TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Nachrichtentechnik

Algorithms for Distribution Matching

Patrick Schulte

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor–Ingenieurs

genehmigten Dissertation.

Vorsitzender:	Prof. Dr.-Ing. Thomas Eibert
Prüfer der Dissertation:	1. Prof. Dr. sc. techn. Gerhard Kramer 2. Prof. Frans M.J. Willems, Ph.D 3. Prof. Erik Agrell, Ph.D.

Die Dissertation wurde am 20.11.2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 10.03.2020 angenommen.

Preface

This thesis is based on work conducted at the Institute for Communications Engineering (LNT) at the Technical University of Munich (TUM) during the past years. I would like to express my gratitude to some people that lead my on this special journey

First of all, I thank Gerhard Kramer for accepting me as a doctoral student. Gerhard impressed me as a scientist with the incredible speed he picks up new ideas and develops them; as an author to break down concepts to its core and to formulate ideas clearly; and as a person with his view and vision of the academic world.

I thank Frans Willems and Erik Agrell to be my second and third examinor, and Thomas Eibert for chairing the examination.

Then, I would like to thank Georg Böcherer who gave me direction already during my Master's and has become a dear friend. Our paths will rejoin soon. With Fabian and Georg, I spent countless memorable moments including listening to the soundtrack "Conquest of Paradise" when preparing for the final of the Bell Labs Prize.

René Essiambre and Gerhard gave me the chance to work at the Bell Labs in Crawford Hill for one summer. I learned a lot about the capacity of the optical fiber, but also about the history of communication, and eventually I could connect names to faces.

The LNT consists of many different characters that all contribute in their way to a great atmosphere. It is this what made our time special.

Finally I would like to thank my family, in particular parents, my sister, and Sophie who supported me during my time as a student.

München, May 2020 Patrick Schulte

Contents

Abstract

Algorithms for distribution matching are considered that approximate discrete memoryless sources by transforming a sequence of coin-flipping bits to a sequence of independent and identically distributed symbols with a desired distribution. Two fixed-to-fixed length code book constructions are studied. The code classes are constant composition codes, i.e., codes that consist only of permutations of one word, and codes that approximate discrete memoryless sources best in terms of informational divergence. An introduction to sequential enumerative coding, divide-and-conquer enumerative coding and arithmetic coding is given. These are well studied source coding techniques that can implement the code classes.

Distribution matchers are a building block of probabilistic amplitude shaping which is a coded modulation technique for bandwidth efficient communication. Distribution matchers in combination with probabilistic amplitude shaping offer a new degree of freedom to adapt the transmission rate and they permit to obtain a shaping gain on top of the coding gain. The shaping gain over the additive white Gaussian noise channel is up to 1.53 dB. Limitations of the shaping gain are shown for block based distribution matchers with finite block lengths and with amplitude shift keying.

Constant composition code books suffer from a rate loss because there is redundancy in the code book. This rate loss is severe for short block lengths and the improvement to conventional transmission schemes is lost. Two ways are presented to improve the performance of probabilistic amplitude shaping with constant composition distribution matching. First, product distribution matching uses multiple binary constant composition distribution matchers in parallel to reduce the rate loss. Two algorithms are proposed to approximate non-product distributions by using multiple distribution matchers. Second, rather than removing redundancy, the decoder can make use of additional code constraints. Low density parity check codes can be made aware of the resulting code word distribution using supplementary check nodes.

Zusammenfassung

Diese Arbeit beschäftigt sich mit Algorithmen zur Verteilungsanpassung, welche gleichverteile Bitsequenzen in Sequenzen unabhängiger und identisch verteilter Symbole mit einer gewünschten Verteilung umwandeln. Eine Untergruppe dieser Algorithmen bliden Sequenzen fester Länge auf Sequenzen fester Länge ab. Es werden zwei Code Konstruktionen dieser Art untersucht. Der erste Code besteht nur aus Permutationen eines Wortes und somit bleibt die empirische Verteilung über einen Block erhalten. Der zweite Code nähert die Zielverteilung im Sinn der Kullback-Leibler-Divergenz am Besten an. Nach einer Einführung in arithmetischer Quellcodierung und Enumerative Coding werden diese zum einkodieren und dekodieren verwendet.

Diese Algorithmen sind ein Voraussetzung für Probabilistic Amplitude Shaping, einer neuen Technik für bandbreiteneffiziente, codierte Modulation. Die Kombination dieser Technologien verfügt über eine neue Stellschraube zur Ratenanpassung und ermöglicht eine Einsparung an Sendeleistung um bis zu 1.53 dB für den additiven Gaußschen Kanal. Es werden Grenzen der möglichen Einsparung für endliche Blocklängen ermittelt unter der Bedingung, dass eine digitale Amplitudenmodulation verwendet wird.

Die Codes, welche nur aus Permutationen bestehen weisen einen Ratenverlust auf und sind aus diesem Grund weniger effizient. Es werden zwei Möglichkeiten präsentiert, mit diesem Ratenverlust umzugehen. Die erste Möglichkeit ist, die Verteilungsanpasser für Produktverteilungen in mehrere separate Verteilungsanpasser aufzuteilen. Hierdurch kann der Verlust stark verringert werden. Für allgemeine Verteilungen werden zwei Algorithmen vorgestellt, die eine Separierung ermöglichen. Bei der zweite Möglichkeit wird die Redundanz am Dekodierer als zusäztliche Einschränkung des Fehlerkorrigierenden Codes verstanden. Der Tanner Graph eines Low-Density-Parity-Check Code wird am Dekodierer mit zusätzlichen Knoten versehen, welche die Verteilung der Symbole prüfen.

1

Introduction

In [1] Shannon defines the channel capacity as the maximum mutual information between a transmitter and receiver pair evaluated over all possible information sources, i.e., the channel input probabilities. Most channels have a non-uniform distribution as a solution to this problem. For instance, the additive white Gaussian noise (AWGN) channel requires a Gaussian code book to approach capacity.

In [2] the authors give a historical background on telephone line modems from the 1950s up to 1984. Coded modulation had just become available in commercial modems. At this time they (re)introduce the concept of non-uniform probabilities on constellations of higher order modulation, and state that "Some sort of mapping [...] must be made into the [...] N-dimensional vectors [...] which have least energy among all such vectors". They are rather pessimistic about the complexity of such an algorithm. Also, the considerations do not include forward error correction (FEC) at this point. We call non-equiprobable signaling *probabilistic shaping*.

With the development of trellis coded modulation (TCM) [3] in 1982 and shell mapping in the 1990s [4–7], there arrived means to implement FEC codes, higher order modulation and probabilistic shaping together as done in the ITU-T V.34 modem standard [8]. However, to support multiple transmission rates one must adapt the FEC code for each rate which complicates chip design. In modern communication applications, the support of multiple rates in one chip is important. This is reflected by the increasing combinations of modulations and FEC codes from DVB-S2 [9] with 53 modulation-coding combinations (modcods) to DVB-S2X [10] with 116 modcods. Each combination defines a transmission rate.

Böcherer [11] modified the Bliss-scheme [12] that was designed for magnetic recording and combined it in a clever way with higher order modulation to implement probabilistic shaping. This scheme requires a so called distribution matcher (DM) which is a non-linear precoder that transforms uniformly distributed bit sequences into symbol

sequences with a non-uniform distribution.

DMs are also known as shaping codes [13] and prefix-free DMs were proposed in [2, Sec. IV.A]. In [6, 14] Huffman codes are used for matching. Optimal variable-to-fixed and fixed-to-variable length DMs are proposed in [15] and [16], respectively. The code books of the matchers in [6, 14–16] must be generated offline and stored, but this is infeasible for the large code word lengths that are needed to achieve the maximum rate. This problem is solved in [17, 18] where arithmetic coding generates code words on the fly. The DMs proposed in [17, 18] are asymptotically optimal. However, all these approaches [6, 14–18] are variable length, which leads to varying transmission rate, large buffer sizes, error propagation and synchronization problems [6, Sec. I]. Block codes (or fixed-to-fixed (f2f) length codes) do not have these issues. Amjad [19, Sec. 4.8] suggests to concatenate short codes and the authors of [20] employ a FEC decoder to build an f2f length matcher. The dematchers, i.e., the inverse operation at the receiver side of [19, 20] cannot always recover the input sequence with zero error. Hence, systematic errors are introduced that cannot be corrected by the error correction code or by retransmission. The thesis [21] proposes adaptive arithmetic distribution matching (AADM) that is an invertible f2f length DM that generates entropy typical sequences, but the algorithm is computationally complex. The above list of algorithms is not complete and there have been many new solutions invented in the past four years.

There is a strong connection between source coding and DM that is reflected by the tools used to implement DMs. In Chapter 2 we review enumerative coding and arithmetic coding that we use later for implementing DM classes. Furthermore, we introduce the problem of energy efficient signaling from the information theoretic point of view and we show how current coded modulation (CM) systems work.

Chapter 3 reviews shaping from the energy perspective and motivates the energy gains that we can expect for higher order modulation. The discussion is based on the geometry of constellations in higher dimensions, i.e., we compare n-D cubes and n-balls. There are also results for finite block lengths and constraints on amplitude shift keying (ASK) constellations. We next review probabilistic amplitude shaping (PAS) as a CM scheme that can achieve those gains. Here, we point out that PAS requires only simple modifications of a CM system described in Chapter 2. We mainly require a systematic FEC encoder and constrain how the code words are mapped to symbols. Apart from that, the only new component is the DM.

The main part of the thesis is Chapters 4 and 5. In Chapter 4 we introduce the optimal DM with respect to informational divergence, which was suggested by Amjad in [19]. We show that the unnormalized divergence scales logarithmically in the block length. This behavior does not harm energy efficient transmission systems. We then connect informational divergence with a per letter weight function and show for which distributions a feasible implementation exists. The optimal code book consists of the sequences of least weight. The proposed algorithmic solutions use enumerative coding and shell mapping to index the optimal code books.

We next consider constant composition distribution matching (CCDM). CCDM uses

permutations of one code word. At first glance, one might expect that this approach cannot compete with the optimal solution in terms of rate. However, we show that CCDM achieves the same rate as the optimal code book asymptotically in the block length. In the algorithmic part we suggest how to index the code book of CCDM with enumerative coding in a sequential and divide-and-conquer manner and with arithmetic coding. The advantage of arithmetic coding is that it works for long blocks where CCDM is competitive. The algorithm was suggested and analyzed for finite arithmetic precision by Ramabadran in [22] for binary codes, and for arbitrary alphabets by Pikus [23].

For short blocks, CCDM exhibits poor performance, especially for large modulation formats. We therefore introduce product distribution matching (PDM) which separates product distributions into multiple DMs of smaller alphabets. For distributions that are not product distributions we provide an optimization algorithm to find product distributions that are close in terms of informational divergence. In [24] Fehenberger reproduces the behavior of one CCDM with large alphabet size using multiple binary CCDMs. A second approach to handle CCDM for short block lengths is presented in Chapter 5.

In [25] a solution for list decoding was presented. List decoders compute a list of code words sorted by probability, followed by supplementary tests to check if the constant composition (CC) constraint is fulfilled. List decoding is a common technique to improve the decoding performance of polar codes combined with cyclic redundancy check (CRC) codes [26]. We may accomplish list decoding for low-density parity-check (LDPC) codes by adding supplementary check nodes to the LDPC code's Tanner graph. This way, we make the decoder aware of the CCDM code book and the decoder can use the redundancy to improve the error rates.

In Chapter 5, we describe an iterative joint decoder for shaping and LDPC codes. We introduce two kinds of CC constraint nodes, i.e., bit-based and symbol-based nodes. Symbol-based CC constraint nodes verify the constraint imposed by the CCDM, while bit-based CC constraint nodes independently check the CC constraint on each shaped bit-level. Symbol-based CC nodes process symbol log-likelihoods (LLs) and therefore require a conversion from log-likelihood ratios (LLRs) to symbol LLs. Simulations show a gain of about 0.5 dB over the PAS decoder [11] for length 192 5G LDPC codes [27] at a spectral efficiency of 1.5 bit per channel use.

2

Preliminaries

2.1 General Definitions

We write matrices in uppercase bold letters $\boldsymbol{X}$, sets in caligraphic letters $\mathcal{X}$, random variables with uppercase sans-serif letters X, and their realizations with lowercase letters x.

The cardinality of a set $\mathcal{X}$, i.e., the number of its elements is written as $|\mathcal{X}|$, and the *Cartesian product* of two sets $\mathcal{X}$, $\mathcal{Y}$ is

$$\mathcal{X} \times \mathcal{Y} = \{(x, y) | x \in \mathcal{X}, y \in \mathcal{Y}\}. \tag{2.1}$$

The *n-fold product set* $\mathcal{X}^n$ is

$$\mathcal{X}^n = \underbrace{\mathcal{X} \times \mathcal{X} \times \cdots \times \mathcal{X}}_{n-\text{times}}. \tag{2.2}$$

Let X be a discrete random variable with probability mass function (pmf) P_{X} defined on a set $\mathcal{X}$, i.e., we have

$$P_{\mathsf{X}}(a) \geq 0 \quad \forall a \in \mathcal{X} \quad \text{and} \quad \sum_{a \in \mathcal{X}} P_{\mathsf{X}}(a) = 1. \tag{2.3}$$

The *support* of a pmf is the subset of the sample space for which the pmf is positive:

$$\text{supp}(P_{\mathsf{X}}) := \{x | x \in \mathcal{X}, P_{\mathsf{X}}(x) > 0\}. \tag{2.4}$$

Let P_{XY} be a *joint distribution* of two random variables on a set $\mathcal{X} \times \mathcal{Y}$, i.e., we have

$$P_{\mathsf{XY}}(a,b) \geq 0 \quad \forall (a,b) \in \mathcal{X} \times \mathcal{Y} \quad \text{and} \quad \sum_{(a,b)\in\mathcal{X}\times\mathcal{Y}} P_{\mathsf{XY}}(a,b) = 1. \tag{2.5}$$

The *marginal distribution* is

$$P_{\mathsf{X}}(a) = \sum_{b\in\mathcal{Y}} P_{\mathsf{XY}}(a,b). \tag{2.6}$$

The *conditional distribution* is

$$P_{\mathsf{X}|\mathsf{Y}}(a|b) = \frac{P_{\mathsf{XY}}(a,b)}{P_{\mathsf{Y}}(b)} \tag{2.7}$$

if

$$P_{\mathsf{Y}}(b) > 0. \tag{2.8}$$

We call two random variables *independent* if their joint distribution is the product distribution

$$P_{\mathsf{XY}}(a,b) = P_{\mathsf{X}}(a) \cdot P_{\mathsf{Y}}(b) \quad \forall a,b \in \mathcal{X},\mathcal{Y}. \tag{2.9}$$

Independence implies that the conditional distribution simplifies to the marginal distribution

$$P_{\mathsf{X}|\mathsf{Y}}(a|b) = \frac{P_{\mathsf{XY}}(a,b)}{P_{\mathsf{Y}}(b)} = \frac{P_{\mathsf{X}}(a) \cdot P_{\mathsf{Y}}(b)}{P_{\mathsf{Y}}(b)} = P_{\mathsf{X}}(a). \tag{2.10}$$

The *expectation value* of a real-valued function defined on $\mathcal{X}$ is defined as

$$\mathbb{E}\left[f(\mathsf{X})\right] = \sum_{a\in\operatorname{supp} P_{\mathsf{X}}} P_{\mathsf{X}}(a) \cdot f\left(a\right). \tag{2.11}$$

The *mean* or *first moment* is

$$\mu = \mathbb{E}\left[\mathsf{X}\right]. \tag{2.12}$$

The *empirical average* of n realizations of x_i is defined as

$$\bar{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i. \tag{2.13}$$

The weak law of large numbers states that

$$\lim_{n\to\infty} \bar{x}_n = \mu \tag{2.14}$$

if the x_i, $i = 1,\ldots,n$, are independent realizations of X. The *variance* (or second central moment) is defined as the expected squared deviation from the mean

$$\operatorname{Var}\left[\mathsf{X}\right] = \mathbb{E}\left[(\mathsf{X}-\mu)^2\right] = \mathbb{E}\left[\mathsf{X}^2\right] - \mu^2. \tag{2.15}$$

We call a discrete random variable X *uniformly* distributed if

$$P_{\mathsf{X}}(a) = \frac{1}{|\mathcal{X}|} \quad \forall a \in \mathcal{X}. \tag{2.16}$$

The *Bernoulli* distributed random variable $\mathsf{X} \sim$Bernoulli(p) has a discrete distribution with

$$\Pr(\mathsf{X} = 0) = p = 1 - \Pr(\mathsf{X} = 1). \tag{2.17}$$

A Bernoulli$(1/2)$ distributed binary random variable is uniformly distributed.

We denote an n-dimensional vector of random variables as $\mathsf{A}^n = \mathsf{A}_1\mathsf{A}_2 \dots \mathsf{A}_n$ with realization $a^n = a_1 a_2 \dots a_n$. If the dimension of a vector(or length of a *string*, or length of a *finite sequence*) is known or not important in the context we write it as a bold letter $\boldsymbol{a}$. The concatenation of two vectors or sequences a^n, b^k is written as $[a^n, b^k]$. A subsequence from the i-th element of the sequence a^n to the j-th element of a^n is written as a_i^j.

2.2 Information Theory

An *event* is a subset of the sample space. We write events $\{x\}$ where x is an element of the sample space simply as x. If an event x occurs with positive probability, then its *self-information* is defined as

$$\iota\left(P_{\mathsf{X}}(x)\right) = -\log_2(P_{\mathsf{X}}(x)) \tag{2.18}$$

measured in bits. As probability takes on values between 0 and 1, self-information is always non-negative.

