Practical Compression with Model-Code Separation

Ying-zong Huang and Gregory W. Wornell
Massachusetts Institute of Technology
Dept. of Electrical Engineering and Computer Science
Cambridge, MA 02139
{zong,gww}@mit.edu

Abstract

Two aspects of compression — data modeling and coding — are not always conceived as distinct, nor implemented as such in current compression systems, leading to difficulties of an architectural nature. This work contributes an alternative "model-code separation" architecture for compression, based on iterative message-passing algorithms over graphical models representing the modeling and coding aspects of compression. Systems following this architecture resolve important challenges posed by current systems, and stand to benefit further from advances in the understanding of data and the algorithms that process them.

1 Introduction

In compression, data modeling refers to introducing prior knowledge, i.e. data model, into the system; coding refers to assigning the compressed output to each input. In a *joint model-code architecture*, the two tasks are treated as one. The classical random codebook scheme of Shannon source coding theory is an example. Current systems, which often take the form of Fig. 1, likewise have this architecture despite the appearances of encoder decomposition; upon inspection, there is no clear delineation that isolates data modeling from compressive coding.

In terms of system design, a joint model-code architecture is limiting. It embeds design-time assumptions, particularly about the data model, deeply into the entire system, and it requires a great deal of expert cleverness to realize a new system for each data type. Many domains now generate prolific datasets with complex statistical structure, e.g. financial time

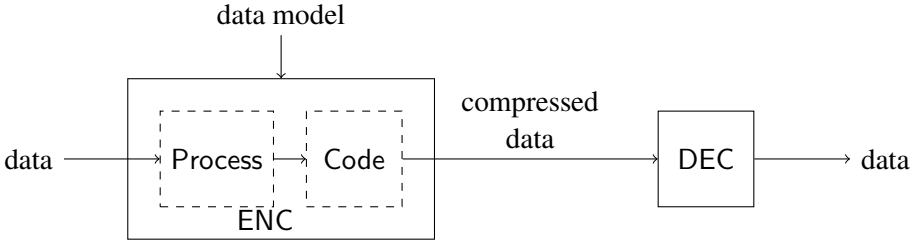


Figure 1: Current compression systems have a joint model-code architecture. Process (often a representational transform) removes the bulk of the structural redundancy from data, while Code (entropy coding) compresses simple, usually memoryless, residual structures. The data model is used to design both Process and Code together.

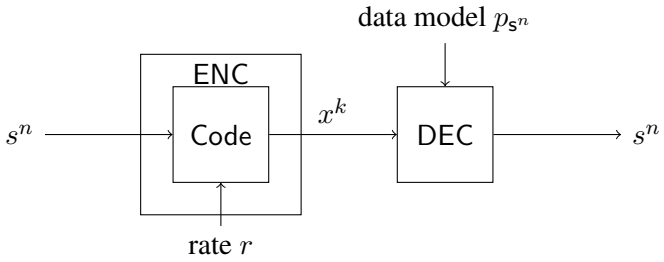


Figure 2: System diagram for the model-code separation architecture. On the left is a model-free encoder. On the right is an inferential decoder.

series, bioscience data, search/ranking/social media databases, etc., and joint architecture design for such data is challenging. Computing paradigms too are evolving, e.g. towards mobile acquisition devices backed by cloud servers on a network, and demand a flexibility with respect to information and complexity allocation within the system pipeline. Some important requirements for contemporary and future compression can benefit from new architectural thinking, among them —

(1) *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. (2) *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, in particular in the data model. (3) *Security*: If the data is stored with an untrusted third party provider, we want it to be compressible in the encrypted domain, securely. (4) *Mobility*: We want our power constrained mobile devices to have state-of-the-art compression capability even with a lightweight encoder. (5) *Distributivity*: Our data may be distributed across a network of machines or sensors, and we want them to be locally compressible. (6) *Robustness*: Networks may corrupt compressed data with losses, and we should allow recovery mechanisms without changing the compression mechanism.

