue

Our data demand continues to grow every year. The International Data Corporation (IDC) estimates that by the year 2020, 40 zettabytes of data will be created, replicated, and consumed each year, an order of magnitude more than today [132]. Nearly all of the data then existing will be new data, some of it will be specialized data in enterprise, scientific, and technology domains, some of it will be user generated media content, some of it will be measurements and samples from distributed sensors and devices. These are complex data for complex applications, especially at scale, so the opportunities to create high-performance data compression systems are truly great, but the challenges are equally daunting.

Whereas current systems are constrained by their joint model-code architecture to monolithic, immutable, and non-reusable designs, a flexible, modular, and extensible model-code separation architecture for compression frees us to meet the requirements of modern design we set out with.

This thesis provides the basic tools and insights to work with this new architecture, but it is only a modest beginning. Data compression lies at the intersection of the theories of information, learning, coding, complexity, and algorithms. As the various fields from which this work draws on advance, the compression systems we can design advance with them. Future research will undoubtedly enrich our understanding of these areas at the same time that it helps us to build better systems.

Index

Bibliography

[1] K. Sayood, *Introduction to Data Compression, Fourth Edition.* Morgan Kaufmann, Oct. 2012.

[2] D. Salomon, *Data Compression: The Complete Reference.* Springer, Feb. 2004.

[3] A. Said, "Introduction to arithmetic coding-theory and practice," *Hewlett Packard Laboratories Report*, 2004.

[4] T. Bell, I. H. Witten, and J. G. Cleary, "Modeling for text compression," *ACM Computing Surveys (CSUR)*, vol. 21, no. 4, pp. 557–591, 1989.

[5] J. Rissanen and J. Langdon, G.G., "Universal modeling and coding," *IEEE Transactions on Information Theory*, vol. 27, no. 1, pp. 12–23, Jan. 1981.

[6] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, Oct. 1948.

[7] R. Gray and L. Davisson, "Source coding theorems without the ergodic assumption," *IEEE Transactions on Information Theory*, vol. 20, no. 4, pp. 502–516, Jul. 1974.

[8] D. Slepian and J. Wolf, "Noiseless coding of correlated information sources," *IEEE Transactions on Information Theory*, vol. 19, no. 4, pp. 471–480, Jul. 1973.

[9] A. Hitron and U. Erez, "Optimality of linear codes over PAM for the modulo-additive gaussian channel," in *2012 IEEE International Symposium on Information Theory Proceedings (ISIT)*, Jul. 2012, pp. 1742–1746.

[10] A. G. Sahebi and S. S. Pradhan, "On the capacity of abelian group codes over discrete memoryless channels," in *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on.* IEEE, 2011, pp. 1743–1747.

[11] R. Dobrushin, "Asymptotic optimality of group and systematic codes for some channels," *Theory Probab. Appl.*, vol. 8, no. 1, pp. 47–60, Jan. 1963.

[12] J.-P. Barthelmy, G. Cohen, and A. Lobstein, *Algorithmic Complexity and Telecommunication Problems.* CRC Press, Jan. 1997.

[13] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms.* Cambridge University Press, Oct. 2003.

[14] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[15] E. Hof, I. Sason, and S. Shamai, "Performance bounds for nonbinary linear block codes over memoryless symmetric channels," *IEEE Transactions on Information Theory*, vol. 55, no. 3, pp. 977–996, Mar. 2009.

[16] A. Kakhaki, H. Abadi, P. Pad, H. Saeedi, F. Marvasti, and K. Alishahi, "Capacity achieving linear codes with random binary sparse generating matrices over the binary symmetric channel," in *2012 IEEE International Symposium on Information Theory Proceedings (ISIT)*, Jul. 2012, pp. 621–625.

[17] S. Kumar, A. Young, N. Maoris, and H. Pfister, "A proof of threshold saturation for spatially-coupled LDPC codes on BMS channels," in *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Oct. 2012, pp. 176–184.

[18] S. Kudekar, C. Measson, T. Richardson, and R. Urbanke, "Threshold saturation on BMS channels via spatial coupling," in *2010 6th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, Sep. 2010, pp. 309–313.