The *entropy* of a random variable X is the expectation of the self-information of X, i.e.,

$$\mathbb{H}\left(\mathsf{X}\right) = \mathbb{E}\left[\iota\left(P_{\mathsf{X}}(\mathsf{X})\right)\right] = \mathbb{E}\left[-\log_2\left(P_{\mathsf{X}}(X)\right)\right] = \sum_{x \in \mathrm{supp}(P_{\mathsf{X}})} -P_{\mathsf{X}}(x)\log_2\left(P_{\mathsf{X}}(x)\right) \tag{2.19}$$

where $\mathrm{supp}(P_{\mathsf{X}}) \subseteq \mathcal{X}$ is the support of P_{X}, i.e., the subset of x in $\mathcal{X}$ with positive probability. With slight abuse of notation we also write

$$\mathbb{H}\left(P_{\mathsf{X}}\right) = \mathbb{H}\left(\mathsf{X}\right) \tag{2.20}$$

and $\mathbb{H}\left(p\right)$ for binary entropies.

As self-information is non-negative, the entropy is also non-negative. We can bound entropy by

$$0 \overset{(i)}{\leq} \mathbb{H}\left(\mathsf{X}\right) \overset{(ii)}{\leq} \log_2\left(|\mathcal{X}|\right) \tag{2.21}$$

with equality in (i) if and only if $P_{\mathsf{X}}(a) = 1$ for some $a \in \mathcal{X}$ and in (ii) if and only if X is uniformly distributed on $\mathcal{X}$. Entropy is a measure for uncertainty about the outcome before performing the random experiment.

A source with memory generates symbol sequences $x_1, x_2, \ldots, x_n$ that are not independent of each other. Any joint distribution P_{X^n} can be written as

$$P_{\mathsf{X}^n}(x^n) = \prod_{i=1}^{n} P_{\mathsf{X_i}|\mathsf{X}^{i-1}}(x_i|x^{i-1}) \tag{2.22}$$

after applying (2.7) n-times.

The marginal distribution of the i-th symbol is denoted by P_{X_i}. The *letter distribution* $P_{\bar{\mathsf{X}}}$ of a length n block is defined as

$$P_{\bar{\mathsf{X}}}(a) = \frac{1}{n} \sum_{i=1}^{n} P_{\mathsf{X}_i}(a). \tag{2.23}$$

A *discrete memoryless source* (DMS) generates identically distributed symbols that are independent. For a DMS, equation (2.22) simplifies with (2.10) to

$$P_{\mathsf{X}^n}(x^n) = \prod_{i=1}^{n} P_{\mathsf{X}}(x_i). \tag{2.24}$$

To emphasize the joint independence of the X_i we define

$$P_{\mathsf{X}}^n(x^n) := \prod_{i=1}^{n} P_{\mathsf{X}}(x_i). \tag{2.25}$$

The *informational divergence*, also known as the *Kullback-Leibler divergence* of two distributions P_{X} and P_{Y} on $\mathcal{X}$ is defined as

$$\mathbb{D}\left(P_{\mathsf{X}} \| P_{\mathsf{Y}}\right) = \mathbb{E}\left[\log_2\left(\frac{P_{\mathsf{X}}(\mathsf{X})}{P_{\mathsf{Y}}(\mathsf{X})}\right)\right] = \sum_{a \in \mathrm{supp}(P_{\mathsf{X}})} P_{\mathsf{X}}(a) \log_2\left(\frac{P_{\mathsf{X}}(a)}{P_{\mathsf{Y}}(a)}\right). \tag{2.26}$$

We refer to it simply as divergence. We have

$$\mathbb{D}\left(P_{\mathsf{X}} \| P_{\mathsf{Y}}\right) \geq 0 \tag{2.27}$$

with equality if and only if $P_{\mathsf{X}} = P_{\mathsf{Y}}$. Note that $\mathbb{D}\left(P_{\mathsf{X}} \| P_{\mathsf{Y}}\right) \neq \mathbb{D}\left(P_{\mathsf{Y}} \| P_{\mathsf{X}}\right)$ in general and that $\log_2\left(P_{\mathsf{X}}(a)/P_{\mathsf{Y}}(a)\right)$ might be negative for some a. For binary probability distributions $P_{\mathsf{X}} = [p, 1-p]$ and $P_{\mathsf{Y}} = [q, 1-q]$ we may write

$$\mathbb{D}\left(p \| q\right) := \mathbb{D}\left(P_{\mathsf{X}} \| P_{\mathsf{Y}}\right). \tag{2.28}$$

If X_1 and X_2 are independent random variables, and so are Y_1 and Y_2, then we have

$$\mathbb{D}\left(P_{\mathsf{X}_1\mathsf{X}_2}\|P_{\mathsf{Y}_1\mathsf{Y}_2}\right) = \mathbb{D}\left(P_{\mathsf{X}_1}\|P_{\mathsf{Y}_1}\right) + \mathbb{D}\left(P_{\mathsf{X}_2}\|P_{\mathsf{Y}_2}\right). \tag{2.29}$$

The *mutual information* (MI) of two random variables X and Y with joint pmf P_{XY} is

$$\mathbb{I}\left(\mathsf{X};\mathsf{Y}\right) = \mathbb{D}\left(P_{\mathsf{XY}}\|P_{\mathsf{X}} \times P_{\mathsf{Y}}\right) \tag{2.30}$$

where

$$(P_{\mathsf{X}} \times P_{\mathsf{Y}})(xy) := P_{\mathsf{X}}(x) \cdot P_{\mathsf{Y}}(y). \tag{2.31}$$

We may also define mutual information via entropies, i.e., we have

$$\mathbb{I}\left(\mathsf{X};\mathsf{Y}\right) = \mathbb{H}\left(\mathsf{X}\right) - \mathbb{H}\left(\mathsf{X}|\mathsf{Y}\right) = \mathbb{H}\left(\mathsf{Y}\right) - \mathbb{H}\left(\mathsf{Y}|\mathsf{X}\right). \tag{2.32}$$

The *normalized informational divergence* is greater than or equal to the corresponding divergence of the letter distribution because of the convexity of divergence [28, Theorem 2.7.2]

$$\frac{\mathbb{D}\left(P_{\mathsf{X}^n}\|P_{\mathsf{Y}}^n\right)}{n} \geq \mathbb{D}\left(P_{\bar{\mathsf{X}}}\|P_{\mathsf{Y}}\right). \tag{2.33}$$

Lemma 2.1 (Bound on Entropy Difference)**.** Let $0 < p < 1$ and $0 < p - \epsilon < p$. Then

$$\mathbb{H}\left(p\right) - \mathbb{H}\left(p-\epsilon\right) \leq \epsilon \log_2 \frac{1-p+\epsilon}{p-\epsilon}. \tag{2.34}$$

Proof: The bound follows from

$$\mathbb{H}\left(p\right) - \mathbb{H}\left(p-\epsilon\right) = \epsilon \log_2 \frac{1-p+\epsilon}{p-\epsilon} - \mathbb{D}\left(p\|p-\epsilon\right) \tag{2.35}$$

and the non-negativity of divergence. ■

2.3 Counting

The empirical distribution $P_{\bar{\mathsf{A}},a^n}(\alpha)$ of the sequence $a^n \in \mathcal{A}^n$ is

$$P_{\bar{\mathsf{A}},a^n}(\alpha) := \frac{n_\alpha(a^n)}{n} \tag{2.36}$$

where $n_\alpha(a^n) = |\{i : a_i = \alpha\}|$ is the number of times the symbol $\alpha \in \mathcal{A}$ appears in a^n. We call $n - n_0(a^n)$ the *weight* of a^n. The authors of [29, Sec. 2.1] call $P_{\hat{\mathsf{A}},a^n}(\cdot)$ the *type* of a^n. An n-type is a type based on a length-n sequence. Note that the n-types partition the alphabet $\mathcal{A}^n$ into equivalence classes, called the *type sets*. We denote the type set that contains all sequences of length n with empirical distribution P_{A} by $\mathcal{T}^n_{P_{\mathsf{A}}}$.

For binary sequences of length n, the cardinality of a type set $\mathcal{T}^n_{P_\mathsf{A}}$ with $P_\mathsf{A}(0) = n_0/n$ and $P_\mathsf{A}(1) = n_1/n$ is

$$\left|\mathcal{T}^n_{P_\mathsf{A}}\right| = \binom{n}{n_0} = \binom{n}{n_1} = \frac{n!}{n_0! \cdot n_1!}. \tag{2.37}$$

For a q-ary alphabet $\mathcal{A} = \{0, 1, \ldots, q-1\}$ we obtain the multinomial expression

$$\left|\mathcal{T}^n_{P_\mathsf{A}}\right| = \binom{n}{n_0, n_1, \ldots, n_{q-1}} = \frac{n!}{n_0! \cdot n_1! \cdot \ldots \cdot n_{q-1}!}. \tag{2.38}$$

Lemma 2.2 (Bounds on binomial coefficients)**.** If $0 < p < 1$ and np is integer, then

$$\frac{2^{n\mathbb{H}(p)}}{\sqrt{8np(1-p)}} \leq \binom{n}{np} \leq \frac{2^{n\mathbb{H}(p)}}{\sqrt{2\pi np(1-p)}}. \tag{2.39}$$

Proof: The proof follows by applying Stirling's approximation twice, see [30, Ch. 4.7]. ■

Lemma 2.3 (Bounds on multinomial coefficients)**.** Let $n_1, n_2, \ldots, n_q$ be positive integers and $n = \sum_{i=1}^q n_i$. Then $P_\mathsf{A}(i) = p_i = \frac{n_i}{n}\ \forall i \in \{1, \ldots, q\}$ is a pmf and we have

$$\frac{2^{n\mathbb{H}(P_\mathsf{A})}}{\sqrt{\frac{8^{q-1}}{n}\prod_{i=1}^q n_i}} \leq \binom{n}{n_1, n_2, \ldots, n_q} \leq \frac{2^{n\mathbb{H}(P_\mathsf{A})}}{\sqrt{\frac{(2\pi)^{q-1}}{n}\prod_{i=1}^q n_i}}. \tag{2.40}$$

Proof: The proof follows by splitting the multinomial into binomials by recursively applying

$$\binom{n}{n_1, n_2, \ldots, n_q} = \frac{n!}{n_1! \ldots n_q!} = \frac{n!}{n_1!(n-n_1)!}\frac{(n-n_1)!}{n_2! \ldots n_q!} = \binom{n}{n_1}\binom{n-n_1}{n_2, \ldots, n_q} \tag{2.41}$$

and then using Lemma 2.2 for each binomial. The *grouping rule for entropy* [28, Problem 2.27]

$$\mathbb{H}\left([p_1, p_2, \ldots, p_q]\right) = \mathbb{H}\left([p_1, 1-p_1]\right) + (1-p_1)\mathbb{H}\left([p_2, \ldots, p_q]\right) \tag{2.42}$$

relates the binary entropies. ■

Lemma 2.4 (Bounds on partial sums of binomial coefficients [31])**.** If $0 \leq p < 1/2$ and np is integer, then

$$\binom{n}{np}\alpha\beta \leq \sum_{i=0}^{np}\binom{n}{i} \leq \binom{n}{np}\alpha \tag{2.43}$$

with

$$\alpha = \frac{1-p+1/n}{1-2p+1/n} \tag{2.44}$$

and

$$\beta = \frac{n(1-2p)^2}{1+n(1-2p)^2}. \tag{2.45}$$

Lemma 2.5 (Sum of binomial coefficients weighted with distance from the center [32, p. 166])**.** For every positive integer n and non-negative integer k, $k \leq n$, we have

$$\sum_{i=0}^{k} \binom{n}{i} \left(\frac{n}{2} - i\right) = \frac{k+1}{2} \binom{n}{k+1}.$$

2.4 Enumerative Coding

Enumerative coding was introduced as a general encoding scheme by Cover [33]. The technique was implicitly used by Schalkwijk [34] and Lynch [35] for special cases. The idea of enumerative coding is to enumerate sequences a^n in any subset $\mathcal{C}$ of $\mathcal{A}^n$ in lexicographical order, i.e., a sequence a^n is ranked before a sequence b^n if the first letter that differs comes first in some ordering than the respective letter of the sequence b^n. An ordering could be the numeric value or the order of the alphabet. In the original paper [33], permutations of one word and binary sequences up to a certain weight are encoded using enumerative coding. Lossless compression of a DMS with using enumerative coding is a challenge because a lossless block code for a DMS implies that $\mathcal{C} = \mathcal{A}^n$ where $\mathcal{A} = \operatorname{supp} P_A$. However, we can add a prefix that tells the sequence type, after which follows the index. Thus, we use a variable length code because the index has variable length. This idea is related to universal codes. Enumerative coding can be used for lossy source coding, i.e., if the sequence is an element of $\mathcal{C}$ then we transmit its index. If the sequence is not an element of $\mathcal{C}$ then we transmit a supplementary error message.

Consider the toy example that a source emits only binary sequences of length 8 and these sequences consist of exactly one "1" and seven "0"s. Enumerative coding suggests to transmit the index where the "1" occurs instead of transmitting the entire sequence. The receiver can map the index back to the original sequence.

2.4.1 Decoding

The core of enumerative coding is knowing how many sequences a^n there are in the code book $\mathcal{C}$ with an arbitrary prefix b^k, i.e., how many sequences $a^n \in \mathcal{C}$ start with the letters b^k. We overload the counting notation by defining

$$n_{b^k}(\mathcal{C}) = |\{a^n \in \mathcal{C} : a_1^k = b^k\}|. \tag{2.46}$$

Throughout this thesis we use encoding and decoding to mean that we encode an index N to the respective code word a^n of a code book $\mathcal{C}$ and decode a code word to the respective index. In the *decoding* step we count how many sequences there are that

come first in lexicographical order. Given that we are aware of all $n_{b^k}(\mathcal{C})$, the index N of a sequence a^n is given by

$$N = \sum_{k=1}^{n} \sum_{\alpha<a_k} n_{[a^{k-1},\alpha]}(\mathcal{C}). \tag{2.47}$$

In (2.47) we start with the first letter and proceed to the last letter. The first outer summand calculates how many sequences start with a smaller letter than a_1. Those sequences come first in order. The second summand collects all sequences that start with a_1 and continue with a letter that is lower in order than a_2. We continue in this way for all indices. Note that we assign 0 to the first sequence in order, because we count how many sequences are below.

Example 2.1 (decoding 5-ary digits)**.** In this example, we use enumerative coding to convert a length 4 sequence of 5-ary digits, i.e., $\mathcal{A} = \{0, 1, 2, 3, 4\}$, into its integer value (base 10). We first need to identify how many sequences start with a suffix b^k. Fortunately, in this case n_{b^k} depends only on the length of the suffix k. For example, a length 3 suffix implies that we can still choose one digit freely, i.e., there are 5 possibilities. There are always 25 possibilities for a length 2 suffix.

For the sequence $[1, 3, 0, 0]$ there are four outer summands in (2.47). The first evaluates to 125, since there are 5^3 sequences of length 4 that start with a 0. The second one is $n_{[1,0]} + n_{[1,1]} + n_{[1,2]} = 75$. The last two summands are zero because there is no letter in the order below 0. Hence, the sequence $[1, 3, 0, 0]$ is mapped to the index $N = 200$.

2.4.2 Encoding

For *encoding*, i.e., finding the sequence a^n in the N-th position of the lexicographically ordered list, we use Algorithm 2.1.

Algorithm 2.1 Enumerative Coding: Encoder

Require: N, $n_{\boldsymbol{b}}(\mathcal{C})$ $\quad \forall$ Prefixes $\boldsymbol{b}$
  find a_1 such that:
    $\sum_{\alpha<a_1} n_{[\alpha]}(\mathcal{C}) \leq N < \sum_{\alpha\leq a_1} n_{[\alpha]}(\mathcal{C})$
  $N = N - \sum_{\alpha<a_1} n_{[\alpha]}(\mathcal{C})$
  for $i = 2, 3, \ldots, n$ **do**
    find a_j such that:
      $\sum_{\alpha<a_i} n_{[\boldsymbol{a}_1^{i-1},\alpha]}(\mathcal{C}) \leq N < \sum_{\alpha\leq a_i} n_{[a^{i-1},\alpha]}(\mathcal{C})$
    $N = N - \sum_{\alpha<a_j} n_{[\boldsymbol{a}_1^{j-1},\alpha]}(\mathcal{C})$.
  end for
  return a^n

Example 2.2 (encoding to 5-ary digits)**.** We want to encode the index 200. Consider the first step. n_α is always 125, i.e., for a_1 we fulfill the requirement for the letter "1" since $125 \leq 200 < 250$. After the update, N equals 75. For $k = 2$, the n_α is always 25 and for $a_2 = 3$ we have $75 \leq 75 < 100$, and the new N is 0. The remaining indices are both 0 since $N = 0$. Hence, the index 200 is mapped to the sequence $[1, 3, 0, 0]$

The above examples are very simple because n_{b^k} depends on k only. For more complicated sets we must either store the n_{b^k} or compute them on the fly.

2.5 Divide-and-Conquer Enumerative Coding

For some indexing problems one can find a divide-and-conquer strategy that requires less memory resources but can be computationally more complex than the sequential approach. As the name suggests, we divide the problem into two easier problems that will again be subdivided until the solution is easy, and we then back-compute the solution. Shell mapping (SM) is an example of divide-and-conquer enumerative coding that we discuss in Sec. 4.2.6. At this point we do not describe general encoding and decoding algorithms but rather instantiate them for concrete problems later. We again consider the example of finding the index of a 5-ary sequence to show the benefits of a divide and conquer approach.