Contribution and prior work — This paper presents a practical, alternative *model-code seperation architecture* for general data compression (Fig. 2). The architecture divides into a part for data modelers and a part for coding theorists, mediating them using the common language of graphs and message-passing algorithms. In the canonical realization, a *model-free encoder* only makes choices about coding, and produces a compressed bit-stream agnostic to the data model (or indeed, to any information-preserving transformation of the source); it instead leaves to an *inferential decoder* to apply the data model. Separation allows us the flexibility to alter the data model without affecting the underlying redundancy-extraction machinery (i.e. coding). The compressed stream on which standards are defined is also a true information stream that can be re-ordered, partially lost, accumulated from multiple origins, etc. Architecturally, separation incurs no performance penalty, and practically we show separation systems obtaining performance competitive with and sometimes superior to existing solutions.

We are aware of no prior work offering model-code separation as a practical architecture for general compression, though it must have been known as a theoretical possibility: the Slepian-Wolf random binning scheme applied to a single source has this architecture. Sub-

sequent applications have been in decoder side-information problems that feature partial data model separation from coding (though not stated as such), and involve some inferential decoding, e.g. in Wyner-Ziv video coding [1], encrypted compression [2, 3], universal Lempel-Ziv with refinement [4], and other "data-like" side-information problems; and in sparse reconstruction [5], Bayesian compressed sensing [6], and other "structure-like" side-information problems. More recently, graphical model representations and coding algorithms have also converged in signal reconstruction problems, e.g. in compressing memoryless sources [7], in image reconstruction given "code-like" side information [8], in sensing [9], in lossy compression [10], and in the dual problem of coding for channels with memory [11, 12, 13]. Our work brings these constructions together in a new light.

2 Basic system

The model-code separation system here presented is a practical realization of the otherwise NP-hard Slepian-Wolf scheme for arbitrary sources. In Slepian-Wolf, we see key components that enable model-code separation: randomized bin assignment as *model-free coding*, and typicality as *model representation*. We replace these with two lower complexity analogs: low-density parity-check (LDPC) codes for coding, and probabilistic graphical models (PGM) for model representation. These are good choices for two reasons: (1) low-complexity message-passing methods are empirically good for LDPC decoding and data inference individually, and there is reason to believe the same remains the case for compression [14]; (2) without a complexity constraint, linear codes achieve entropy rate for general sources [7] and exact inference can be computed on PGM's, so no architectural loss is incurred for these choices.

In this section objects are over the alphabet $\mathbf{S} = \mathbf{GF}(q)$. To compress an n-vector sequence $\mathrm{s}^n \in \mathbf{S}^n$, we assume (1) a stochastic data model $\mathsf{s}^n \sim p_{\mathsf{s}^n}$ from which s^n is drawn; (2) a coding ensemble $\mathscr{H}(n,k)$ of $k \times n$ parity matrices of a rate k/n LDPC source code (i.e. rate $1 - k/n$ LDPC channel code). Construct the following encoder and decoder.

2.1 Encoder

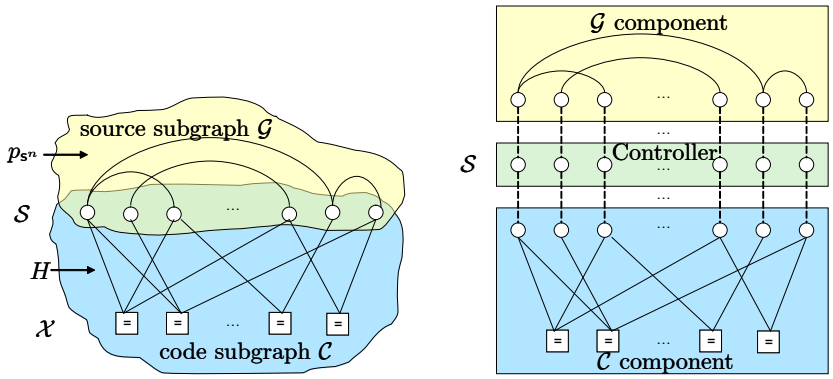
Setting k to target a nominal rate $r_{\text{code}} = k/n$, and choosing a random $H \in \mathscr{H}(n,k)$, produce a hash $\mathrm{x}^k = H\mathrm{s}^n$ as the compressed result.