[19] I. Sason and B. Shuval, "On universal LDPC code ensembles over memoryless symmetric channels," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp. 5182–5202, Aug. 2011.

[20] J. Lu and J. Moura, "Linear time encoding of LDPC codes," *IEEE Transactions on Information Theory*, vol. 56, no. 1, pp. 233–249, Jan. 2010.

[21] M. H. Taghavi, "Decoding linear codes via optimization and graph-based techniques," Ph.D. dissertation, University of California, San Diego, 2008.

[22] I. Csiszár, "Linear codes for sources and source networks: Error exponents, universal coding," *IEEE Transactions on Information Theory*, vol. 28, no. 4, pp. 585–592, Jul. 1982.

[23] J. L. Massey, "Joint source and channel coding," 1977.

[24] G. Caire, S. Shamai, and S. Verdú, "Noiseless data compression with low-density parity-check codes," *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 66, pp. 263–284, 2004.

[25] J. H. Bae and A. Anastasopoulos, "Capacity-achieving codes for finite-state channels with maximum-likelihood decoding," *Selected Areas in Communications, IEEE Journal on*, vol. 27, no. 6, pp. 974–984, 2009.

[26] S. Kudekar and K. Kasai, "Threshold saturation on channels with memory via spatial coupling," in *2011 IEEE International Symposium on Information Theory Proceedings (ISIT)*, Jul. 2011, pp. 2562–2566.

[27] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Foundations Trends in Machine Learning*, vol. 1, pp. 1–305, 2008.

[28] P. Clifford, "Markov random fields in statistics," *Disorder in physical systems*, pp. 19–32, 1990.

[29] C. M. Bishop, *Pattern Recognition and Machine Learning.* Springer, Oct. 2007.

[30] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques.* The MIT Press, Jul. 2009.

[31] B. Potetz and T. S. Lee, "Efficient belief propagation for higher-order cliques using linear constraint nodes," *Computer Vision and Image Understanding*, vol. 112, no. 1, pp. 39–54, Oct. 2008.

[32] Y. Weiss and W. Freeman, "On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 736–744, Feb. 2001.

[33] N. Shulman, "Communication over an unknown channel via common broadcasting," Ph.D. dissertation, Tel-Aviv University, 2003.

[34] N. C. Wormald, "Models of random regular graphs," *London Mathematical Society Lecture Note Series*, pp. 239–298, 1999.

[35] A. Aaron, S. D. Rane, E. Setton, and B. Girod, "Transform-domain Wyner-Ziv codec for video," in *Proceedings of SPIE*, vol. 5308, 2004, pp. 520–528.

[36] M. Johnson, P. Ishwar, V. Prabhakaran, D. Schonberg, and K. Ramchandran, "On compressing encrypted data," *IEEE Transactions on Signal Processing*, vol. 52, no. 10, pp. 2992–3006, Oct. 2004.

[37] D. Schonberg, S. Draper, and K. Ramchandran, "On compression of encrypted images," in *Proceedings of the International Conference on Image Processing*, 2006, pp. 269–272.

[38] S. Jalali, S. Verdú, and T. Weissman, "A universal scheme for Wyner-Ziv coding of discrete sources," *IEEE Transactions on Information Theory*, vol. 56, no. 4, pp. 1737–1750, Apr. 2010.

[39] D. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[40] L. He and L. Carin, “Exploiting structure in wavelet-based bayesian compressive sensing,” *IEEE Transactions on Signal Processing*, vol. 57, no. 9, pp. 3488–3497, 2009.

[41] T. Richardson and R. Urbanke, *Modern Coding Theory.* Cambridge University Press, Mar. 2008.

[42] M. Wainwright, E. Maneva, and E. Martinian, “Lossy source compression using low-density generator matrix codes: Analysis and algorithms,” *IEEE Transactions on Information Theory*, vol. 56, no. 3, pp. 1351–1368, Mar. 2010.