Example 2.3 (decoding 5-ary digits with divide and conquer principle)**.** Consider again the sequence $[1, 3, 0, 0]$. We divide the sequence into two sequences $[1, 3]$ and $[0, 0]$ and ask for their position in lexicographical ordering. With a supplementary dividing step we obtain [1], [3], [0], [0], with the obvious indices 1, 3, 0 and 0. From this point on, we move back in the tree and calculate the index of the combined sequence. The index of $[\boldsymbol{x}, \boldsymbol{y}]$ is 5 times the index of the first sequence $[\boldsymbol{x}]$ plus the index of the second sequence $[\boldsymbol{y}]$. We obtain 8 and 0 for the problems $[1, 3]$ and $[0, 0]$, respectively. In the last step we combine the solutions again; this time we multiply the solution of the left-hand side with 25, since the prefix now has length 2. In the end we obtain 200 again. We summarize the decoding process in Fig. 2.1.

In enumerative coding we required $n_{b^k}(\mathcal{C})$ of all prefixes b^k. Now, we must know only the prefixes where k is a power of 2. However, for some problems we are restricted to block lengths that are powers of two to make use of this advantage.

2.6 Arithmetic Coding

Arithmetic coding is a source coding technique and in [36, Chap. 1.2] we find a brief history of its main ideas that date back to the early days of information theory [1, Sec. 8]. Despite its early development, arithmetic coding became famous in the 1970s when the algorithm was formulated with finite precision arithmetic [37, 38].

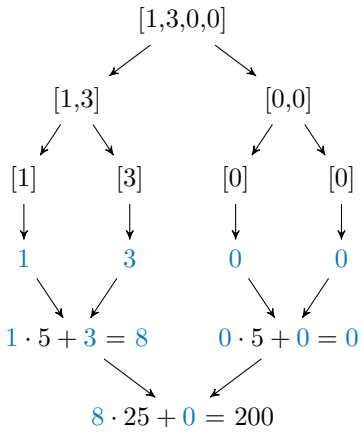


Figure 2.1: Illustration of a divide-and-conquer approach. The algorithm finds the index of the sequence $[1, 3, 0, 0]$ in the lexicographical ordered list.

Arithmetic coding divides the interval $[0, 1)$ into subintervals according to the cumulative mass function (cmf) of the source model. $[0, 1)$ is a half-open interval, which means that it contains all points greater than or equal to 0 and less than 1. The source symbols are read sequentially and the interval is updated after each source symbol. After several refinement steps we obtain an interval $[s_l, s_u)$ and we look for the shortest binary fixed point number $c = 0.b_1b_2 \dots b_k$ such that $[c, c+2^{-k}) \subseteq [s_l, s_u)$. We call $[c, c+2^{-k})$ the code interval. The bits $b_1, \dots b_k$ are then transmitted. Note that for larger source intervals it is easier to find a large code interval that can be represented by few bits $b_1, \dots, b_k$. On the decoder side we are aware of $b_1, \dots, b_k$ and try to find the interval $[s_l, s_u)$.

Similar to enumerative coding, we are interested in inverting the source coding problem, i.e., we start with Bernoulli(1/2) distributed bits and try to map them to a sequence of symbols with a non-uniform distribution. In the following, we describe the methods developed in [18, 21, 39, 40].

2.6.1 Encoding

For encoding and decoding we use the following terminology.

- ▷ There are *source intervals* and *code intervals* that represent probabilities on the source side and on the code side, respectively.
- ▷ When we *refine* an interval, this means that we partition an interval according to a cmf and select one of those partitions as a new interval.
- ▷ The sequence of symbols that we use to refine an interval is the *assigned sequence*.
- ▷ An interval $[a, b)$ *identifies* an interval $[c, d)$ if $[a, b) \subseteq [c, d)$.

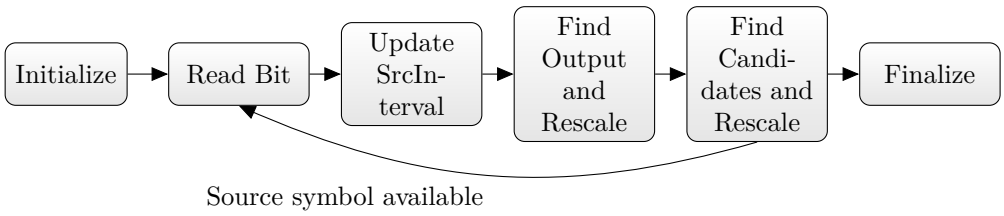


Figure 2.2: On-the-fly arithmetic encoding algorithm.

▷ We call a code interval $[c, d)$ a *code candidate* if it overlaps with the source interval $[a, b)$, i.e., $[a, b) \cap [c, d)$ is not empty.

The arithmetic encoder consists of an initialization step, a loop and an finalization step. The initialization provides a source and code interval that both span from 0 to 1. In the loop the algorithm first reads a new bit and then refines the source interval. In case the source interval identifies a code interval, the assigned code symbols are a prefix of the code word. When we refine an interval, all partitions remain inside the original interval. Therefore every refinement of the source interval remains within the source interval and consequently remains in the code interval. This means that sequences assigned to a code word candidate start with the same prefix and this common prefix grows during the process of encoding. Consequently, this *sure prefix* may be sent or written to an output buffer. We are no longer interested in the other code sequences that are not a prefix of the code and scale the intervals such that the borders of the sure prefix now correspond to 0 and 1. Let l be the lower border and u the upper border, then the linear scaling is defined as

$$x \to \hat{x} = \frac{x - l}{u - l}. \tag{2.48}$$

In case the source interval does not identify a code interval, there is no output and no scaling.

There are source code sequences that lead to source intervals that do not identify a code symbol but toggle around a border. We therefore introduce a further method that guarantees scaling after a certain number of new input bits. We call the method 'Check for Lower Level Candidates and Rescale' in Fig. 2.2 and illustrate it in Fig. 2.3.

Consider two neighboring code candidates $[l_1; m_1)$ and $[m_1; u_1)$ and a source interval $[s_l; s_u)$ with

$$[s_l; s_u) \subseteq [l_1; u_1) \tag{2.49}$$
$$[s_l; s_u) \nsubseteq [m_1; u_1) \tag{2.50}$$
$$[s_l; s_u) \nsubseteq [l_1; m_1). \tag{2.51}$$

Consider the q-ary target distribution P_X with $\mathcal{X} = \{0, 1, \ldots q-1\}$. Then, $P_X(0)$ and $P_X(q-1)$ are the probabilities of the first symbol and the q-th symbol, respectively. We

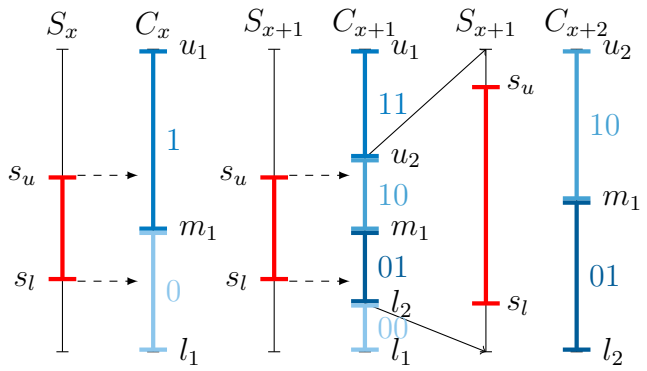


Figure 2.3: Toggling the input if lower level subintervals are promising.

calculate the borders of the next level for the toggle case as

$$u_2 = m_1 + P(0) \cdot (u_1 - m_1)$$
$$l_2 = m_1 - P(q-1) \cdot (m_1 - l_1).$$

We allow a scaling to

$$[l_1, u_2) \quad \text{if } [s_l; s_u) \subseteq [l_1; u_2) \tag{2.52}$$
$$[l_2, u_1) \quad \text{if } [s_l; s_u) \subseteq [l_2; u_1) \tag{2.53}$$
$$[l_2, u_2) \quad \text{if } [s_l; s_u) \subseteq [l_2; u_2) \tag{2.54}$$

where (2.54) is a combination of (2.52) and (2.53). This way we guarantee scaling after a certain number of input symbols.

The finalization step makes sure that the code interval identifies a source interval, i.e., the code interval is a subset of the source interval and therefore shorter. This way the decoder can retrieve also the last bits. The size of an interval represents the probability of its assigned sequence. Due to the finalization step we know that $P_{\mathsf{X}}^n(a^n)$ must be smaller than 2^{-k}, where k is the number of read input bits. Note that k varies. We implicitly assume that those bits are Bernoulli(1/2) distributed and the self-information of the source sequence is $-\log_2(2^{-k}) = k$. Let a^n be the output of the arithmetic encoder. We have

$$2^{-k} \geq P_{\mathsf{X}}^n(a^n) \tag{2.55}$$
$$\Rightarrow k \leq \iota\left(P_{\mathsf{X}}^n(a^n)\right). \tag{2.56}$$

Baur and Böcherer bounded the ratio of source interval size to code interval size in [18, Proposition 1] .

Lemma 2.6. The ratio of the interval sizes is bounded as

$$1 \leq \frac{2^{-k}}{P_{\mathsf{X}}^n(a^n)} \leq \frac{1}{P_{\mathsf{X}}(0) \cdot P_{\mathsf{X}}(1)}. \tag{2.57}$$

where a^n is the output of the DM. We can easily generalize the bound to non-binary pmfs.

Lemma 2.7. The ratio of source interval size to code interval size for non-binary target distributions is bounded as

$$1 \leq \frac{2^{-k}}{P_{\mathsf{X}}^n(a^n)} \leq \left(\min_{\alpha \in \operatorname{supp}(P_{\mathsf{X}})} P_{\mathsf{X}}(\alpha) \cdot \max_{\alpha \in \operatorname{supp}(P_{\mathsf{X}})} P_{\mathsf{X}}(\alpha) \right)^{-1}. \tag{2.58}$$

The proof follows the same steps as in [18, Proposition 1]. Furthermore, we can insert (2.57) into the definition of divergence and obtain for the binary case

$$\mathbb{D}\left(P_{\mathsf{Z}} \| P_{\mathsf{X}}^n\right) \leq \log_2 \left(\frac{1}{P_{\mathsf{X}}(0) \cdot P_{\mathsf{X}}(1)} \right) \tag{2.59}$$

and for the non-binary case

$$\mathbb{D}\left(P_{\mathsf{Z}} \| P_{\mathsf{X}}^n\right) \leq -\log_2 \left(\min_{\alpha \in \operatorname{supp}(P_{\mathsf{X}})} P_{\mathsf{X}}(\alpha) \cdot \max_{\alpha \in \operatorname{supp}(P_{\mathsf{X}})} P_{\mathsf{X}}(\alpha) \right). \tag{2.60}$$

The random variable Z captures the dyadic distribution that stems from the varying number of input bits.

Example 2.4 (arithmetic encoding algorithm)**.** Let the input bits be Bernoulli(1/2) distributed and let the target distribution be $P_{\mathsf{X}}(0) = 1 - P_{\mathsf{X}}(1) = 0.4$. Consider encoding the binary sequence '10'. The initialization step provides two intervals. The source interval is $[0, 1)$ and code interval is split into two subintervals $[0, 0.4)$ and $[0.4, 1)$. In the first execution of the loop we read the first bit, which is '1'. This bit corresponds to the upper half of the source interval. Consequently, the source interval will be refined to $[0.5, 1)$. We now check for a possible output symbol. $[0.5, 1)$ clearly is a subinterval of $[0.4, 1)$, so we are sure that the first code symbol must be 1. We can thus output the first symbol. We call this property *online* encoding. As the other candidate is not interesting anymore, we scale according to (2.48). The borders of the code interval assigned to '1' map to 0 and 1 again. The resulting intervals are $[\frac{1}{6}, 1)$ and $[0, 1)$ for the source interval and code interval, respectively. Conditions (2.52) to (2.54) are not fulfilled, so there is no lower level scaling.

The second loop starts. The second bit is 0, so the source interval refines to $[\frac{1}{6}, \frac{7}{12})$ while the code interval is again subdivided into $[0, 0.4)$ and $[0.4, 1)$.

This source interval does not identify a new code symbol yet, because it is neither a subinterval of $[0, 0.4)$ nor of $[0.4, 1)$. We have read the whole input sequence and

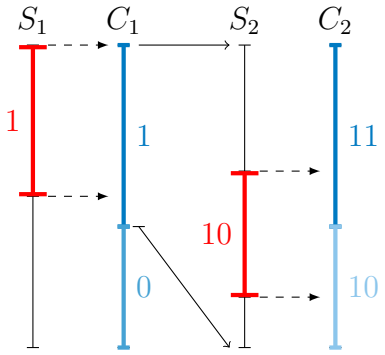


Figure 2.4: Encoding '10' with target distribution $P_Y \sim$ Bernoulli(0.4).

need to finalize the code word. The code word needs to identify the source interval $[\frac{1}{6}, \frac{7}{12})$. Otherwise the source sequence cannot be fully decoded. We therefore start the finalization step. Fig. 2.5 illustrates this example.

We refine the code intervals according to the target distributions and obtain new borders at 0.16 and 0.64. There is still no clear identification, but we obtain two candidates '110' and '101' that remain interesting. '111' and '100' and descendents can not be chosen anymore. In this example we perform a linear scaling such that these two remaining candidates range from 0 to 1. We refine and there are two sequences '1100' and '1011' identifying the source interval. As '1011' is the larger interval, and therefore more probable, we choose this output.

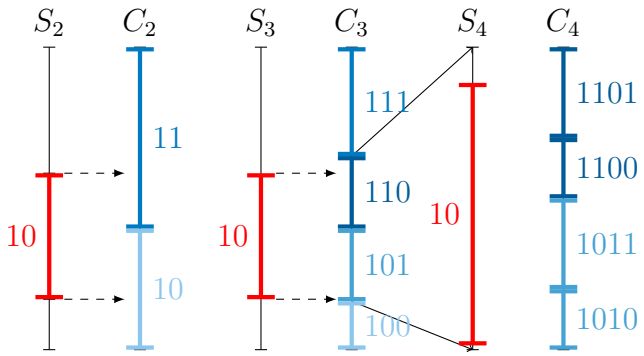


Figure 2.5: Finalizing '10' with target distribution $P_Y \sim$ Bernoulli(0.4) .

2.6.2 Decoding

The general mechanism of the decoder emulates the encoder, see Fig. 2.6. For encoding before the finalization step we tried to identify code intervals by refining a source interval. We now do the opposite. The decoder receives code symbols, and each code symbol refines the code interval. We know that the source interval defined by the source sequence

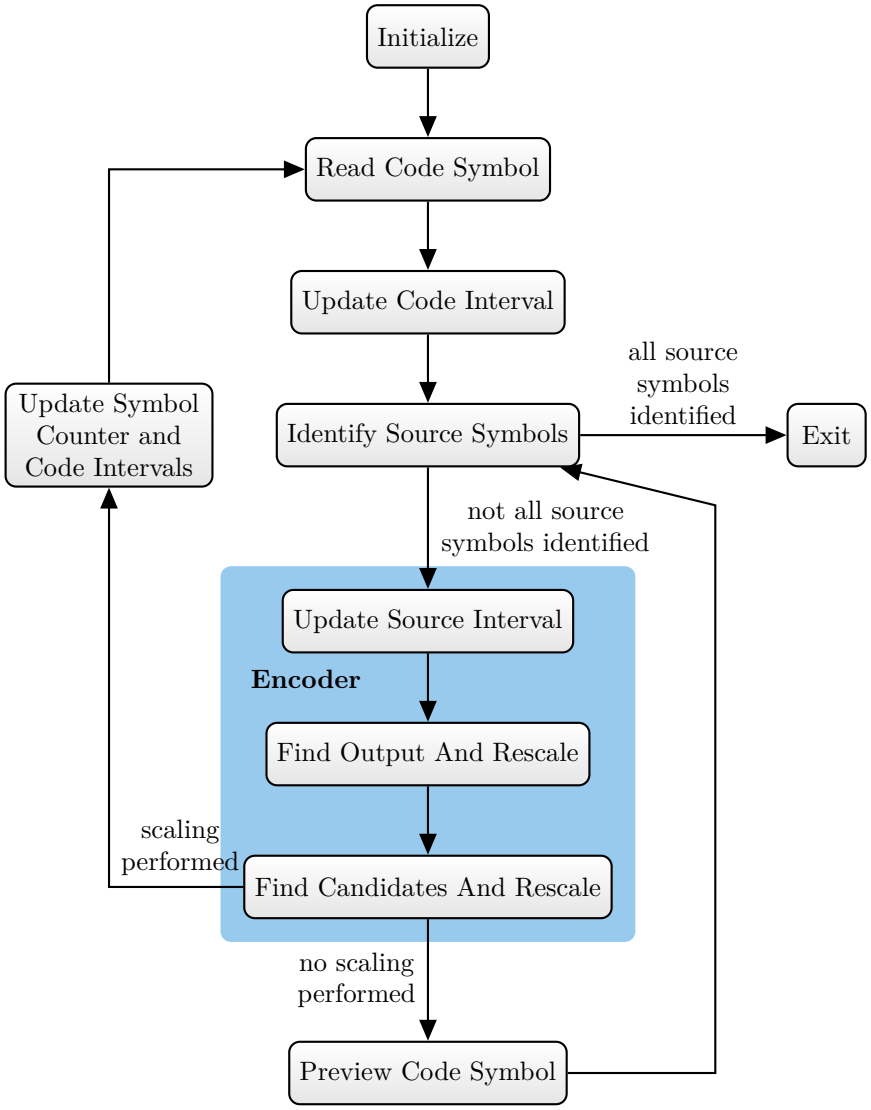


Figure 2.6: On-the-fly arithmetic decoding scheme.

is located in this code interval, because the finalization of the encoding process took care of that property.