2.2 Decoder

Code and source subgraphs — H is represented by a factor graph $\mathcal{C} = (\mathcal{S}, \mathcal{X}, \mathcal{F})$ where $\mathcal{S} \triangleq \{\mathsf{s}_1, ..., \mathsf{s}_n\}$ and $\mathcal{X} \triangleq \{f_1, ..., f_k\}$ are respectively source and factor nodes. An edge connects $f_a \in \mathcal{X}$ and $\mathsf{s}_i \in \mathcal{S}$ if and only if $H_{a,i} \neq 0$, forming the neighborhood $\mathcal{N}_a^{\mathcal{C}}$ of f_a. The *code subgraph* $\mathcal{C}$ models the *hash constraint function*

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

where $c(s^n) = 1$ if and only if all constraints are satisfied. This graph is identical to that used in LDPC channel code decoding.



(a) (b)

Figure 3: (a) The structure of the combined decoder source-code graph $\mathcal{U}$ for the system in Section 2. (b) The differently colored subgraphs behave as if modular components connected via dashed virtual ports.

p_{s^n} is represented by an undirected *source subgraph* $\mathcal{G} = (\mathcal{S}, \mathcal{E})$ over the maximal cliques of which it factors:

$$p_{\mathsf{s}^n}(s^n) = \frac{1}{\mathcal{Z}} \prod_{C \in \mathrm{cl}(\mathcal{G})} \psi_C(s_C) \tag{2}$$

In the interest of clarity, the sequel only describes algorithms over pairwise models where each $\psi_C(\cdot)$ is unary or binary:

$$p_{\mathsf{s}^n}(s^n) = \frac{1}{\mathcal{Z}} \prod_{\mathsf{s}_i \in \mathcal{S}} \phi_i(s_i) \prod_{(\mathsf{s}_i, \mathsf{s}_j) \in \mathcal{E}} \psi_{ij}(s_i, s_j) \tag{3}$$

Decoding algorithm — Let $\mathcal{U} \triangleq \mathcal{G} \cup \mathcal{C} = (\mathcal{S}, \mathcal{X}, \mathcal{E} \cup \mathcal{F})$ be a combined source-code graph, where the source nodes $\mathcal{S}$ are shared between the source and code subgraphs (Fig. 3a). The decoder runs belief propagation (BP) on $\mathcal{U}$, representing approximate maximization on the *joint objective*

$$u(s^n) \triangleq c(s^n) p_{\mathsf{s}^n}(s^n) \tag{4}$$

The full BP consists of iteratively computing local functions called *messages* to and from neighbor nodes called *ports*. The following are the main nodal computations for decoding ($\mu^{j \leftarrow i}$ is a source-side message in the variable s_j at node s_i out on port s_j; $m^{i \leftarrow a}$ and $m^{i \rightarrow a}$ are respectively a code-side message in the variable s_i at node f_a on port s_i, and one at node s_i on port f_a; $\mathcal{N}_i^{\mathcal{G}}, \mathcal{N}_i^{\mathcal{C}}, \mathcal{N}_a^{\mathcal{C}}$ are respetively neighborhoods of s_i in $\mathcal{G}$, s_i in $\mathcal{C}$, and f_a in $\mathcal{C}$):

$$\mu^{j \leftarrow i}(s_j) \Leftarrow \sum_{s_i} \psi_{ij}(s_i, s_j) \phi_i(s_i) \prod_{\mathsf{s}_k \in \mathcal{N}_i^{\mathcal{G}} \setminus \mathsf{s}_j} \mu^{i \leftarrow k}(s_i) \left[\prod_{f_a \in \mathcal{N}_i^{\mathcal{C}}} m^{i \leftarrow a}(s_i) \right] \tag{5}$$