[43] V. Chandar, D. Shah, and G. W. Wornell, “A simple message-passing algorithm for compressed sensing,” in *2010 IEEE International Symposium on Information Theory Proceedings (ISIT)*, Jun. 2010, pp. 1968–1972.

[44] V. B. Chandar, “Sparse graph codes for compression, sensing, and secrecy,” Ph.D. dissertation, Massachusetts Institute of Technology, 2010.

[45] M. G. Reyes, “Cutset based processing and compression of markov random fields,” Ph.D. dissertation, The University of Michigan, 2011.

[46] H. D. Pfister, “On the capacity of finite state channels and the analysis of convolutional accumulate-m codes,” Ph.D. dissertation, University of California, San Diego, 2003.

[47] G. Colavolpe, “On LDPC codes over channels with memory,” *IEEE Transactions on Wireless Communications*, vol. 5, no. 7, pp. 1757–1766, 2006.

[48] R. Koetter, A. Singer, and M. Tüchler, “Turbo equalization,” *IEEE Signal Processing Magazine*, vol. 21, no. 1, pp. 67–80, Jan. 2004.

[49] J. Langdon, G.G., “An introduction to arithmetic coding,” *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 135–149, Mar. 1984.

[50] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, May 1977.

[51] ——, “Compression of individual sequences via variable-rate coding,” *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, Sep. 1978.

[52] F. M. J. Willems, Y. Shtarkov, and T. Tjalkens, “The context-tree weighting method: basic properties,” *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 653–664, May 1995.

[53] P. Howard, F. Kossentini, B. Martins, S. Forchhammer, and W. Rucklidge, “The emerging JBIG2 standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 7, pp. 838–848, Nov. 1998.

[54] M. Kardar, “8.334 statistical mechanics II notes, lecture 18.” MIT OCW, 2008.

[55] D. Cimasoni, "A generalized Kac-Ward formula," *J. Stat. Mech.*, 2010.

[56] A. Galluccio, "New algorithm for the ising problem: Partition function for finite lattice graphs," 1999.

[57] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 619–637, 2001.

[58] S. Kudekar, T. Richardson, and R. Urbanke, "Threshold saturation via spatial coupling: Why convolutional LDPC ensembles perform so well over the BEC," *IEEE Transactions on Information Theory*, vol. 57, no. 2, pp. 803–834, Feb. 2011.

[59] D. Truhachev, D. G. Mitchell, M. Lentmaier, and D. J. Costello, "New codes on graphs constructed by connecting spatially coupled chains," in *Information Theory and Applications Workshop (ITA), 2012*, 2012, pp. 392–397.

[60] H. Pfister, I. Sason, and R. Urbanke, "Capacity-achieving ensembles for the binary erasure channel with bounded complexity," *IEEE Transactions on Information Theory*, vol. 51, no. 7, pp. 2352–2379, Jul. 2005.

[61] H. Pishro-Nik and F. Fekri, "On decoding of low-density parity-check codes over the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 50, no. 3, pp. 439–454, Mar. 2004.

[62] I. Andriyanova and J. Tillich, "Designing a good low-rate sparse-graph code," *IEEE Transactions on Communications*, vol. 60, no. 11, pp. 3181–3190, Nov. 2012.

[63] D. Burshtein and G. Miller, "An efficient maximum-likelihood decoding of LDPC codes over the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 50, no. 11, pp. 2837–2844, 2004.

[64] S. ten Brink, "Designing iterative decoding schemes with the extrinsic information transfer chart," *AEU Int. J. Electron. Commun*, vol. 54, no. 6, pp. 389–398, 2000.

[65] C. Measson, A. Montanari, and R. Urbanke, "Maxwell construction: The hidden bridge between iterative and maximum a posteriori decoding," *IEEE Transactions on Information Theory*, vol. 54, no. 12, pp. 5277–5307, Dec. 2008.

[66] I. Csiszár and J. Körner, *Information theory: coding theorems for discrete memoryless systems.* Academic Press, 1981.