However, decoding turns out to be more complex than the encoding part because of the linear scaling. We need to take care that the scalings calculate the same borders because rounding errors accumulate. For this reason we put parts of the encoder inside the decoder. The encoder worked with a source interval and a code interval. The decoder needs a supplementary interval that we call code candidates.

We start reading the first code symbols and update the code interval. If the code interval identifies source symbols, then we feed this information directly into an encoder that checks for possible scaling operations. If we did not identify enough source symbols

for a scaling, then we read upcoming code symbols that refine the code interval again. We go back to identifying source symbols. If there was a scaling performed, then we obtain new candidates and candidate borders that we feed back to the start of the procedure.

To put it into a nutshell: We refine a code interval until it identifies enough source symbols to imitate the behavior of the encoder. If the encoder performs a scaling, then we reset the code interval to the state of the encoder and restart the procedure.

Example 2.4 shows how to encode the sequence '10' in 4 steps to '1011'. The reverse operation takes 7 steps.

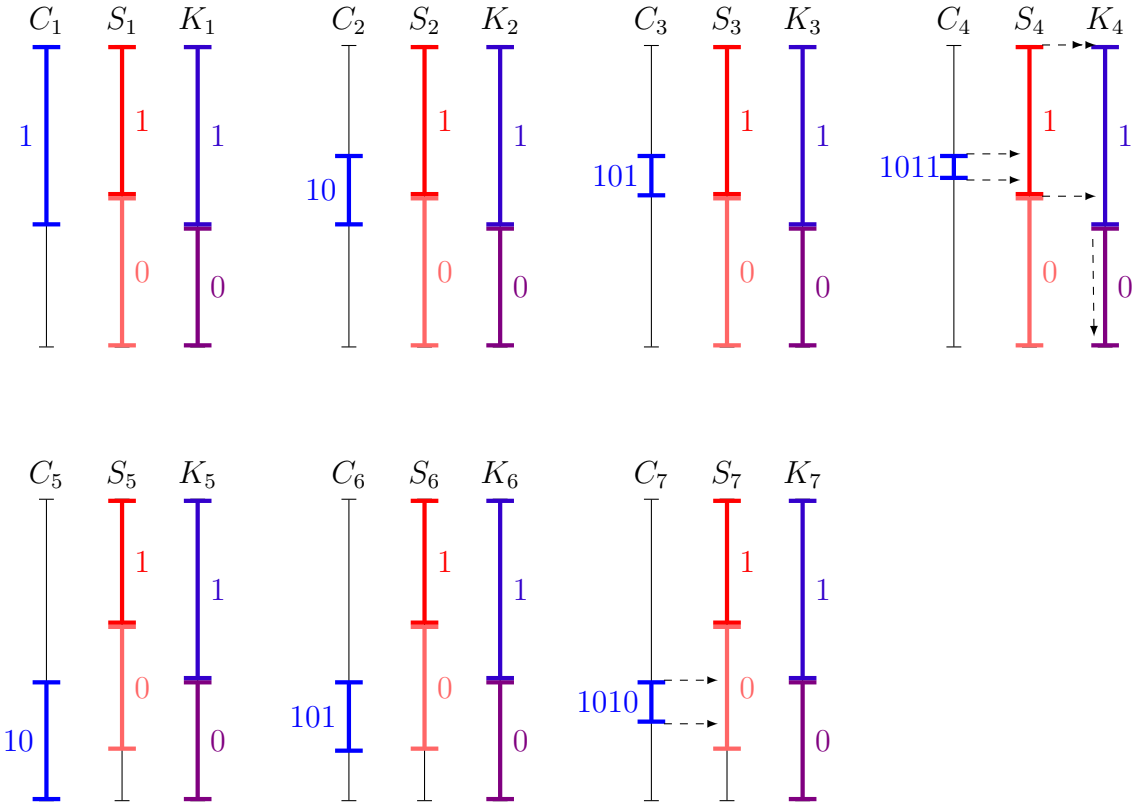


Figure 2.7: Decoding '1011'. C_i, S_i, K_i label the code interval, source interval and code candidate at the i-th step, respectively.

Example 2.5 (Arithmetic Decoding)**.** All steps of the decoding process are illustrated in Fig. 2.7. C_1, S_1, K_1 show the state of the code interval, source interval and current code candidates after initializing and reading the first code symbol. The first code symbol is '1'. Consequently the interesting interval is $[0.4, 1)$. There is no source symbol we could identify and we therefore refine. Also for the next two steps there is only refinement of the code interval. The second bit is '0' and the resulting code interval is $[0.4, 0.64)$. '1' leads to $[0.496, 0.64)$. S_2, S_3, K_2 and K_3 remain unchanged. The fourth symbol

defines the interval $[0.5536, 0.64)$ which is a subset of $[0.5, 1)$. We can write out the source symbol '1'. We put this input back into the encoder-simulator and observe that the encoder identifies the first code word, because the source interval $[0.5, 1)$ is a subset of the code candidate $[0.4, 1)$. The encoder performs a scaling according to (2.48) with $u = 1$, $l = 0.5$. The source interval transforms from $[0.5, 1)$ to $[1/6, 1)$. We also write the state of the encoder-simulator back to the code interval. Note that the input pointer resets to the second symbol. C_5, S_5, K_5 show this new state after reading again the second source symbol. One can see from the figure that the next source bit must be '0'. However, the algorithm suggests to refine until the source symbol is clearly identified. This is achieved after reading the fourth symbol, and we are done. As the length of the source sequence is fixed, we know that we have decoded the complete sequence and stop the algorithm.

2.7 Information Theoretic Transmission Problem

Energy Efficient Communication

We are interested in reliable communication at high rates over a noisy channel. Information theory suggests to maximize the mutual information [1], i.e, the channel capacity is

$$C = \max_{P_{\mathsf{X}}} \mathbb{I}(\mathsf{X}; \mathsf{Y}) \tag{2.61}$$

where X is the channel input and Y is the channel output. Equation (2.61) is not necessarily a complete description of the transmission problem because we often need to add constraints to X, e.g., a power constraint.

Channel model. To solve problem (2.61) we need access to $P_{\mathsf{Y}|\mathsf{X}}(y|x)$. For AWGN channels, we have

$$\mathsf{Y} = \mathsf{X} + \mathsf{Z} \tag{2.62}$$

where Z is Gaussian noise with zero mean and variance σ_N^2. For AWGN channels we must include a power constraint to obtain meaningful results.

Transceiver design. The transceiver chooses the signal set and is limited due to quantization, among other considerations. This may lead to further constraints on the input distribution and its support. We detail the transceiver in Sec. 2.8.

The above list is not a complete description. Eventually, we compute a constrained capacity-achieving input distribution directly from the solution of (2.61) by using the Blahut-Arimoto algorithm [41, 42].

For energy efficient communication, suppose that P_{X} is the capacity-achieving input distribution of a discrete memoryless channel (DMC) with capacity C. Let $\tilde{\mathsf{Y}}^n$ be the

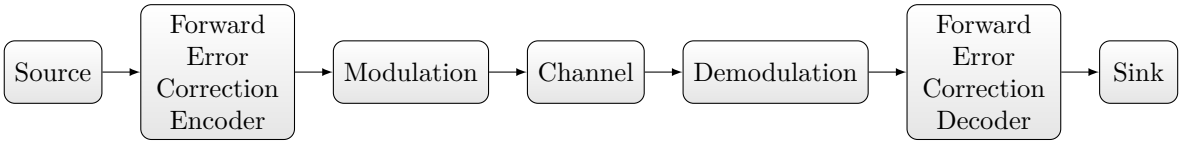


Figure 2.8: Transmission chain from source to sink.

channel output for an input $\tilde{\mathsf{X}}^n$. If $\mathbb{I}(\tilde{\mathsf{X}}^n;\tilde{\mathsf{Y}}^n)$ is the mutual information between the input and output sequences, then we have [15, eq. (23)]

$$C - \frac{\mathbb{D}\left(P_{\tilde{\mathsf{X}}^n} \| P_{\mathsf{X}}^n\right)}{n} \leq \frac{\mathbb{I}(\tilde{\mathsf{X}}^n;\tilde{\mathsf{Y}}^n)}{n} \leq C. \tag{2.63}$$

Hence, a small *normalized* divergence guarantees a mutual information close to capacity.

Stealth Communication

For stealth communication, suppose that an adversary wants to detect a transmission over a DMC, i.e., the adversary is interested in the *activity* rather than the *content*. Suppose that P_{Y} is the distribution the adversary expects to observe at the channel output when *no* transmission occurs, and suppose that P_{X} is a distribution for which the channel responds with exactly this output distribution. Let again $\tilde{\mathsf{Y}}^n$ be the channel output for an input $\tilde{\mathsf{X}}^n$. In [43, Sec. IV & Lemma 1], the authors showed that if

$$\mathbb{D}\left(P_{\tilde{\mathsf{Y}}^n} \| P_{\mathsf{Y}}^n\right) \to 0 \tag{2.64}$$

as $n \to \infty$, then the best an adversary can do is to guess without observing $\tilde{\mathsf{Y}}^n$. By the data processing inequality [28, Theorem 2.8.1], we have [43]

$$\mathbb{D}\left(P_{\tilde{\mathsf{X}}^n} \| P_{\mathsf{X}}^n\right) \geq \mathbb{D}\left(P_{\tilde{\mathsf{Y}}^n} \| P_{\mathsf{Y}}^n\right). \tag{2.65}$$

That is, zero *unnormalized* divergence guarantees *stealth*.

2.8 Coded Modulation

We use a simplified model for a transceiver consisting of the following components. The main purpose is to show how PAS changes the transceiver.

Source

The source emits Bernoulli(1/2) distributed bits. This is a good model for compressed sources and also a fair point to start if we do not know the source statistics.

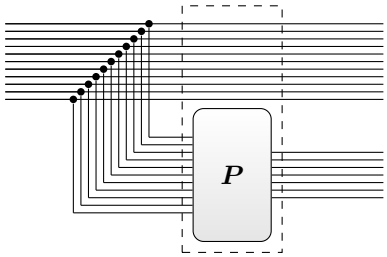


Figure 2.9: Block diagram of a systematic FEC encoder of the form $\boldsymbol{G} = [\boldsymbol{I}|\boldsymbol{P}]$.

Forward error correction encoder

A noisy channel changes the transmitted symbols. A FEC code translates a message into a vector in a higher dimensional space and increases the "distance" between messages. We can interpret this as adding redundancy because we use a larger space than necessary to represent the message. The increased distance helps the decoder to recover the message. We are interested in binary linear block codes, i.e., we can represent the encoding as a matrix vector multiplication $\boldsymbol{u} \cdot \boldsymbol{G}$ in the binary field GF(2), where the vector $\boldsymbol{u}$ is the message and the matrix $\boldsymbol{G}$ is a generator matrix of the code. Note that the generator matrix is not unique and there exists (except for pathological cases) a systematic encoding such that the message is preserved in the code word, i.e., there are columns that are all zero except for one entry that is '1'. We consider generator matrices of the form

$$\boldsymbol{G} = [\boldsymbol{I}|\boldsymbol{P}] \tag{2.66}$$

which is the concatenation of an identity matrix $\boldsymbol{I}$ and a parity matrix $\boldsymbol{P}$. We depict the systematic encoding in Fig. 2.9.

Modulation

A modulator is a mapping from binary strings to either the real numbers (1-D) or the complex plane (2-D). We call the image of the mapping a constellation and each element of the image a constellation point. If the number of constellation points per dimension is greater than 2 then we use the term *higher order modulation*.

We differentiate between:

- ASK and quadrature amplitude modulation (QAM) constellations, i.e., constellations on a uniformly spaced grid, and other constellations. For uniformly spaced constellations all signal points are similarly reliable and therefore their energy plays a differentiating role.

- ▷ Multi-ring constellations and non-uniform constellations (NUCs) [9,10] are prominent in satellite communications. Some NUCs are known as geometric shaping [44, 45]. Here, more signal points are close to the origin, which reduces the average energy. At the same time these constellations are less reliable because the points are closer together.

In this thesis we are interested in QAM and ASK constellations. A M-ASK constellation is a 1-D constellation that consists of M equally spaced points around the origin, i.e.,

$$\{-(M-1), \ldots -3, -1, 1, 3, \ldots, (M-1)\}$$

where M is even. We call M the modulation order.

A M-QAM constellation is a 2-D constellation on a square grid and can be expressed as the Cartesian product of two $\sqrt{M}$-ASK constellations if $\sqrt{M}$ is an integer. We do not consider the case where $\sqrt{M}$ is not an integer.

Practical systems usually use constellation orders M that are powers of 2 because we use $\log_2 M$ bits to address M constellation points. We call the mapping from bits to constellation points a *labeling*. In Fig. 2.10 we see 8-ASK and 64-QAM constellations with binary reflected Gray code (BRGC) labelings.

Demodulation

A demodulator transforms the received, noisy symbols from the channel into likelihoods according to a channel model. A symbol metric [46] or multi-level [47,48] demodulation for higher order modulation can be complex. There are many simplifications such as bitwise demapping [49].

Forward error correction decoder

The FEC decoder tries to find the most likely code word according to the channel observations. This problem is difficult and decoders use structure in the code to simplify decoding, e.g., to find a possibly suboptimal solution with reasonable computational complexity.

Figure 2.10: 8-ASK and 64-QAM constellations with BRGC labelings.

3

Probabilistic Shaping and Probabilistic Amplitude Shaping

In this chapter we motivate the shaping gain from an energy perspective for the AWGN channel. We then introduce PAS [11] to combine FEC, modulation and shaping. DM is an important building block of PAS and we define its interface and desired properties in the last section.

3.1 Energy Perspective

3.1.1 Continuous Constellations

We investigate a simplified model of CM to develop intuition for DM. Consider a transmission block of length n and an equal spacing between transmission symbols per dimension. With a growing constellation order M per dimension we approximate a continuous constellation.

Consider the n-cube (also known as the n-dimensional hypercube) with support volume V and density ρ (number of points per volume unit). The density is zero outside the support and constant at ρ_0 inside the support. The integral over the support should capture the number of constellation points, i.e., for the n-cube we have

$$M^n = \int_V \rho_0 \, \mathrm{d}V \tag{3.1}$$

Note that eq. (3.1) is valid for any support. We define the bit rate $\bar{m}$ to express the

average number of bits transmitted per dimension, i.e., we have

$$\bar{m} = \log_2 \left(\int_V \rho_0 \, \mathrm{d}V \right) / n. \tag{3.2}$$

For the n-cube this evaluates to $\log_2 M$.

When we scale each dimension of an n-D constellation by a factor Δ greater than one the volume increases by Δ^n while the number of transmission points remains the same. Thus, the new density is $\rho_\Delta = \rho_0 / \Delta^n$. This scaling corresponds to increasing the transmission power by a factor of Δ^2.

Note that introducing a FEC code changes the density by reducing the number of points inside a support. This effect is rather different than scaling because the FEC code changes the geometry including the number of nearest neighbors. The density model does not capture the coding gain because it loses information about the geometry. The following arguments also hold when the geometric structure of the points does not change, i.e., we use the same code.

Probabilistic shaping can form power efficient constellations. The most power efficient shape of equally spaced points is an n-ball. We want to compare the average energy of an n-ball and an n-cube with same volume and density.

The volume of an n-cube with side length A is

$$V_C = A^n \tag{3.3}$$

and its energy is

$$E_C = \int_{-\frac{A}{2}}^{\frac{A}{2}} \int_{-\frac{A}{2}}^{\frac{A}{2}} \dots \int_{-\frac{A}{2}}^{\frac{A}{2}} \left(x_1^2 + x_2^2 + \dots + x_n^2 \right) \mathrm{d}x_n \dots \mathrm{d}x_2 \, \mathrm{d}x_1. \tag{3.4}$$

where the x_i are the Cartesian coordinates. We obtain

$$E_C = n \cdot \frac{1}{12} A^{n+2} = n \cdot \frac{1}{12} A^2 V_C. \tag{3.5}$$

The volume of an n-ball with $n = 2k$, k an integer, is

$$V_B = \int_V 1 \mathrm{d}V = \tag{3.6}$$

$$= \int_0^{2\pi} \int_0^{\pi} \dots \int_0^{\pi} \int_0^{R} r^{n-1} \sin^{n-2}(\phi_1) \dots \sin(\phi_{n-2}) \mathrm{d}r \mathrm{d}\phi_1 \dots \mathrm{d}\phi_{n-1} \tag{3.7}$$

$$= \frac{\pi^{n/2}}{(n/2)!} R^n. \tag{3.8}$$

To calculate the energy, we integrate over the squared distance to the origin r^2, i.e.,

$$E_B = \int_V r^2 \, \mathrm{d}V = R^2 \frac{n}{n+2} V_B. \tag{3.9}$$

n-cubes and n-balls have the same volume if

$$1 = \frac{V_C}{V_B} = \frac{A^n}{\frac{\pi^{(n/2)}}{(n/2)!} R^n} \Rightarrow \frac{A}{R} = \frac{\sqrt{\pi}}{\sqrt[n]{(n/2)!}}. \tag{3.10}$$

We may approximate the denominator using Stirling's approximation and obtain

$$\frac{A}{R} \approx \sqrt{\frac{2\pi e}{n}} \cdot \sqrt[2n]{\pi n}. \tag{3.11}$$

Comparing the energy for the same volume, we obtain

$$\frac{E_C}{E_B} = \frac{\frac{1}{12} n A^2}{R^2 \frac{n}{n+2}} = \frac{\frac{1}{12}(n+2)\pi}{\sqrt[n/2]{(n/2)!}} \approx \frac{\frac{1}{6}\pi e}{\sqrt[n]{\pi n}} \frac{n+2}{n}. \tag{3.12}$$

For large n, we can drop the second factor and the root in (3.12) because they converge to 1. We obtain the well known shaping gain [6, 50]

$$\frac{E_C}{E_B} = \frac{\pi e}{6} \approx 1.53\text{dB}. \tag{3.13}$$

For this calculation we assumed a constant density inside the volume. A FEC code decreases the number of points and if a constant density is maintained we can obtain a shaping gain on top of the FEC gain.