$$m^{i \rightarrow a}(s_i) \Leftarrow \prod_{f_b \in \mathcal{N}_i^{\mathcal{C}} \setminus f_a} m^{i \leftarrow b}(s_i) \left[\prod_{\mathsf{s}_j \in \mathcal{N}_i^{\mathcal{G}}} \phi_i(s_i) \mu^{i \leftarrow j}(s_i) \right] \tag{6}$$

$$m^{i\leftarrow a}(s_i) \;\Leftarrow\; \sum_{\mathcal{N}_a^{\mathcal{C}}\setminus \mathbf{s}_i} f_a(s_{\mathcal{N}_a^{\mathcal{C}}}) \prod_{\mathbf{s}_j\in\mathcal{N}_a^{\mathcal{C}}\setminus \mathbf{s}_i} m^{j\to a}(s_j) \tag{7}$$

Additionally, there is a total belief computation and marginal decoding

$$\hat{s}_i = \arg\max_{s_i} \left[\prod_{f_a\in\mathcal{N}_i^{\mathcal{C}}} m^{i\leftarrow a}(s_i)\right]\left[\prod_{\mathbf{s}_j\in\mathcal{N}_i^{\mathcal{G}}} \phi_i(s_i)\mu^{i\leftarrow j}(s_i)\right] \tag{8}$$

BP is run over multiple iterations until convergence or declaration of failure.

Decoding complexity scales with both coding and data model complexity. If $\mathcal{C}$ has largest factor degree ρ, and $\mathcal{G}$ 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 LDPC codes and all the finite-order models we see later in this paper, complexity is $\mathcal{O}(n)$.

Modular architecture — The bracketed terms of (5) and (6) may be conceptualized as single lumped messages that do not change with respect to which port the output is emitted on. Therefore, when a node of $\mathcal{S}$ emits 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 entire system architecturally modular, with external messages and virtual ports acting as the only interface between graph-inferential components. Finally, while each component can compute a total belief (8) using these external messages, it makes sense for a main *controller* to handle component-agnostic operations. The functional factorization and suggested interfaces are given in Fig. 3b.

2.3 Schedule, doping, rate selection

BP scheduling is much simplified by the modular architecture, which focuses decisions at the level of component interactions. A parallel schedule would have each component compute internal messages, then exchange external messages at the same time. A serial schedule would have one component active at a time, and present its latest external messages to the next component. This serial 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.

Depending on the data model, the decoding process may not begin without initial beliefs. Therefore the encoder may randomly select a fraction r_{dope} of source nodes $\mathcal{D} \subseteq \mathcal{S}$ to describe directly to the decoder. In decoding these are presented as deterministic messages $d^{\mathcal{D}}(s_{\mathcal{D}}) = \mathbb{1}_{\{s_{\mathcal{D}}=\mathbf{s}_{\mathcal{D}}\}}(s_{\mathcal{D}})$ in the controller and multiplied into all external messages passing through it. These "doping" symbols also anchor the decoding process, and only a small amount — which can be optimized — is necessary. If we consider doping as a part of the code, where we augment H with additional unit-weight checksum rows to rate $r = r_{\text{code}} + r_{\text{dope}}$, then doping can be jointly optimized with the code.

The system presented can be used in many ways. For the direct fixed-rate, no-feedback setting, the model-free encoder needs the compression rate to be specified. This may come

from entropy estimates or upstream hints, analogous to capacity estimates in channel coding. However, nothing precludes building a traditional zero-error variable-rate system if we wish to bind the data model in the encoder. The presented encoder can simply be augmented with a decoder simulation and choose the rate at which the simulated decoding succeeds. If feedback is available, the decoder can acknowledge sufficiency as x^k is sent letter-by-letter. This immediately obtains a zero-error rateless system. In a broadcast setting, even feedback is not necessary [15]. 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.

3 Large alphabets and processed data

The basic compression system of Section 2 assumes the data and the code are over the same field $\mathbf{S} = \mathbf{GF}(q)$. While the study of non-binary LDPC coding is progressing, 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 neglects correlation between bit planes. Instead, we take the more general approach of inserting into the decoder an additional subgraph that models how symbols of s^n are represented in another alphabet.