[67] Y. Choi and W. Szpankowski, "Compression of graphical structures: Fundamental limits, algorithms, and experiments," *IEEE Transactions on Information Theory*, vol. 58, no. 2, pp. 620–638, Feb. 2012.

[68] Y. Baryshnikov, J. Duda, and W. Szpankowski, "Markov field types and tilings," in *2014 IEEE International Symposium on Information Theory (ISIT)*, Jun. 2014, pp. 2639–2643.

[69] M. Li, *An Introduction to Kolmogorov Complexity and Its Applications.* Springer, Nov. 2008.

[70] R. G. Downey, D. R. Hirschfeldt, and G. L. Forte, "Randomness and reductibility," in *Mathematical Foundations of Computer Science 2001.* Springer Berlin Heidelberg, Jan. 2001, no. 2136, pp. 316–327.

[71] G. J. Chaitin, "Information-theoretic computation complexity," *IEEE Transactions on Information Theory*, vol. 20, no. 1, pp. 10–15, Jan. 1974.

[72] T. M. Cover and J. A. Thomas, *Elements of information theory.* Wiley, 1991.

[73] J. G. Cleary and I. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Transactions on Communications*, vol. 32, no. 4, pp. 396–402, Apr. 1984.

[74] G. V. Cormack and R. N. S. Horspool, "Data compression using dynamic markov modelling," *The Computer Journal*, vol. 30, no. 6, pp. 541–550, 1987.

[75] J. Rissanen, "A universal data compression system," *IEEE Transactions on Information Theory*, vol. 29, no. 5, pp. 656–664, Sep. 1983.

[76] S. Deorowicz, "Universal lossless data compression algorithms," Ph.D. dissertation, Silesian University of Technology, 2003.

[77] T. Matsuta, T. Uyematsu, and R. Matsumoto, "Universal Slepian-Wolf source codes using low-density parity-check matrices," in *2010 IEEE International Symposium on Information Theory Proceedings (ISIT)*, Jun. 2010, pp. 186–190.

[78] Z. Ghahramani, "Graphical models: parameter learning," *Handbook of Brain Theory and Neural Networks*, vol. 2, pp. 486–490, 2002.

[79] P. Abbeel, D. Koller, and A. Y. Ng, "Learning factor graphs in polynomial time and sample complexity," *The Journal of Machine Learning Research*, vol. 7, pp. 1743–1788, 2006.

[80] L. Ruschendorf, "Convergence of the iterative proportional fitting procedure," *Ann. Statist.*, vol. 23, no. 4, pp. 1160–1174, Aug. 1995.

[81] S. Scholl, F. Kienle, M. Helmling, and S. Ruzika, "ML vs. BP decoding of binary and non-binary LDPC codes," in *Turbo Codes and Iterative Information Processing (ISTC), 2012 7th International Symposium on.* IEEE, 2012, pp. 71–75.

[82] N. Chang, "Rate adaptive non-binary LDPC codes with low encoding complexity," in *2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, Nov. 2011, pp. 664–668.

[83] Y. Zhao and J. Garcia-Frias, "Data compression of correlated non-binary sources using punctured turbo codes," in *Data Compression Conference, 2002. Proceedings. DCC 2002*, 2002, pp. 242–251.

[84] D. H. Schonberg, "Practical distributed source coding and its application to the compression of encrypted data," Ph.D. dissertation, University of California, Berkeley, 2007.

[85] Y. Xu, "Potts model and generalizations: Exact results and statistical physics," Ph.D. dissertation, Stony Brook University, 2012.

[86] M. Sakalli, W. Pearlman, and M. Farshchian, "SPIHT algorithms using depth first search algorithm with minimum memory usage," in *2006 40th Annual Conference on Information Sciences and Systems*, Mar. 2006, pp. 1158–1163.

[87] T. Berger, *Rate Distortion Theory: Mathematical Basis for Data Compression.* Prentice Hall, Oct. 1971.

[88] A. Wyner and J. Ziv, "The rate-distortion function for source coding with side information at the decoder," *IEEE Transactions on Information Theory*, vol. 22, no. 1, pp. 1–10, Jan. 1976.