Equations (3.12) and (3.10) give insight about the shaping gain in the finite length regime. In Fig. 3.1 we show the shaping gain E_C/E_B for continuous constellations in dB vs. the number of dimensions used for shaping. The shaping gain increases rather quickly as the number of dimensions grows. However, we need to form a perfect ball over about 800 dimensions for a shaping gain of 1.5 dB.

Example 3.1 (Continuous shaping gain - 2D)**.** We consider a simple example, i.e., continuous shaping in 2D. Let us normalize the volume (area) to 1, which results in a square with length $A = 1$ and a circle of radius $R = \frac{1}{\sqrt{\pi}}$. The energy of the square is 1/6 and the energy of the circle is $\frac{1}{2\pi}$. The ratio of energies evaluates to $\frac{E_C}{E_B} = \frac{\pi}{3} \approx 1.0472$. We are allowed to scale the constellation by a factor of $\sqrt{\frac{E_C}{E_B}} = \sqrt{\frac{\pi}{3}} \approx 1.0233$ which corresponds to 0.2dB from unshaped to shaped in 2 dimensions. Fig. 3.2a visualizes the continuous 2 dimensional constellations.

From (3.10) we can already see a disadvantage of n-balls in comparison with n-cubes.

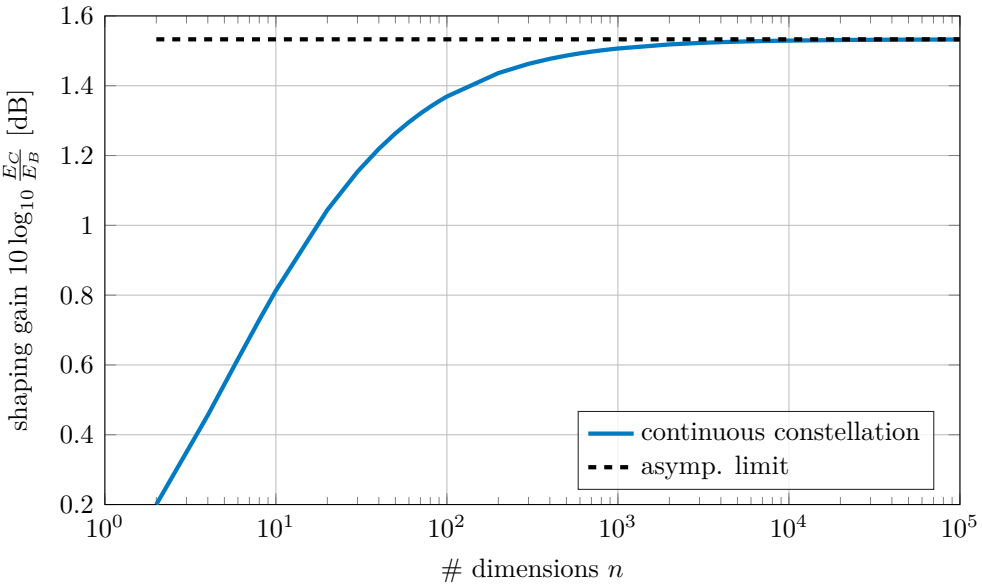


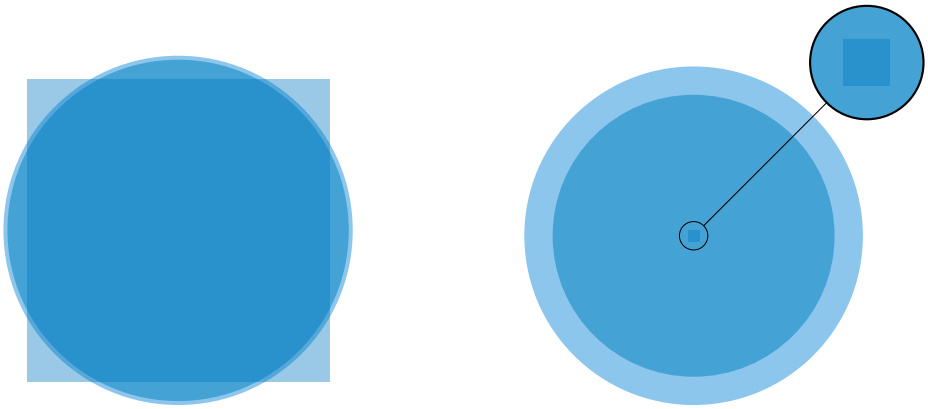
Figure 3.1: Shaping gain vs. block length.

If we compare the radius of an n-ball with the side length of an n-cube for a fixed rate, we see that the side length of the cube does not increase while the radius of the ball increases with $\sqrt{n}$, see Fig. 3.3. In other words, the range of each dimension is smaller for the n-cube than for the n-ball. At the same time, the longest diagonal of the cube grows with the same speed in the number of dimensions n. For a system design we prefer that the range of the constellation does not grow. Fig. 3.2 illustrates the growth of the radius in comparison to the side length of a cube and the shaping gain. Fig. 3.2a shows the proportions for the case $n = 2$ and Fig. 3.2b for $n = 10000$. We see a projection on two dimensions. The n-cube collapses to the square and the n-ball and the dark circle, respectively. We choose a projection where 9998 dimensions of the n-cube collapse. The light circle depicts the projection of a scaled n-ball with the same energy as the n-cube. In Fig. 3.2a the square and the darker circle have the same area. There is nearly no visible difference. In Fig. 3.2b we can clearly see the shaping gain from the dark circle to the light circle. Fig 3.3 shows the side length, the longest diagonal of an n-cube and the radius of an n-bsall with same volume.

3.1.2 Discrete Constellation

Continuous constellations give an intuition where the shaping gain stems from and they allow to estimate gains for practical systems. We now consider ASK constellations $\mathcal{X}$ and higher dimensional ASK constellations $\mathcal{X}^n$. An M-ASK constellation $\mathcal{X}$ consists of the points

$$\mathcal{X} = \{-M+1, -M+3, \ldots, M-1\} \tag{3.14}$$



(a) 2D (b) 10000D projection on 2D

Figure 3.2: Shaping gain and dimension effects in higher dimensions.
(a) The square and the inner, dark circle have the same area. The outer, light circle is scaled such that it has the same average energy as the square. Since the circles are hard to distinguish, the shaping gain is small.
(b) This is a projection of the a 10000-cube, and two 10000-balls onto 2 dimensions. The cube and the dark n-ball have same volume. The outer, light ball is scaled such that it has the same average energy as the cube. The projection is chosen such that 9998 dimensions of the cube collapse.

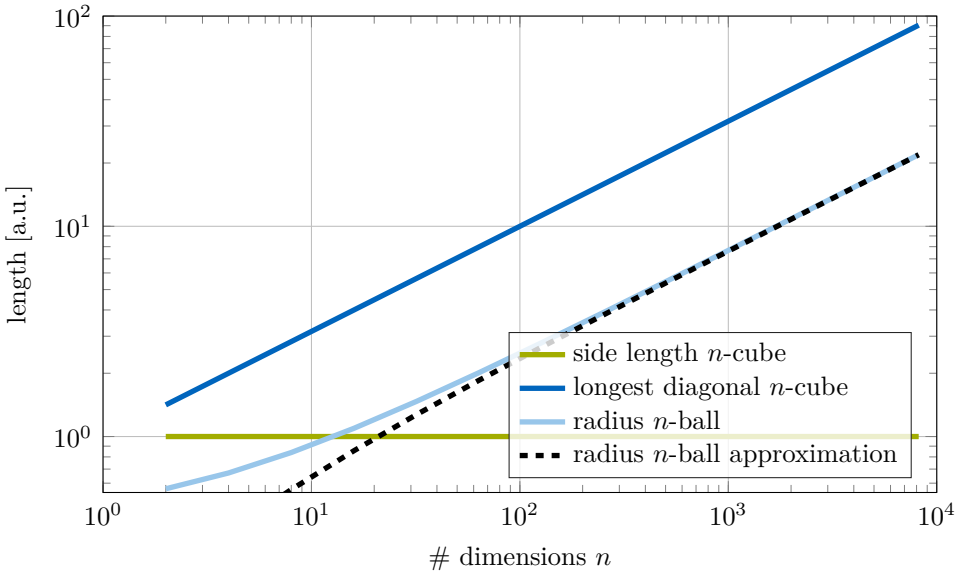


Figure 3.3: Side length(—) and longest diagonal (—) within an n-cube in comparison to the radius of an n-ball (—) with same volume. The approximation of the n-ball radius (- - -) fits quite well for dimensions larger than 8.

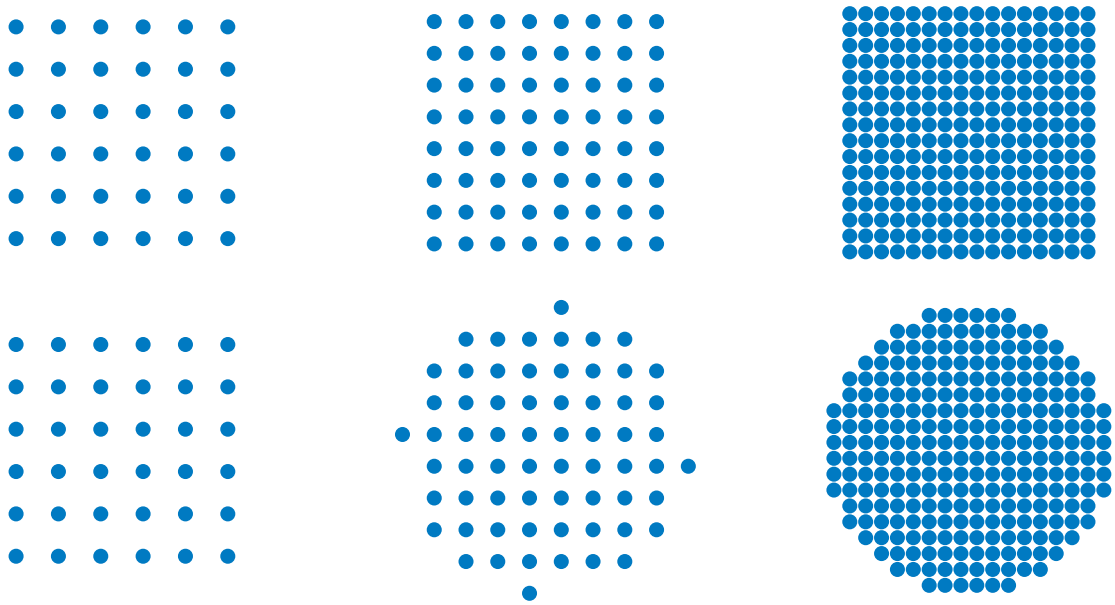


Figure 3.4: The upper diagrams depict 36, 64 and 256-QAM constellations. The second row shows constellations that have 36, 64 and 256 constellation points of lowest energy, i.e., smallest distance to the origin in the same grid. Observe that for 36 points the constellations are identical.

where M is an even integer. We can address this constellation with $m = \log_2 |\mathcal{X}|$ bits.

Consider an n-D constellation $\mathcal{X}'$ that is defined on the same grid as an n-D ASK constellation $\mathcal{X}^n$, i.e., $\mathcal{X}' \subseteq \mathcal{X}^n$. We call the constellation $\mathcal{X}$ the *base constellation* of $\mathcal{X}'$ and we call the order of $\mathcal{X}$ the *base constellation order* M_B. Consider, e.g., the bottom right constellations in Fig. 3.4. The base constellation is 18-ASK and the base constellation order is 18.

For discrete, n-D constellations $\mathcal{X}'$, the average constellation bit rate becomes

$$\bar{m} = \frac{\log_2 |\mathcal{X}'|}{n}. \tag{3.15}$$

The average energy per symbol and dimension is

$$E_{X'} = \frac{1}{n\,|\mathcal{X}'|} \sum_{x^n \in \mathcal{X}'} \sum_{i=1}^{n} x_i^2. \tag{3.16}$$

For a M-ASK constellation this expression simplifies to

$$E_X = \frac{1}{M} \sum_{i=1}^{M} (2i-1)^2 = \frac{1}{3}[4M^2 - 1]. \tag{3.17}$$

We anticipate two problems for energy efficient discrete constellations.

Problem 1 In the continuous case, the radius of an n-ball with same volume and density (and therefore rate) as an n-cube grows for higher dimensions, while the side-length of the cube is constant. At the same time we need to use many dimensions for a reasonably high shaping gain. A growing radius in the continuous case corresponds to an increasing number of signal points per dimension for discrete constellations, i.e., to transmit on average $\bar{m}$ bits we need a M_B-ASK base constellation with $\log_2(M_B) \gg \bar{m}$ for high dimensions. This implies a more complex decoder. In Fig. 3.2b the side length of a 10000-cube is about 24 times smaller than the diameter of a ball with same volume. However, the longest diagonal exceeds the ball diameter by a factor greater than 2. We show numerical results how a restriction of the basic constellation order affects the shaping gain for short and long blocks.

Problem 2 Continuous constellations may approximate higher order modulation quite well. However, for small constellations the grid is rather coarse and it can be difficult to see a gain at all. Fig. 3.4 shows 36, 64 and 256 QAM constellations and a power efficient modulation on the same grid. 256 QAM resembles a circle because it is fine enough, but the effect on 64 QAM is rather limited. For 36 QAM there is no better rearrangement, i.e., the constellations are identical and there is no shaping gain. Intuitively, for longer blocks we expect a positive effect because the number of corner points increases and the diagonals grow.

Figures 3.5, 3.6 and 3.7 show the energy ratio $E_X/E_{X'}$ in dB where E_X is the energy of an M-ASK constellation and $E_{X'}$ is the energy of a shaped constellation. The shaped constellations consist of the M^n least energy points of an n-D grid with M_B-ASK base constellation and $M_B > M$. We added the energy ratio E_B/E_C of the continuous constellations (3.12) as references, and paid attention that the largest base constellation M_B does not restrict the energy ratio in the region shown, i.e., performance does not improve for a larger M_B. Today's systems use constellation orders that are powers of 2. Consequently, we consider only integer values for $\bar{m}$ in the comparison. Shaped constellations require $\bar{m} \cdot n$ to be an integer value , i.e., we can choose $\bar{m}$ more flexibly than for ASK constellations.

Observe that restricting the base constellation may lead to a significant loss in $E_{X'}/E_X$. However, a base constellation of $M_B \approx 2 \cdot M$ gives close to optimal performance for block lengths up to 300.

We address Problem 2 in Fig. 3.8, where the base constellation is not restricted and we display $\bar{m}$ from 2 to 5. Note that $E_{X'}/E_X$ depends on the constellation order and a higher constellation allows a higher shaping gain. Still, there is a gain for low constellation orders. For $\bar{m} = 2$ the result is not smooth which stems from the discrete nature of the optimization problem.

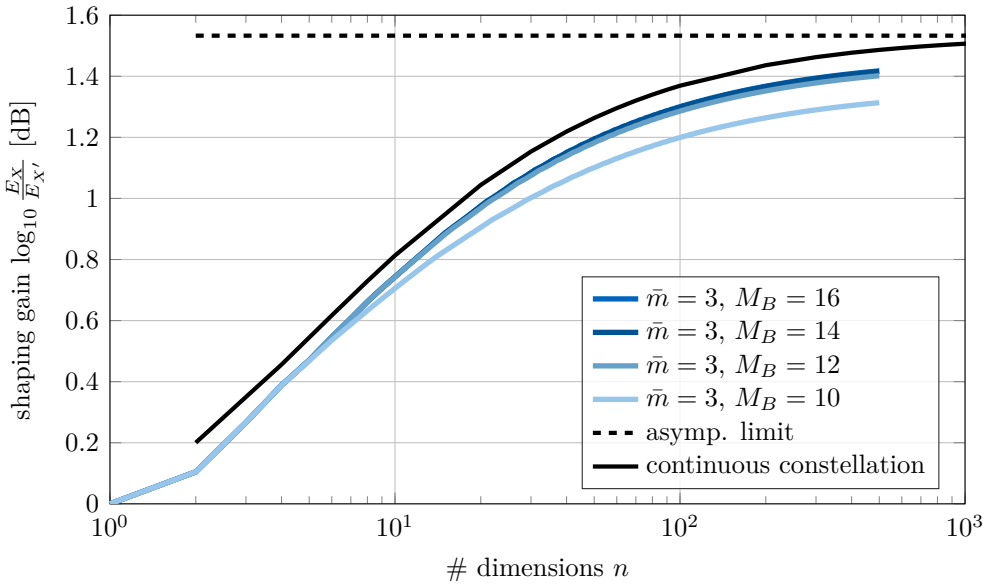


Figure 3.5: Shaping gain of a discrete constellation with rate $\bar{m} = 3$ per dimension over an 8-ASK constellation using a base constellation order M_B of 10 to 16 points.