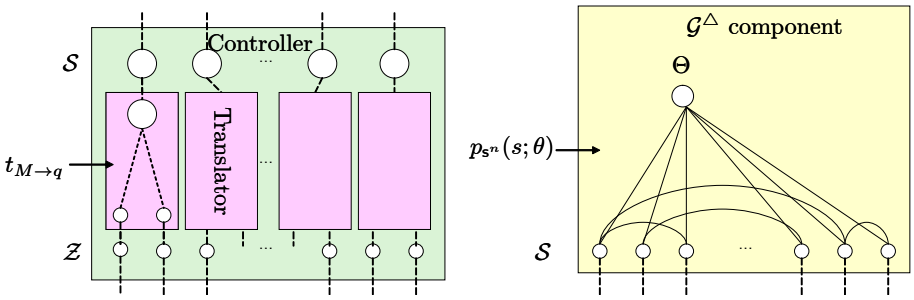
Suppose $s^n = \{s_1, ..., s_n\}$ is an abstract n-symbol data sequence serialized by symbol-level *representational maps*. For ease of discussion, we assume all s_i belong to the same alphabet $\mathbf{S}$ of size M, and so there is one map for all n symbols, though this need not be the case. The representation 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.

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 $\beta \in \mathbf{GF}(q)$, $T_{M\to q}(m^{(M)})_\omega(\beta) \triangleq \sum_{\alpha\in\mathcal{S}} m^{(M)}(\alpha)\mathbf{1}\{t_{M\to q}(\alpha)_\omega = \beta\}$; and for $\alpha \in \mathbf{S}$, $T_{q\to M}(m^{(q)})(\alpha) \triangleq \prod_{\omega=1}^{B} m_\omega^{(q)}(t_{M\to q}(\alpha)_\omega)$.

Now we have the tools to compress large-alphabet data. Given a binary LDPC parity matrix $H \in \mathbf{GF}(2)^{k\times nB}$ and rate $r_{\text{code}} = k/nB$, the compressed output is

$$\mathrm{x}^k = H\mathrm{z}^{nB} = Ht_{M\to 2}(\mathrm{s}^n) \tag{9}$$

Decoding takes into account this additional serialization step in (or prior to) the encoder, and applies the message translation functions during BP. It is unnecessary to work out what replaces (5)-(8), due to modularity. Architecturally, message translation is an isolated function. Recalling Fig. 3b, we can consider this construction as a decoder extension by the addition of a batch of message translator components that interface with an updated



(a) Message translator (b) Augmented data model

Figure 4: In extensions of the basic model-code separation architecture, new modular components may be added or swapped into the decoder to compress (a) large alphabet data, or (b) with a parametric model.

controller that holds both the source symbols $\mathcal{S} = \{\mathsf{s}_1, ..., \mathsf{s}_n\}$ and the serialized bits $\mathcal{Z} = \{\mathsf{z}_{i,1}, \mathsf{z}_{i,2}, ..., \mathsf{z}_{i,B}\}$ (Fig. 4a). The controller further interfaces to the usual, unaltered source component over the alphabet $\mathbf{S}$ and code component over $\mathbf{GF}(2)$.

This design is useful for many more processing situations than just translating between alphabets. The controller is 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 the nodes, such as may be induced by encryption, quantization, channel coding, etc.

4 Uncertain models

Graphical models can represent model uncertainty very naturally. We consider two cases: (1) when there is a mismatch between the "true" model and the assumed model, (2) when the "true" model is "partially known," i.e., as belonging to a parametric family of models, and we may only assume it is one among the family.