[89] A. D. Wyner, "The rate-distortion function for source coding with side information at the decoder-II: General sources," *Information and Control*, vol. 38, no. 1, pp. 60–80, Jul. 1978.

[90] S. Pradhan and K. Ramchandran, "Geometric proof of rate-distortion function of gaussian sources with side information at the decoder," in *IEEE International Symposium on Information Theory, 2000. Proceedings*, 2000, pp. 351–.

[91] P. Chou, T. Lookabaugh, and R. Gray, "Entropy-constrained vector quantization," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 1, pp. 31–42, Jan. 1989.

[92] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, Mar. 1982.

[93] M. J. Sabin and R. Gray, "Global convergence and empirical consistency of the generalized lloyd algorithm," *IEEE Transactions on Information Theory*, vol. 32, no. 2, pp. 148–155, Mar. 1986.

[94] P. Chou, T. Lookabaugh, and R. Gray, “Optimal pruning with applications to tree-structured source coding and modeling,” *IEEE Transactions on Information Theory*, vol. 35, no. 2, pp. 299–315, Mar. 1989.

[95] E. Riskin and R. Gray, “A greedy tree growing algorithm for the design of variable rate vector quantizers,” *IEEE Transactions on Signal Processing*, vol. 39, no. 11, pp. 2500–2507, Nov. 1991.

[96] M. R. Carbonara, J. E. Fowler, and S. C. Ahalt, “Compression of digital video data using artifical neural network differential vector quantization,” Sep. 1992, pp. 422–433.

[97] A. H. Dekker, “Kohonen neural networks for optimal colour quantization,” *Network: Computation in Neural Systems*, vol. 5, no. 3, pp. 351–367, 1994.

[98] A. Aaron, E. Setton, and B. Girod, “Towards practical Wyner-Ziv coding of video,” in *2003 International Conference on Image Processing, 2003. ICIP 2003. Proceedings*, vol. 3, Sep. 2003, pp. III–869–72 vol.2.

[99] D. Rebollo-Monedero and B. Girod, “Generalization of the rate-distortion function for Wyner-Ziv coding of noisy sources in the quadratic-gaussian case,” in *Data Compression Conference, 2005. Proceedings. DCC 2005.* IEEE, 2005, pp. 23–32.

[100] D. Kubasov, K. Lajnef, and C. Guillemot, “A hybrid encoder/decoder rate control for Wyner-Ziv video coding with a feedback channel,” in *IEEE 9th Workshop on Multimedia Signal Processing, 2007. MMSP 2007*, Oct. 2007, pp. 251–254.

[101] D.-k. He, A. Jagmohan, L. Lu, and V. Sheinin, “Wyner-Ziv video compression using rateless LDPC codes,” in *Proc. VCIP*, vol. 8, 2008.

[102] A. Buzo, J. Gray, A, R. Gray, and J. Markel, “Speech coding based upon vector quantization,” *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 28, no. 5, pp. 562–574, Oct. 1980.

[103] L. R. Rabiner, M. M. Sondhi, and S. E. Levinson, “Note on the properties of a vector quantizer for LPC coefficients,” *Bell System Technical Journal*, vol. 62, no. 8, pp. 2603–2616, Oct. 1983.

[104] S. Mehrotra, W.-g. Chen, and K. Kotteri, “Low bitrate audio coding using generalized adaptive gain shape vector quantization across channels,” in *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on.* IEEE, 2009, pp. 9–12.

[105] T. Sreenivas and M. Dietz, “Vector quantization of scale factors in advanced audio coder (AAC),” in *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, 1998*, vol. 6, May 1998, pp. 3641–3644 vol.6.

[106] T. Painter and A. Spanias, "Perceptual coding of digital audio," *Proceedings of the IEEE*, vol. 88, no. 4, pp. 451–515, 2000.

[107] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG 2000 still image compression standard," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 36–58, Sep. 2001.

[108] H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-complexity transform and quantization in h.264/AVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 598–603, Jul. 2003.