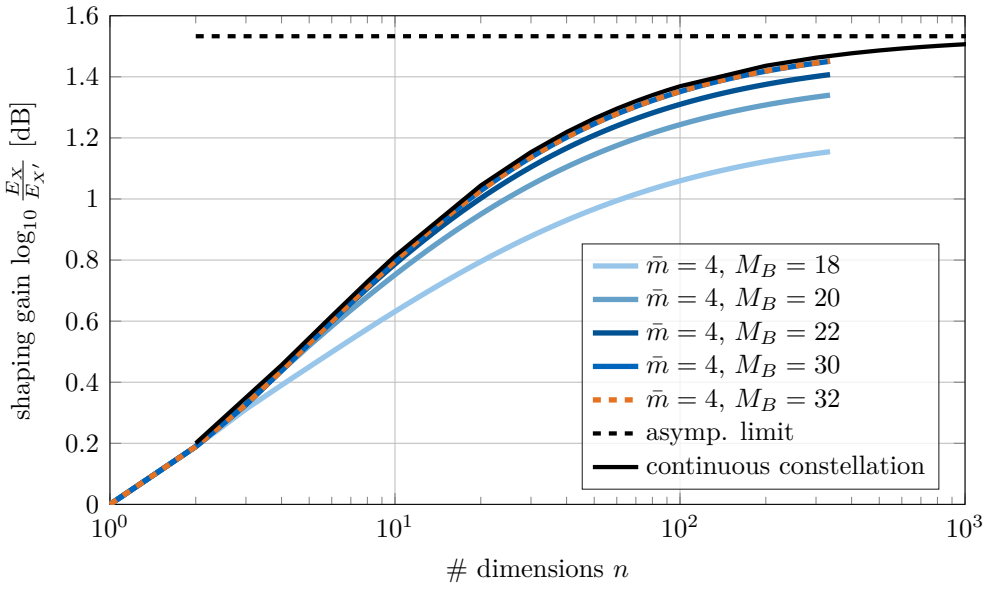


Figure 3.6: Shaping gain of a discrete constellation with rate $\bar{m} = 4$ per dimension over a 16-ASK constellation using a base constellation M_B of 16 to 32 points.

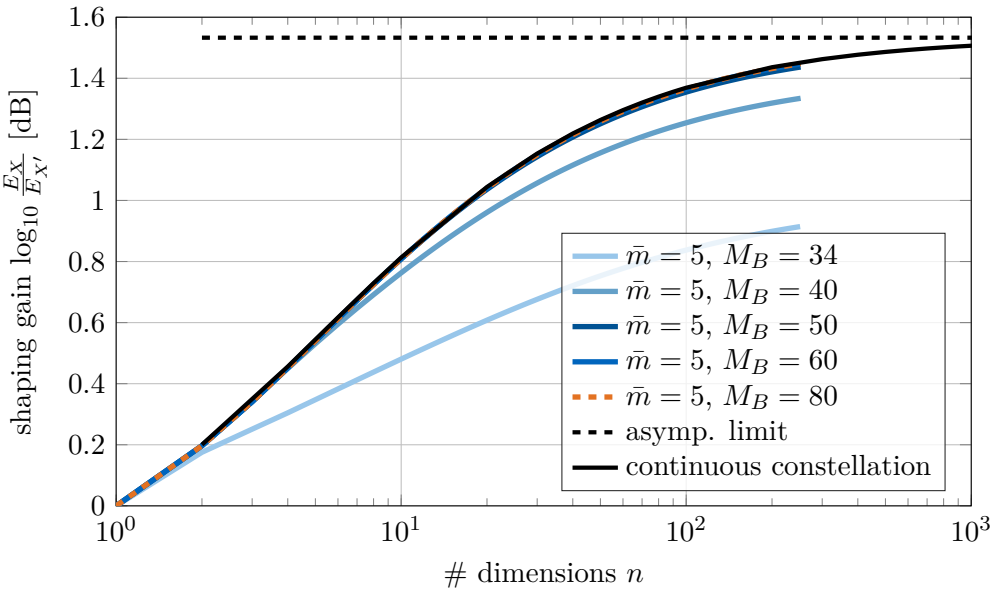


Figure 3.7: Shaping gain of a discrete constellation with rate $\bar{m} = 5$ per dimension over a 32-ASK constellation using a base constellation M_B of 34 to 80 points.

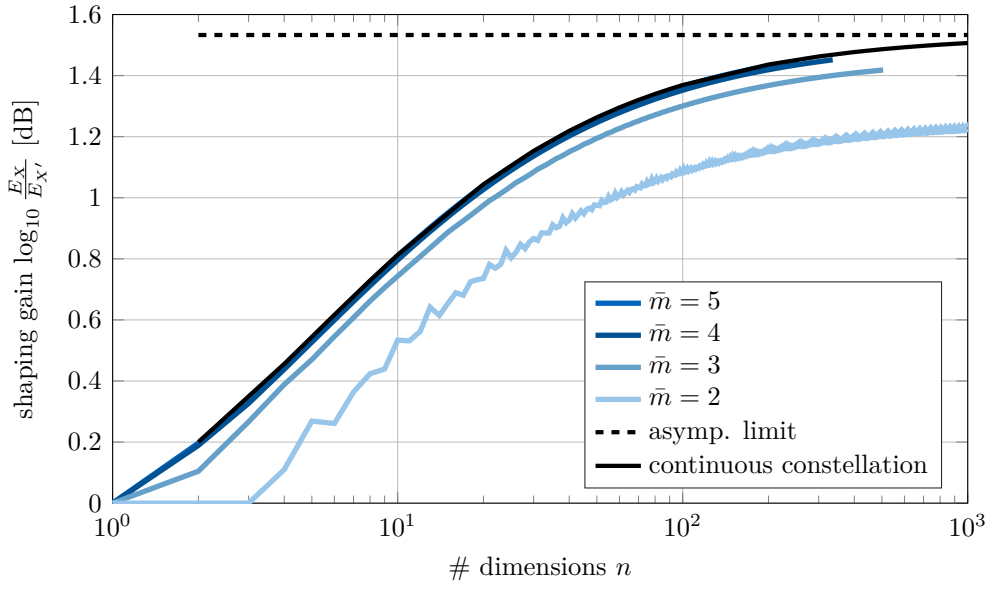


Figure 3.8: Shaping gain of discrete constellations with different average modulation rates.

3.2 Probabilistic Amplitude Shaping

In [11] Böcherer *et al.* introduce a transceiver architecture called PAS that separates FEC and shaping. Shaping requires a DM.

3.2.1 Underlying Principles

Probability Factorization

The optimal input distribution for the AWGN channel has a symmetry because the energy of a constellation point is independent of its sign and the constellation is symmetric around the origin. Therefore it is possible to write the optimal input distribution as the product distribution

$$P_{\mathsf{X}}(x) = P_{\mathsf{A}}(a) \cdot P_{\mathsf{S}}(s) \tag{3.18}$$

where S is a binary random variable that is Bernoulli(1/2) distributed and $a \in \mathcal{A}$ with $|\mathcal{A}| = |\mathcal{X}|/2$. Thus we can obtain the optimal input distribution in two steps. First, a random variable A selects one of the pairs (x_1, x_2) of symbols that have the same probability so that $P_{\mathsf{X}}(x_1) + P_{\mathsf{X}}(x_2) = 2P_{\mathsf{X}}(x_1)$ and then S selects one symbol of this pair. A and S are independent. This description is rather abstract but in the AWGN case with ASK modulation, there is a nice description for S and A: we interpret them as the amplitude and sign of a constellation point, i.e., $\mathsf{X} = \mathsf{S} \cdot \mathsf{A}$ with realizations $s \in \{-1, 1\}$ and $a \in \{1, 3, \dots M-1\}$. Note that this is not the only possible description, e.g., for 4-ASK we could choose $s \in \{-1, 1\}$ and $a \in \{-1, 3\}$ rather than $a \in \{1, 3\}$.

Uniform Check Bit Assumption

In Sec. 2.8 we defined the encoding of linear codes as a vector-matrix multiplication. For the special case of systematic encoding, we have

$$\boldsymbol{c} = \boldsymbol{u} \cdot \boldsymbol{G} = \boldsymbol{u} \cdot [\boldsymbol{I}|\boldsymbol{P}] \tag{3.19}$$

where $\boldsymbol{P}$ is usually rather dense. In [51, Sec. 7.1.3] it is shown that the distribution of the parity bits becomes uniform for an increasing number of nonzero entries in the columns of $\boldsymbol{P}$. This does not necessarily hold in the pathological case of having deterministic bits in $\boldsymbol{u}$. The multiplication with the identity matrix $\boldsymbol{I}$ copies $\boldsymbol{u}$ into the first part of $\boldsymbol{c}$, see also Fig. 2.9.

3.2.2 Encoding

For the encoding scheme, we go through the components of a CM scheme from the desired output towards the source(s).

Consider a transmission block of n of M-ASK constellation points, i.e., we need $m = \log_2 M$ bits to label a constellation point. We use a labeling that differs for symbols

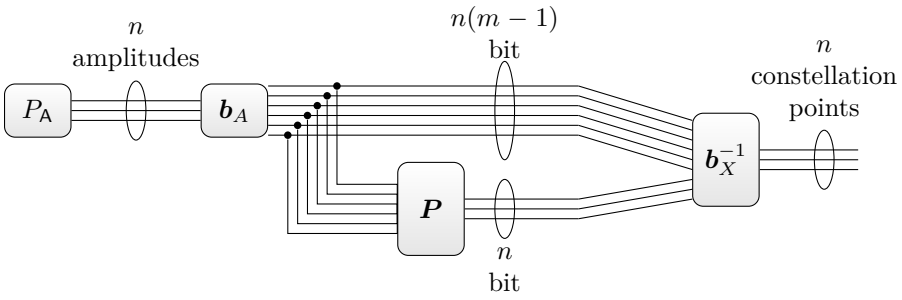


Figure 3.9: PAS encoder from the amplitude and bit perspective of one block. The labeling function $\boldsymbol{b}_A$ translates n amplitudes into $n(m-1)$ bits. The parity bits that are computed in $\boldsymbol{P}$ complete the amplitude bits.

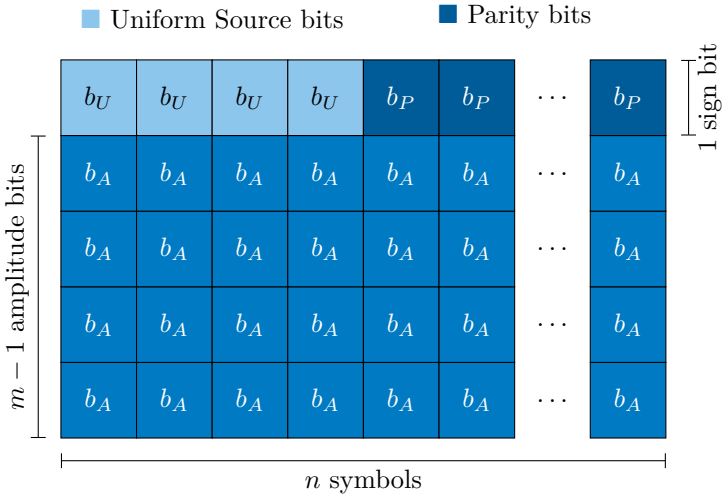


Figure 3.10: Collection of bits that represent one transmission block of n constellation symbols. $m-1$ bits represent one amplitude and one bit represents a sign. In extended PAS, some sign bits stem from the FEC encoder and some stem from a source P_U. In normal PAS, all sign bits are parity bits. While the sign bits can be permuted, the amplitude bits within one column are bound together.

with the same amplitude only in the first bit that we the call sign bit. A labeling that fulfills this requirement is the BRGC [52]. This way we obtain two labeling functions, one for amplitudes and one for constellation points

$$b_A \colon a \mapsto b_A(a) \in \{0,1\}^{m-1} \tag{3.20}$$

$$b_X \colon x \mapsto b_X(x) \in \{0,1\}^{m}. \tag{3.21}$$

We call their inverse functions b_A^{-1}, b_X^{-1} symbol mappers.

For transmitting n M-ASK symbols, we can choose $n \cdot m$ bits, or n amplitudes and n signs. In an uncoded scenario, i.e., without FEC, we need two DMSs. The first source produces amplitudes with probability P_{A} and the second DMS P_{U} emits Bernoulli(1/2) bits. The binary representations of one amplitude and one sign bit form one constellation point with the use of the symbol mapper b_X^{-1}. In a coded scenario, we can replace Bernoulli(1/2) distributed bits from the source P_{U} either completely (see PAS [11, Sec. IV]) or partially (see extended PAS [11, Sec. IV-D]) by the parity bits from the systematic FEC encoder. In Fig. 3.9 we can see the encoding process of PAS for a whole block. Fig. 3.10 shows how the parity bits and the amplitudes are arranged for extended PAS. More detailed descriptions of PAS can be found, e.g., in [11, 53, 54].

It remains to clarify how to obtain a DMS with distribution P_{A} from Bernoulli(1/2) bits. We develop theory and algorithms to accomplish this in the next chapter.

4

Fixed Length Distribution Matching Algorithms

4.1 Distribution Matching

In Sec. 2.7, equations (2.63) and (2.61) show the main goal of DMs, i.e., to approximate a distribution and to transmit data at maximum rate. A DM transforms sequences of independent, uniformly distributed symbols into sequences that approximate a DMS with a target distribution. A dematcher performs the inverse operation and recovers the input symbols from the output sequence.

Let us examine the architecture. The interface is shown in Fig. 4.1, where a sequence of uniformly distributed bits B^k is transformed into a sequence $\tilde{\mathsf{A}}^n$. We model the B^k bits as uniformly independent and identically distributed (iid) because this is a reasonable assumption after source coding. We want an invertible transformation on the image such that we can recover the original bit-sequence without error. We do not allow an internal source of randomness at the matcher, and therefore we consider one-to-one variable length mappings.

Optimal variable-to-fixed (v2f)- and fixed-to-variable (f2v)-length DMs are proposed in [6, 14–16]. The code books of these DMs must be generated offline and stored. Since this is infeasible for large code word lengths, algorithms were proposed that use arithmetic coding to calculate the code book online [17, 18]. All these approaches have either variable input or variable output lengths, which can lead to varying transmission rate, large buffer sizes, error propagation, and synchronization problems [6, Sec. I].

Fixed-to-fixed (f2f)-length DMs do not suffer from these problems. The authors of [19, Sec. 4.8] and [20] therefore suggest to use block codes to build a f2f-length DM. However, these schemes include many-to-one mappings and hence cannot always recover the input sequence without error. To overcome these problems, f2f-length one-to-one DMs were

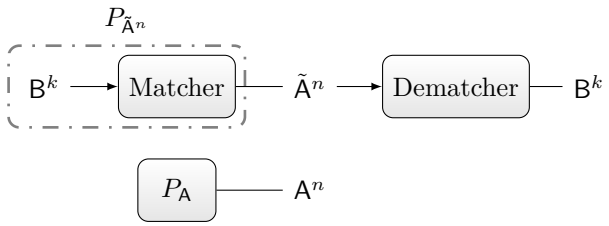


Figure 4.1: Matching input block $\mathsf{B}^k = (\mathsf{B}_1, \ldots, \mathsf{B}_k)$ to output symbols $\tilde{\mathsf{A}}^n = (\tilde{\mathsf{A}}_1, \ldots, \tilde{\mathsf{A}}_n)$ and reconstructing the original sequence at the dematcher. The DM should emulate a DMS with distribution P_{A}.

proposed.

In [21] and [55] arithmetic coding is used to index entropy-typical and letter-typical sequences, respectively. Both approaches are complex, especially for larger alphabets. The DMs proposed in [56,57] are similar to [55], and include a variable length code that preselects a type. The DMs of [58, 59] use enumerative coding to index a codebook of minimal energy sequences. This approach is limited to short block lengths because of memory consumption. This problem is addressed in [60] by using floating point arithmetic instead of integer arithmetic in a smart way. The performance loss is negligible. In [61] the complexity issue of DMs is solved for short blocks with concatenated small and medium sized look-up tables (LUTs). The DM proposed in [62] consists of variable length trees and a framing method that guarantees the f2f length property.

In [63] a combination of DM and FEC is proposed and further analysed in [64]. The method is based on polar codes [65,66] and is not restricted to symmetric distributions, i.e., it can be used for on-off keying (OOK), see [67]. This method can be used as a f2f-length one-to-one DM as well.

Depending on the DM type we have different expressions for divergence.

A fixed-to-fixed length DM is most constrained and most practical. For this class we have divergence expressions such as

$$\mathbb{D}\left(P_{\tilde{\mathsf{A}}^n} \| P_{\mathsf{A}}^n\right) = \sum_{\boldsymbol{c} \in \mathcal{C}} \frac{1}{|\mathcal{C}|} \log_2 \frac{1/|\mathcal{C}|}{P_{\mathsf{A}}^n(\boldsymbol{c})}. \tag{4.1}$$

All code words in the code book appear with same probability and the only choice is to decide if a word is in the code book or not. This also means that we have no further possibility to make the logarithm go to zero for all code words.