***Model mismatch* —** Compressing using the incorrect data model results in the performance degradation suggested by large deviation theory. Given input data drawn from a source distributed as $p_{\mathsf{s}^n}(s^n)$ but decoding using a source subgraph modeling $q(s^n)$, is equivalent to approximate maximization of $u'(s^n) \triangleq c(s^n)q(s^n)$. Theory suggests for exact maximization that coding takes place at rate approaching

$$R(p_{\mathsf{s}^n}, q) \triangleq \lim_{n\to\infty} \frac{1}{n}\mathbb{E}(-\log q(\mathsf{s}^n)) = \mathbb{H}(\mathsf{s}) + \frac{1}{n}D(p_{\mathsf{s}^n}||q) \tag{10}$$

Since this rate serves the same role as entropy rate does when there is no mismatch, we expect and indeed find similar relative performance.

***Parametric models* —** Suppose we are to compress a $\mathbf{Bern}(\theta)$ source with unknown $\theta \in [0, 1]$. While θ is deterministic, we can construct a Bayesian belief $p_\Theta(\theta)$ for it, and 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

$$p_{\mathsf{s}^n\Theta}(s^n, \theta) = p_\Theta(\theta)\prod_{i=1}^{n} p_{\mathsf{s}_i|\Theta}(s_i|\theta) \tag{11}$$

Given a graph $\mathcal{G}$ for a $\mathbf{Bern}(p)$ data model, we can augment the nodes $\mathcal{S} = \{\mathsf{s}_1, ..., \mathsf{s}_n\}$ by a node Θ, connected to all of them via additional factors $\pi(s_i, \theta)$, in the *augmented graphical model* $\mathcal{G}^{\triangle}$ for (s^n, Θ). Fig. 4b shows a component form. If we connect $\mathcal{G}^{\triangle}$ with the code graph $\mathcal{C}$, we can run BP to optimize over the new objective $u^{\triangle}(s^n, \theta) \triangleq c(s^n) p_{\mathsf{s}^n\Theta}(s^n, \theta)$. We will marginalize over both each s_i and Θ, but of course we only care about the former — the latter estimate of θ is a side effect.

This construction can be extended in many directions of practical interest. For example, we can have a prior on Θ itself. It can be extended to offline or online learning, by retaining the belief on Θ across multiple samples. It can be extended to hierarchical models (e.g. hidden Markov models), taking hidden variables as unknown parameters.

5 Performance

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

- `SEP-prot`: An model-code separation system with off-the-shelf library of binary LDPC codes (H column weight 3, quasi-regular). Doping rate r_{dope} is fixed and noted. Rate performance denotes the minimal $r_{\text{code}} + r_{\text{dope}}$ for which decoding converges (in this case, within 150 iterations) to the correct result.

- `SEP-thresh`: The adjusted rate performance $\epsilon^{\text{BP}} + r_{\text{dope}}$ associated with `SEP-prot`. This can be viewed as another (idealized) system instance. The BP decoding threshold ϵ^{BP} of an LDPC code [16] serves as a better proxy for the "utilizable" rate of a code than r_{code}, in that $r_{\text{code}} - \epsilon^{\text{BP}}$ is not primarily an architectural loss, but that associated with code selection and decoding method. Coding optimization such as with [17, 18] can be expected to close the gap.

- `ARITH`: Arithmetic coding with symbol probabilities supplied.

- `GZIP`: Lempel-Ziv class universal compressor. Input data is pre-flattened to a bitstream. Output length is the compressed file size, less the compressed file size of a zero-length file to account for overhead.

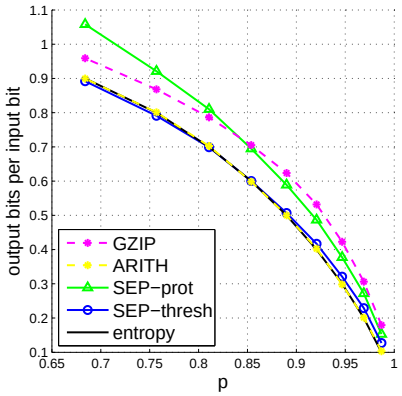
- `CTW`: Context-tree weighting universal compressor. Input data is pre-flattened.