[109] E. Martinian, "Dynamic information and constraints in source and channel coding," Ph.D. dissertation, Massachusetts Institute of Technology, 2004.

[110] M. Wainwright and E. Maneva, "Lossy source encoding via message-passing and decimation over generalized codewords of LDGM codes," in *International Symposium on Information Theory, 2005. ISIT 2005. Proceedings*, Sep. 2005, pp. 1493–1497.

[111] E. Martinian and M. J. Wainwright, "Analysis of LDGM and compound codes for lossy compression and binning," *arXiv preprint cs/0602046*, 2006.

[112] G. Demay, V. Rathi, and L. K. Rasmussen, "Optimality of LDGM-LDPC compound codes for lossy compression of binary erasure source," in *Information Theory and its Applications (ISITA), 2010 International Symposium on.* IEEE, 2010, pp. 589–594.

[113] A. Gupta and S. Verdú, "Nonlinear sparse-graph codes for lossy compression," *IEEE Transactions on Information Theory*, vol. 55, no. 5, pp. 1961–1975, May 2009.

[114] L. Cappellari, "Lossy source compression of non-uniform binary sources using GQ-LDGM codes," in *Information Theory Workshop (ITW), 2010 IEEE.* IEEE, 2010, pp. 1–5.

[115] Z. Sun, M. Shao, J. Chen, K. M. Wong, and X. Wu, "Achieving the rate-distortion bound with low-density generator matrix codes," *IEEE Transactions on Communications*, vol. 58, no. 6, pp. 1643–1653, Jun. 2010.

[116] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing.* ACM, 1998, pp. 604–613.

[117] P. T. Boufounos and S. Rane, "Efficient coding of signal distances using universal quantized embeddings," in *Data Compression Conference (DCC), 2013.* IEEE, 2013, pp. 251–260.

[118] S. Rane, P. Boufounos, and A. Vetro, "Quantized embeddings: an efficient and universal nearest neighbor method for cloud-based image retrieval," vol. 8856, 2013, pp. 885 609–885 609–11.

[119] P. Boufounos, "Universal rate-efficient scalar quantization," *IEEE Transactions on Information Theory*, vol. 58, no. 3, pp. 1861–1872, Mar. 2012.

[120] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression.* Springer, Nov. 1991.

[121] A. Dembo and I. Kontoyiannis, "Source coding, large deviations, and approximate pattern matching," *IEEE Transactions on Information Theory*, vol. 48, no. 6, pp. 1590–1615, Jun. 2002.

[122] Z. Sun, T. Tan, Y. Wang, and S. Li, "Ordinal palmprint representation for personal identification," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2005, pp. 279–284.

[123] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278 –2324, Nov. 1998.

[124] P. Fränti and E. Ageenko, "On the use of context tree for binary image compression," in *Proceedings of the International Conference on Image Processing*, vol. 3, 1999, pp. 752–756.

[125] Y. Weiss and W. T. Freeman, "What makes a good model of natural images?" in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, 2007, pp. 1–8.

[126] D. Zoran and Y. Weiss, "Natural images, gaussian mixtures and dead leaves," in *Advances in Neural Information Processing Systems*, 2012, pp. 1736–1744.

[127] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[128] R. J. McEliece, "A public-key cryptosystem based on algebraic coding theory," 1978.

[129] G. Chechik, A. Globerson, N. Tishby, and Y. Weiss, "Information bottleneck for gaussian variables," *J. Mach. Learn. Res.*, vol. 6, pp. 165–188, Dec. 2005.

[130] O. Bursalioglu, G. Caire, and D. Divsalar, "Joint source-channel coding for deep-space image transmission using rateless codes," *IEEE Transactions on Communications*, vol. 61, no. 8, pp. 3448–3461, Aug. 2013.

[131] O. Y. Bursalioglu, M. Fresia, G. Caire, and H. V. Poor, "Lossy joint source-channel coding using raptor codes," *International Journal of Digital Multimedia Broadcasting*, vol. 2008, p. e124685, Sep. 2008.

[132] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView: IDC Analyze the Future*, 2012.