A variable-to-fixed length DM offers new degrees of freedom where the target probability $P_{\mathsf{A}}^n(\boldsymbol{c})$ of a code word $\boldsymbol{c}$ can be reflected by the number of bits $\ell(\boldsymbol{c})$ that

represent this code word. We have

$$\mathbb{D}\left(P_{\tilde{\mathsf{A}}^n} \| P_{\mathsf{A}}^n\right) = \sum_{\boldsymbol{c}\in\mathcal{C}} 2^{-\ell(\boldsymbol{c})} \log_2 \frac{2^{-\ell(\boldsymbol{c})}}{P_{\mathsf{A}}^n(\boldsymbol{c})}. \tag{4.2}$$

Note that the code word lengths must fulfill the Kraft inequality with equality, and that we can generate only dyadic distributions this way.

A fixed-to-variable length DM has divergence expressions of the form

$$\mathbb{D}\left(P_{\tilde{\mathsf{A}}^n} \| P_{\mathsf{A}}^n\right) = \sum_{\boldsymbol{c}\in\mathcal{C}} \frac{1}{|\mathcal{C}|} \log_2 \frac{\frac{1}{|\mathcal{C}|}}{P_{\mathsf{A}}^{\ell(\boldsymbol{c})}(\boldsymbol{c})} \tag{4.3}$$

where $\ell(\boldsymbol{c})$ is the variable length of the code word. Again, we select only uniform probabilities.

Resolution coding: In contrast to distribution matching, resolution coding [68] does not require a dyadic input distribution because we can map $m_{\boldsymbol{c}}$ input sequences to the sequence $\boldsymbol{c}$. This increases the flexibility even further, i.e., we obtain a divergence expression such as

$$\mathbb{D}\left(P_{\tilde{\mathsf{A}}^n} \| P_{\mathsf{A}}^n\right) = \sum_{\boldsymbol{c}\in\mathcal{C}} m_{\boldsymbol{c}} 2^{-\ell_{\max}} \log_2 \frac{m_{\boldsymbol{c}} 2^{-\ell_{\max}}}{P_{\mathsf{A}}^n(\boldsymbol{c})} \tag{4.4}$$

where $m_{\boldsymbol{c}} 2^{-\ell_{\max}}$ can basically build any pmf. For an overview of resolution coding and algorithms, we refer to [19].

Distribution Matching and Source Coding

Lossless source coding reduces the expected length of sequences that are the output of a DMS with non-uniform pmf P_{A}. The resulting bit stream is approximately Bernoulli(1/2) distributed. In [6, 14] Huffman codes are used for matching. The decoder of the source coding scheme is used as an encoder and vice versa. This way, a Bernoulli(1/2) bit-stream is encoded into symbols that are approximately distributed according to P_{A}. However, distribution matching has more degrees of freedom than the inversion of lossless source coding. Note that:

- ▷ An inverted lossless source code is invertible on the image. However, the image must be the full space of words, otherwise it would be lossy.
- ▷ A DM needs to be invertible on its image and the image can be a subspace of the whole space. This gives more possibilities to design a coding scheme. This is also the reason why there cannot be a f2f length lossless source coding scheme.

Fixed Length Distribution Matching

A one-to-one f2f DM is an invertible function f from words of length k to words of length n. We denote the inverse function by f^{-1}. The mapping imitates a desired distribution P_{A} by mapping k Bernoulli(1/2) distributed bits B^k to length n strings $\tilde{\mathsf{A}}^n = f(B^k) \in \mathcal{A}^n$. We refer to the image of a DM as the code book $\mathcal{C} := f(\{0,1\}^k)$ and its elements as code words. As bit sequences of length k uniquely index the code words in the code book, and every bit sequence has probability 2^{-k}, every code word has probability $1/|\mathcal{C}| = 2^{-k}$. Consequently, the explicit mapping from input to output does not affect divergence, only the code book matters. The pmf of code word $\boldsymbol{c}$ is thus

$$P_{\mathcal{C}}(\boldsymbol{c}) = \frac{1}{|\mathcal{C}|} \cdot \mathbb{1}(\boldsymbol{c} \in \mathcal{C}) \tag{4.5}$$

where $\mathbb{1}(\cdot)$ evaluates to one if the argument is true and zero otherwise. The concept of one-to-one f2f distribution matching is illustrated in Fig. 4.1.

Definition 4.1. A matching rate $R = k/n$ is achievable for a distribution P_{A} if for any $\epsilon > 0$ and sufficiently large n there is an invertible mapping $f\colon \{0,1\}^k \to \mathcal{A}^n$ for which

$$\frac{\mathbb{D}\left(P_{\mathcal{C}} \| P_{\mathsf{A}}^n\right)}{n} \leq \epsilon. \tag{4.6}$$

The following proposition in [69] relates the rate R and (4.6).

Proposition 4.1 (Converse, [69, Proposition 8])**.** There exists a positive-valued function δ with

$$\delta(\epsilon) \xrightarrow{\epsilon \to 0} 0 \tag{4.7}$$

such that (4.6) implies

$$\frac{k}{n} \leq \frac{P_{\mathsf{A}}}{P_{\mathsf{B}}} + \delta(\epsilon). \tag{4.8}$$

Proposition 4.1 bounds the maximum rate that can be achieved under condition (4.6). Since $\mathbb{H}\left(P_{\mathsf{B}}\right) = 1$, for vanishing informational divergence we have (see [69])

$$R = \frac{k}{n} \leq \mathbb{H}\left(P_{\mathsf{A}}\right) \tag{4.9}$$

for any achievable rate R.

The divergence between the approximated source and the target source can be written as follows:

$$\mathbb{D}\left(P_{\mathcal{C}} \| P_{\mathsf{A}}^n\right) = n\mathbb{H}\left(P_{\bar{\mathsf{A}}}\right) - k + n\mathbb{D}\left(P_{\bar{\mathsf{A}}} \| P_{\mathsf{A}}\right) \tag{4.10}$$

where $P_{\bar{\mathsf{A}}}$ is the empirical distribution that we obtain when we draw randomly one symbol from the code book, i.e., it is the marginal distribution over all code words and

along all indices of the code words:

$$P_{\bar{\mathsf{A}}}(a) = \frac{1}{|\mathcal{C}|} \sum_{\alpha^n \in \mathcal{C}} \frac{n_a(\alpha^n)}{n}. \tag{4.11}$$

We call $P_{\bar{\mathsf{A}}}$ the *letter distribution* of the code book $\mathcal{C}$.

For system design we want to compare different algorithms or code books. In [70], we suggest to consider the difference between the entropy of the letter distribution and the DM rate, i.e.,

$$\mathbb{H}\left(P_{\bar{\mathsf{A}}}\right) - \frac{k}{n} = \mathbb{D}\left(P_{\mathcal{C}} \| P_{\bar{\mathsf{A}}}\right). \tag{4.12}$$

Here, we have an operational meaning for the divergence. We sometimes call this difference of entropy and rate the *rate loss* R_{loss} [71].

4.2 Divergence Optimal Distribution Matching

In this section we derive the optimal DM and show its limitations in terms of divergence for large block lengths. We then give two algorithms to index the respective code books. One of them is SM, which is why we use the term shell-mapping distribution matching (SMDM)

4.2.1 Codebook Construction

In [19, Sec. 4.4.1] a minimum divergence code book construction for a fixed-length DM is proposed. We rewrite the divergence as

$$\begin{aligned}
\mathbb{D}\left(P_{\mathcal{C}} \| P_A^n\right) &= \sum_{a^n \in \mathcal{C}} \frac{1}{|\mathcal{C}|} \log_2 \frac{\frac{1}{|\mathcal{C}|}}{P_{\mathsf{A}}^n(a^n)} && (4.13)\\
&= -\log_2 |\mathcal{C}| + \frac{1}{|\mathcal{C}|} \sum_{a^n \in \mathcal{C}} \log_2 \frac{1}{P_{\mathsf{A}}^n(a^n)} && (4.14)\\
&= -\log_2 |\mathcal{C}| + \frac{1}{|\mathcal{C}|} \sum_{a^n \in \mathcal{C}} \iota\left(P_{\mathsf{A}}^n(a^n)\right) && (4.15)\\
&= -\log_2 |\mathcal{C}| + \frac{1}{|\mathcal{C}|} \sum_{a^n \in \mathcal{C}} \sum_{i=1}^{n} \iota\left(P_{\mathsf{A}}(a_i)\right). && (4.16)
\end{aligned}$$

From this form we can write the problem of finding a code book with *fixed* cardinality M of least divergence as follows:

$$\hat{\mathcal{C}}_M = \underset{\substack{\mathcal{C} \subseteq \mathcal{A}^n \\ |\mathcal{C}| = M}}{\operatorname{argmin}} \mathbb{D}\left(P_{\tilde{\mathsf{A}}^n} \| P_{\mathsf{A}}^n\right) \tag{4.17}$$

$$= \operatorname*{argmin}_{\substack{\mathcal{C}\subseteq\mathcal{A}^n \\ |\mathcal{C}|=M}} \sum_{a^n\in\mathcal{C}} \iota\left(P_{\mathsf{A}}^n(a^n)\right) \tag{4.18}$$

where equality in (4.18) holds because we shifted the objective by $\log_2|\mathcal{C}|$ and scaled it by $|\mathcal{C}|$. Problem (4.18) is solved in [19] by selecting M code words with the least self-information.

Example 4.1. Consider a binary alphabet $\mathcal{A}=\{0,1\}$ with $p=P_{\mathsf{A}}(1)=1-P_{\mathsf{A}}(0)$ and $p<1/2$. Since $p<1/2$, $P_{\mathsf{A}}^n(a^n)$ is monotonically decreasing in the Hamming weight of a^n. Consequently, the code book construction should include the all-zero code word. Next, code words with a single one and $n-1$ zeros are included, and so on. It follows that the probability of the letter 1 in the code book $P_{\bar{\mathsf{A}}}(1)=p_{\mathcal{C}}$ grows monotonically in the code book size $|\mathcal{C}|$. It remains to determine the optimal code book size $|\mathcal{C}|$, which can be done by a line search [19].

In order find the optimal code book

$$\hat{\mathcal{C}} = \operatorname*{argmin}_{\mathcal{C}\subseteq\mathcal{A}^n} \mathbb{D}\left(P_{\tilde{\mathsf{A}}^n}\|P_{\mathsf{A}}^n\right) \tag{4.19}$$

$$= \operatorname*{argmin}_{M} \left(\operatorname*{argmin}_{\substack{\mathcal{C}\subseteq\mathcal{A}^n \\ |\mathcal{C}|=M}} \mathbb{D}\left(P_{\mathcal{C}}\|P_{\mathsf{A}}^n\right) \right) \tag{4.20}$$

we need to search through the solutions $\hat{\mathcal{C}}_M$ of (4.18) for different code book sizes around $M\approx 2^{n\mathbb{H}(P_{\mathsf{A}})}$ [19] which is not difficult.

4.2.2 Analysis

We now characterize the optimal code books.

Lemma 4.2. The best code book consists of all code words up to a certain self-information.

Proof: Suppose $\mathcal{C}$ consists of *all* words a^n with self-information $\iota\left(P_{\mathsf{A}}^n(a^n)\right)$ up to $\hat{I}$ and ℓ code words with self-information exactly I and $I>\hat{I}$. We split the code book into two disjoint parts $\mathcal{C}'$ and $\mathcal{C}''$ with all code words of self-information up to $\hat{I}$ and those ℓ code words with self-information I, respectively. We rewrite the divergence as

$$\mathbb{D}\left(P_{\mathcal{C}}\|P_{\mathsf{A}}^n\right) = -\log_2(|\mathcal{C}'|+|\mathcal{C}''|) + \frac{1}{|\mathcal{C}'|+|\mathcal{C}''|} \underbrace{\sum_{a^n\in\mathcal{C}'} \iota\left(P_{\mathsf{A}}^n(a^n)\right)}_{\bar{I}\cdot|\mathcal{C}'|} \tag{4.21}$$

$$+ \frac{1}{|\mathcal{C}'|+|\mathcal{C}''|} \sum_{a^n\in\mathcal{C}''} \underbrace{\iota\left(P_{\mathsf{A}}^n(a^n)\right)}_{I} \tag{4.22}$$

$$= -\log_2(|\mathcal{C}'| + \ell) + \frac{|\mathcal{C}'|}{|\mathcal{C}'| + \ell}\bar{I} + \frac{\ell}{|\mathcal{C}'| + \ell}I \tag{4.23}$$

where $\bar{I}$ is an average self-information and therefore $\bar{I} < I$. We define a function $\mathbb{D}\left(P_{\mathcal{C}} \| P_{\mathsf{A}}^n\right)'$ for which we assume that ℓ is a continuous variable. Note that $\mathbb{D}\left(P_{\mathcal{C}} \| P_{\mathsf{A}}^n\right)'$ and $\mathbb{D}\left(P_{\mathcal{C}} \| P_{\mathsf{A}}^n\right)$ are equal for integer ℓ. The first and second derivatives with respect to ℓ are

$$\frac{\partial}{\partial \ell}\mathbb{D}\left(P_{\mathcal{C}} \| P_{\mathsf{A}}^n\right)' = -\frac{1}{\ln(2)(|\mathcal{C}'| + \ell)} + \frac{|\mathcal{C}'|}{(|\mathcal{C}'| + \ell)^2}\Delta I \tag{4.24}$$

$$\frac{\partial^2}{\partial \ell^2}\mathbb{D}\left(P_{\mathcal{C}} \| P_{\mathsf{A}}^n\right)' = \frac{1}{\ln(2)(|\mathcal{C}'| + \ell)^2} - 2\Delta I\frac{|\mathcal{C}'|}{(|\mathcal{C}'| + \ell)^3} \tag{4.25}$$

with $\Delta I = I - \bar{I} > 0$. The first derivative evaluates to zero only at

$$\ell_0 = |\mathcal{C}'|(\ln(2)\Delta I - 1) \tag{4.26}$$

which means that there is only one extreme point. Note that ℓ_0 can be negative but is larger than $-|\mathcal{C}'|$. The second derivative at ℓ_0 evaluates to

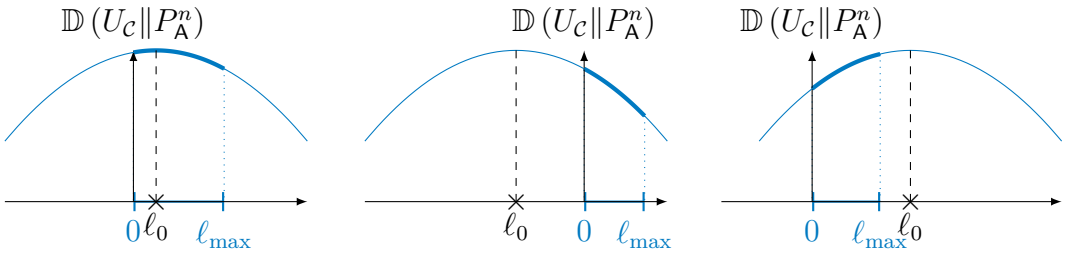
$$\frac{\partial^2}{\partial \ell^2}\mathbb{D}\left(P_{\mathcal{C}} \| P_{\mathsf{A}}^n\right)\Big|_{\ell=\ell_0} = -\frac{1}{\ln(2)^3|\mathcal{C}'|^2\Delta I^2} \tag{4.27}$$

which is negative and therefore this auxiliary divergence has one turning point at ℓ_0 that is a maximum.

We look for the integer $\hat{\ell} \in \{0, 1, \ldots, \ell_{\max}\}$ that minimizes $\mathbb{D}\left(P_{\mathcal{C}} \| P_{\mathsf{A}}^n\right)$ where $\ell_{\max}$ is the number of length n code words that have self-information I. We need to distinguish three cases.

- ▷ $\ell_0 \in [0, \ell_{\max}]$:
  $\mathbb{D}\left(P_{\mathcal{C}} \| P_{\mathsf{A}}^n\right)$ increases on $\{0, \ldots, \lfloor \ell_0 \rfloor\}$ and decreases on $\{\lceil \ell_0 \rceil, \ldots, \ell_{\max}\}$.
- ▷ $\ell_0 < 0$:
  $\mathbb{D}\left(P_{\mathcal{C}} \| P_{\mathsf{A}}^n\right)$ is monotonically increasing on $\{0, 1, \ldots, \ell_{\max}\}$ since the first derivative evaluates to zero for one point only and this turning point is a maximum.
- ▷ $\ell_0 > \ell_{\max}$:
  $\mathbb{D}\left(P_{\mathcal{C}} \| P_{\mathsf{A}}^n\right)$ is monotonically decreasing on $\{0, 1, \ldots, \ell_{\max}\}$ since the first derivative evaluates to zero for one point only and this turning point is a maximum.

All cases are depicted in Fig. 4.2 and in all cases we have $\hat{\ell} = 0$ or $\hat{\ell} = \ell_{\max}$. Thus, for code books of size $|\mathcal{C}| \in \{|\mathcal{C}'|, \ldots, |\mathcal{C}'| + \ell_{\max}\}$ the *minimal* divergence code book is a code book containing all code words up to $\hat{I}$ or I. Since this holds for every self-information $\hat{I}$, the proof is complete.



(a) $\ell_0 \in [0, \ell_{\max}]$ (b) $\ell_0 < 0$ (c) $\ell_0 > \ell_{\max}$

Figure 4.2: Three cases showing how ℓ_0 can be located in the interval $[0, \ell_{\max}]$.

■

Note that this proof is valid for any alphabet $\mathcal{A}$. It turns out that the optimal *binary* code book is the union of κ type sets with the lowest weight. While it is rather intuitive that we choose type sets with lowest weight, it is surprising that we would use only *complete* type sets. For an output of n binary symbols there are thus only $n+1$ optimal code books for all target distributions with $P_{\mathsf{A}}(1) < 0.5$. This result seems surprising, as in the binary case the average typeset cardinality grows exponentially in the block length n while the number of type sets grows only linearly with n (cf. Lemma 2.2). In Fig. 4.3 we draw the divergence of a probability induced by a code book versus a target probability. Small inverted peaks appear in the graph where the code word type changes. Those points are highlighted with gray dashed lines.