- `JBIG2`: Bi-level image compressor (ITU-T T.88; 2000) in lossless mode (based on 2D context dictionaries). Output length is compressed file size, less the compressed file size of a 1-pixel image to account for overhead.

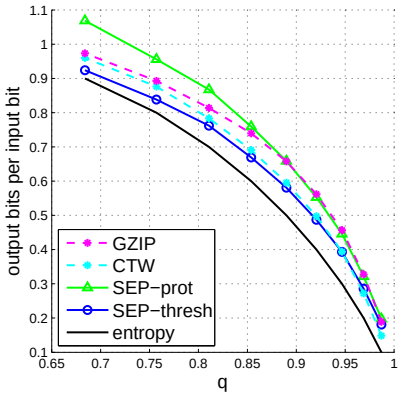
Universal compressors are included for reference, in particular for when there are no better matched compression algorithms

The results for Fig. 5a-d are respectively for the Bernoulli source parameterized by its bias p, the binary Markov source parameterized by recurrence probability q, the 2D lattice

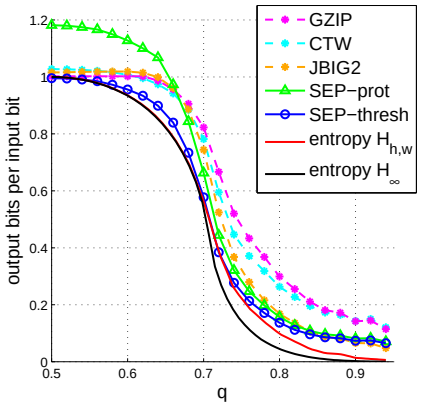
[1] `ARITH` is implemented in MATLAB's Communications System Toolbox (v5.5) as `arithenco`; `GZIP` is implemented in standard GNU distributions as `gzip` (v1.5); `CTW` is found at `http://www.ele.tue.nl/ctw/download.html` (v0.1); `JBIG2` is found at `http://github.com/agl/jbig2enc` (v0.28).



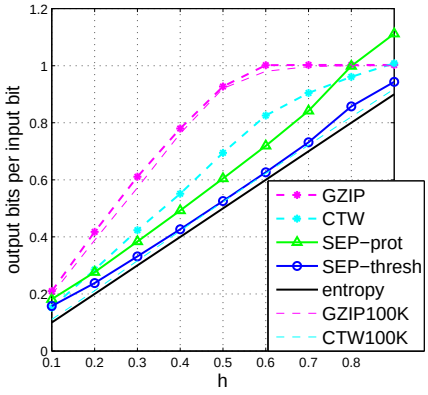
(a) $\mathbf{Bern}(p)$, $n = 10000$, $r_{\text{dope}} = 0$



(b) $\mathbf{Markov}(q)$, $n = 10000$, $r_{\text{dope}} = 0.1$



(c) $\mathbf{Ising}(\frac{1}{2}, q)$, $(h, w) = (100, 100)$, $r_{\text{dope}} = 0.04$



(d) $\mathbf{MarkovZ}[256](h)$, $n = 1000$, $r_{\text{dope}} = 0.04$

Figure 5: Performance of various model-code separation systems.

Ising model with edge potential $\psi_{ij}(s_i, s_j) = q\mathbb{1}_{\{s_i = s_j\}} + (1-q)\mathbb{1}_{\{s_i \neq s_j\}}$ parameterized by q, and a Wiener-process-like Markov source over $\mathbf{Z}_{256}$ with state transition matrix $[q_{(u,v)}]$ parameterized by its entropy h.[2] In Fig. 5d, `GZIP100K` and `CTW100K` are performances when compressing over $n = 100,000$.

Generally `SEP-thresh` has excellent performance. `SEP-prot` is just as good at lower rates while some performance gain is possible at high rates with the choice of a better code; this behavior is also known from channel coding analogs. For Markov sources, `CTW` is able to capture the data structure better than `GZIP` at the same data length, but only reaches performance similar to `SEP-thresh` at longer data lengths. For the 2D Ising model, only `JBIG2` approaches `SEP` performance, while `GZIP` and `CTW` have substantially similar performance, being unable to capture the 2D context well. Part of the reason is that, traditional context-based compressors encode causally, thus they refer to a portion of the data already encoded, thereby losing the performance attributable to the true unrestricted context that `SEP` can access.

[2]This particular case has $q_{(u,v)} \propto \mathbb{P}\left\{|u-v| - \frac{1}{2} < \mathsf{Z} < |u-v| + \frac{1}{2}\right\}$, with $|u-v| \triangleq \min\{|u-v|, |256-(u-v)|\}$ understood cyclically, and Gaussian Z.

6 Conclusion

Although today's compression systems have served us well and given us popular solutions, they are limited by their joint model-code architecture to more restrictive design and usage scenarios. Compression systems going forward need to meet a variety of requirements by making the best use of available technology and data knowledge. We presented a model-code separation architecture and several system constructions that show the ease with which practical compression is possible. More broadly, we provided an open framework within which more participants can think about and experiment with their favorite aspects of compression.

References

[1] A. Aaron, S. D. Rane, E. Setton, and B. Girod, "Transform-domain Wyner-Ziv codec for video," in *Proc. SPIE*, vol. 5308, 2004, pp. 520–528.

[2] M. Johnson, P. Ishwar, V. Prabhakaran, D. Schonberg, and K. Ramchandran, "On compressing encrypted data," *IEEE Trans. Signal Proc.*, vol. 52, no. 10, pp. 2992–3006, Oct. 2004.

[3] D. Schonberg, S. Draper, and K. Ramchandran, "On compression of encrypted images," in *Proc. ICIP*, 2006, pp. 269–272.

[4] S. Jalali, S. Verdu, and T. Weissman, "A universal scheme for Wyner-Ziv coding of discrete sources," *IEEE Trans. Inform. Theory*, vol. 56, no. 4, pp. 1737–1750, Apr. 2010.

[5] D. Donoho, "Compressed sensing," *IEEE Trans. Inform. Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[6] L. He and L. Carin, "Exploiting structure in wavelet-based bayesian compressive sensing," *IEEE Trans. Signal Proc.*, vol. 57, no. 9, pp. 3488–3497, 2009.

[7] G. Caire, S. Shamai, and S. Verdú, "Noiseless data compression with low-density parity-check codes," *DIMACS*, vol. 66, pp. 263–284, 2004.

[8] M. G. Reyes, "Cutset based processing and compression of markov random fields," Ph.D. dissertation, The Univ. of Michigan, 2011.

[9] V. B. Chandar, "Sparse graph codes for compression, sensing, and secrecy," Ph.D. dissertation, MIT, 2010.

[10] M. Wainwright, E. Maneva, and E. Martinian, "Lossy source compression using low-density generator matrix codes: Analysis and algorithms," *IEEE Trans. Inform. Theory*, vol. 56, no. 3, pp. 1351–1368, Mar. 2010.

[11] H. D. Pfister, "On the capacity of finite state channels and the analysis of convolutional accumulate-m codes," Ph.D. dissertation, Univ. of California, San Diego, 2003.

[12] G. Colavolpe, "On LDPC codes over channels with memory," *IEEE Trans. Wireless Commun.*, vol. 5, no. 7, pp. 1757–1766, 2006.

[13] R. Koetter, A. Singer, and M. Tüchler, "Turbo equalization," *IEEE Signal Proc. Magazine*, vol. 21, no. 1, pp. 67–80, Jan. 2004.

[14] S. Kudekar and K. Kasai, "Threshold saturation on channels with memory via spatial coupling," in *Proc. ISIT*, Jul. 2011, pp. 2562–2566.

[15] N. Shulman, "Communication over an unknown channel via common broadcasting," Ph.D. dissertation, Tel-Aviv Univ., 2003.

[16] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge Univ. Press, Mar. 2008.

[17] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory.*, vol. 47, no. 2, pp. 619–637, 2001.

[18] R. Urbanke, "Spatially coupled codes: Good in theory, good in practice," 2011.