From this point on we consider only binary output alphabets, i.e., $\mathcal{A} = \{0, 1\}$ and we consider only code books $\mathcal{C}$ that are unions of type sets. For binary sequences, we write $\mathcal{C}_\kappa$ for the union of all type sets up to weight κ, i.e.,

$$|\mathcal{C}_\kappa| = \sum_{i=0}^{\kappa} \binom{n}{i}$$

which is sometimes called a *Hamming Ball*.

The probability $p_{\mathcal{C}_\kappa}$ to draw a 1 from the code book $\mathcal{C}_\kappa$ is

$$p_{\mathcal{C}_\kappa} = \frac{\sum_{i=0}^{\kappa} \binom{n}{i} i}{\sum_{i=0}^{\kappa} \binom{n}{i} n} \tag{4.28}$$

which corresponds to the probability of 1 of the respective letter distribution. $p_{\mathcal{C}_\kappa}$ increases monotonically in κ.

Lemma 4.3. For every positive integer n and every non-negative integer κ, $\kappa < n/2$, we have

$$0 \leq \frac{\kappa}{n} - p_{\mathcal{C}_\kappa} \leq \frac{1 - \kappa/n}{n(1 - 2k/n)} + \frac{1}{2n^2(1 - 2\kappa/n)^2}. \tag{4.29}$$

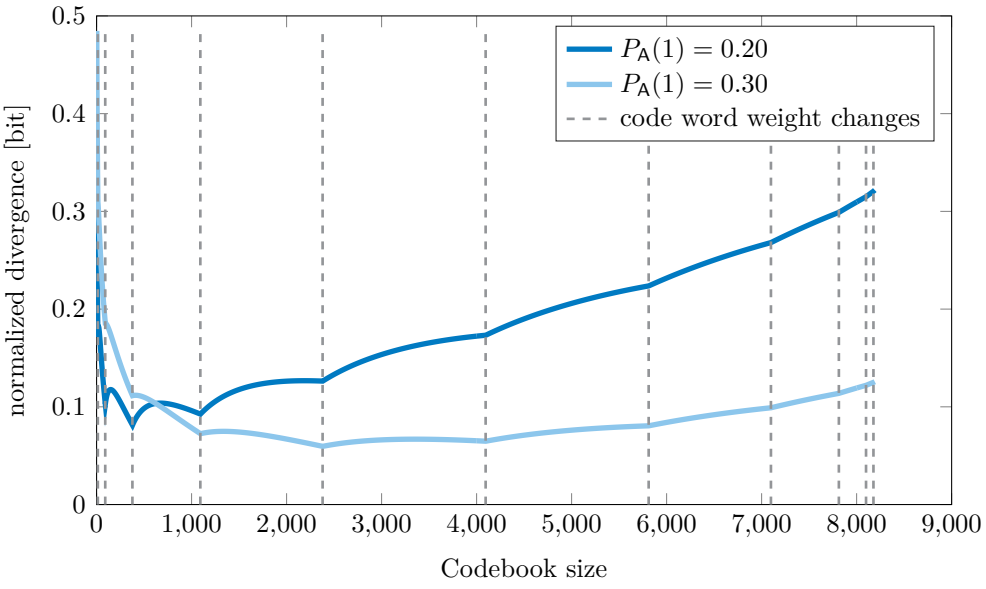


Figure 4.3: Normalized divergence of code book to binary target distribution vs. code book size for two target distributions. The dashed lines indicate where the weight of added code words changes. The code word length is $n = 13$.

Proof: Since $p_{\mathcal{C}_\kappa}$ is the average weight of the code words in $\mathcal{C}_\kappa$, the lower bound follows immediately. For the upper bound, we write

$$p_{\mathcal{C}_\kappa} = \frac{\sum_{i=0}^{\kappa} \binom{n}{i} i}{\sum_{j=0}^{\kappa} \binom{n}{j} n} = \frac{1}{2} - \frac{\sum_{i=0}^{\kappa} \binom{n}{i} (\frac{n}{2} - i)}{\sum_{j=0}^{\kappa} \binom{n}{j} n}. \tag{4.30}$$

Using Lemma 2.5, we can simplify this to

$$\begin{aligned} p_{\mathcal{C}_\kappa} &= \frac{1}{2} - \frac{\frac{\kappa+1}{2}\binom{n}{\kappa+1}}{\sum_{j=0}^{\kappa} \binom{n}{j} n} = \frac{1}{2} - \frac{\frac{n-\kappa}{2}\binom{n}{\kappa}}{\sum_{j=0}^{\kappa} \binom{n}{j} n} \\ &= \frac{1}{2} - \left(\frac{1}{2} - \frac{\kappa}{2n}\right) \frac{\binom{n}{\kappa}}{\sum_{j=0}^{\kappa} \binom{n}{j}}. \end{aligned} \tag{4.31}$$

Let $q := \kappa/n < \frac{1}{2}$ so that $\kappa = nq$. We insert the lower bound of Lemma 2.4 to obtain

$$\begin{aligned} p_{\mathcal{C}_\kappa} &= \frac{1}{2} - \left(\frac{1}{2} - \frac{q}{2}\right) \frac{\binom{n}{nq}}{\sum_{j=0}^{nq} \binom{n}{j}} \\ &\geq \frac{1}{2} - \left(\frac{1}{2} - \frac{q}{2}\right) \frac{1 - 2q + 1/n}{1 - q + 1/n} \left(1 + \frac{1}{n(1-2q)^2}\right) \end{aligned}$$

$$\begin{aligned}
&\geq \frac{1}{2} - \left(\frac{1}{2} - \frac{q}{2}\right) \frac{1-2q+1/n}{1-q} \left(1 + \frac{1}{n(1-2q)^2}\right) \\
&= q - \frac{1-q}{n(1-2q)} - \frac{1}{2n^2(1-2q)^2}
\end{aligned} \tag{4.32}$$

which establishes the upper bound of (4.29). ■

We next show that the optimal code book leads to a divergence that grows at least logarithmically in n. Let

$$\hat{\kappa} = \hat{\kappa}(n) := \underset{\kappa \in \{0,\ldots,n\}}{\operatorname{argmin}} \mathbb{D}\left(U_{\mathcal{C}_\kappa} \| P_{\mathsf{A}}^n\right) \tag{4.33}$$

for a given n and a given target distribution P_{A}. With (4.10) we have

$$\mathbb{D}\left(U_{\mathcal{C}_{\hat{\kappa}}} \| P_{\mathsf{A}}^n\right) = \log_2 \frac{1}{|\mathcal{C}_{\hat{\kappa}}|} + n\mathbb{H}\left(p_{\mathcal{C}_{\hat{\kappa}}}\right) + n\mathbb{D}\left(p_{\mathcal{C}_{\hat{\kappa}}} \| p\right). \tag{4.34}$$

In Sec. 4.3.2 we show that the unnormalized divergence of a suboptimal DM grows logarithmically in n. Thus, the unnormalized divergence of the optimal code book cannot grow faster than logarithmically in n, and we have $\mathbb{D}\left(p_{\mathcal{C}_{\hat{\kappa}}} \| p\right) \to 0$ with $n \to \infty$. Pinsker's inequality thus implies $|p_{\mathcal{C}_{\hat{\kappa}}} - p| \to 0$. From Lemma 4.3 we know that $|p_{\mathcal{C}_{\hat{\kappa}}} - \hat{\kappa}/n| \to 0$ and hence $|\hat{\kappa}/n - p| \to 0$ by the triangle inequality.

For every κ/n, $\kappa/n < \frac{1}{2}$, we obtain the following upper bound from Lemmas 2.2 and 2.4:

$$|\mathcal{C}_\kappa| \leq \binom{n}{\kappa} \frac{n-\kappa+1}{n-2\kappa+1} \leq \frac{2^{n\mathbb{H}(\kappa/n)}(n-\kappa+1)}{\sqrt{2\pi n \frac{\kappa}{n}\frac{n-\kappa}{n}}(n-2\kappa+1)}. \tag{4.35}$$

Consequently, for any code book with $\kappa/n < \frac{1}{2}$ we have

$$\begin{aligned}
&\mathbb{D}\left(U_{\mathcal{C}_\kappa} \| P_{\mathsf{A}}^n\right) \\
&\geq \log_2 \frac{1}{|\mathcal{C}_\kappa|} + n\mathbb{H}\left(p_{\mathcal{C}_\kappa}\right) \\
&\geq -\log_2 \left(\frac{2^{n\mathbb{H}(\kappa/n)}(n-\kappa+1)}{\sqrt{2\pi n \frac{\kappa}{n}\frac{n-\kappa}{n}}(n-2\kappa+1)} \right) + n\mathbb{H}\left(p_{\mathcal{C}_\kappa}\right) \\
&= \frac{1}{2}\log_2 n - n\left(\mathbb{H}\left(\kappa/n\right) - \mathbb{H}\left(p_{\mathcal{C}_\kappa}\right)\right) \\
&\quad + \frac{1}{2}\log_2 \left(\frac{2\pi \frac{\kappa}{n}(1-\frac{\kappa}{n})(1-2\frac{\kappa}{n}+\frac{1}{n})^2}{(1-\frac{\kappa}{n}+\frac{1}{n})^2} \right).
\end{aligned} \tag{4.36}$$

For small n, $\hat{\kappa}/n$ may be greater than $1/2$; but since $\hat{\kappa}/n \to p < 1/2$, the above bound holds also for the optimal code book for n sufficiently large.

Now define $\epsilon(n) = \frac{1-\hat{\kappa}/n}{n(1-2\hat{\kappa}/n)} + \frac{1}{2n^2(1-2\hat{\kappa}/n)^2}$. Lemmas 2.1 and 4.3 give

$$n\mathbb{H}\left(\frac{\hat{\kappa}}{n}\right) - n\mathbb{H}\left(p_{\mathcal{C}_{\hat{\kappa}}}\right) \leq \left(\frac{1-\hat{\kappa}/n}{1-2\hat{\kappa}/n} + \frac{1}{2n(1-2\hat{\kappa}/n)^2}\right) \log_2 \frac{1-\frac{\hat{\kappa}}{n}+\epsilon(n)}{\frac{\hat{\kappa}}{n}-\epsilon(n)}. \tag{4.37}$$

Combining the results (4.36) and (4.37), we obtain a lower bound on the unnormalized divergence for n sufficiently large such that $\hat{q} := \hat{\kappa}/n$ remains smaller than $1/2$:

$$\begin{aligned}
&\mathbb{D}\left(U_{\mathcal{C}_{\hat{\kappa}}} \| P_{\mathsf{A}}^n\right) \\
&\geq \frac{1}{2}\log_2 n \\
&\quad - \left(\frac{1-\hat{q}}{1-2\hat{q}} + \frac{1}{2n(1-2\hat{q})^2}\right) \log_2 \frac{1-\hat{q}+\epsilon(n)}{\hat{q}-\epsilon(n)} \\
&\quad + \frac{1}{2}\log_2 \left(\frac{2\pi\hat{q}(1-\hat{q})(1-2\hat{q}+\frac{1}{n})^2}{(1-\hat{q}+\frac{1}{n})^2}\right).
\end{aligned} \tag{4.38}$$

To show that the unnormalized divergence grows logarithmically in n, we evaluate (4.38) in the limit $n \to \infty$. Specifically, since $\hat{q} \to p$ and $\epsilon(n) \to 0$ with $n \to \infty$, we have

$$\liminf_{n\to\infty} \left(\mathbb{D}\left(U_{\mathcal{C}_{\hat{\kappa}}} \| P_{\mathsf{A}}^n\right) - \frac{1}{2}\log_2 n\right) \geq \frac{1}{2}\log_2 \left(\frac{2\pi p(1-2p)^2}{1-p}\right) - \frac{1-p}{(1-2p)} \log_2 \frac{1-p}{p}. \tag{4.39}$$

Thus, the divergence grows at least logarithmically in n.

4.2.3 Algorithm - Weight Functions

We subdivide the algorithm into three parts: weight functions, generating functions and ordering algorithms. The weight function $W(\cdot)$ assigns to each letter a in the alphabet $\mathcal{A}$ a non-negative, integer weight, i.e.,

$$W \colon \mathcal{A} \to \mathbb{N}_0. \tag{4.40}$$

The weight of a sequence is the sum over the per letter weights, i.e.,

$$W_n \colon \mathcal{A}^n \to \mathbb{N}_0 \tag{4.41}$$

$$W_n(a^n) = \sum_{i=1}^{n} W(a_i). \tag{4.42}$$

In Sec. 4.2.5 and Sec. 4.2.6 we discuss a sequential and a divide-and-conquer algorithm that can order sequences $a^n \in \mathcal{A}^n$ from lowest to highest weight. This ordering is in general not unique because two sequences may have the same weight, e.g., if they are permutations of each other. The order among sequences of the same weight is arbitrary. There are many ways to implement the ordering, e.g., using the divide-and-conquer principle [50] or sequential (enumerative) encoding [72]. The divide-and-conquer algorithm is well known as SM. The SM encoder maps an integer I to the I-th sequence of the ordered list. Hence, the SM encoder is a function

$$f_{\mathrm{SM}} : \{0, 1, \ldots, |\mathcal{A}|^n - 1\} \to \mathcal{A}^n. \tag{4.43}$$

If we restrict the domain of f_{SM} to the integers $\{0, 1, \ldots, M-1\}$, we refer to the image as

$$\mathcal{C}_{\mathrm{SM},M} = f_{\mathrm{SM}}(\{0, 1, \ldots, M-1\}). \tag{4.44}$$

Note that $\mathcal{C}_{\mathrm{SM},M}$ is a solution to the problem

$$\mathcal{C}_{\mathrm{SM},M} = \underset{\substack{\mathcal{C} \subseteq \mathcal{A}^n \\ |\mathcal{C}| = M}}{\operatorname{argmin}} \sum_{a^n \in \mathcal{C}} \sum_{i=1}^{n} W(a_i) \tag{4.45}$$

which means that SM finds the set $\hat{\mathcal{C}}$ of M sequences a^n of smallest weight $\sum_{i=1}^{n} W(a_i)$.

Interface All ordering algorithms require as inputs the code book cardinality M, the output length n, and the weight function W. We consider a binary input DM, so we choose $M = 2^k$ where k is the input block length in bits. The input bits are interpreted as an unsigned integer in the range of $\{0, \ldots, 2^k - 1\}$. The SM output corresponds to the output of a DM

$$f_{\mathrm{SM},k} : \{0, 1\}^k \to \mathcal{A}^n. \tag{4.46}$$

Divergence Optimal Weight Functions

Proposition 4.4. A minimum divergence f2f length DM with a target output probability P_{A} is a shell mapper with weight function

$$\hat{W}(a) = \iota\left(P_{\mathsf{A}}(a)\right) \tag{4.47}$$

Proof. The SM algorithm solves problem (4.45). When we use the *self-information* as a weight function, we solve problem (4.18). With a search over the input length, we can find the best DM independent of the code book size. ■

Example 4.2. Dyadic distributions have the form

$$P_{\mathsf{A}}(a) = 2^{-\ell_a} \tag{4.48}$$

where ℓ_a is a positive integer for all $a \in \text{supp}(P_\mathsf{A})$. The weight function (4.47) is

$$W(a) = \ell_a \quad a \in \text{supp}(P_\mathsf{A}). \tag{4.49}$$

Remark: A non-negative, integer weight function is desirable for implementation. Weight functions constructed with (4.47) do not generally have integer $\hat{W}(a)$. In the following, we show which practically relevant distributions also yield non-negative integer valued weight functions with (4.47).

Proposition 4.5. Consider a finite support discrete distribution that can be expressed as

$$P_\mathsf{A}(a) = \frac{e^{-v\Omega(a)}}{\sum_{\xi \in \text{supp}(P_\mathsf{A})} e^{-v\Omega(\xi)}}, \quad \forall a \in \text{supp}(P_\mathsf{A}) \tag{4.50}$$

with v is positive and Ω is any function

$$\Omega : \text{supp}(P_\mathsf{A}) \to \mathbb{N}_0. \tag{4.51}$$

Then Ω is a non-negative integer weight function.

Proof. Inserting (4.50) into (4.47) we obtain

$$\hat{W}(a) = v\Omega(a)\log_2(e) + \log_2 \sum_{\xi \in \text{supp}(P_\mathsf{A})} e^{-v\Omega(\xi)}.$$

Any translation and positive scaling can be applied on the objective function without changing the code book. We obtain the integer weight function

$$W(a) = \Omega(a) \quad a \in \text{supp}(P_\mathsf{A}). \tag{4.52}$$

■

Example 4.3. The half Maxwell-Boltzmann (MB) distribution is defined as

$$P_\mathsf{A}(a) = \frac{e^{-va^2}}{\sum_{\xi \in \text{supp}(P_\mathsf{A})} e^{-v\xi^2}} \tag{4.53}$$

with $\text{supp}(P_\mathsf{A}) = \{1, 3, 5, \ldots, 2\eta - 1\} = \mathcal{A}$ and positive $v \in \mathbb{R}^+$ and $\eta \in \mathbb{N}$. Comparing with (4.50) we identify the weight function

$$W(a) = a^2 \quad a \in \text{supp}(P_\mathsf{A}) \tag{4.54}$$

which corresponds to the energy of a constellation point. For implementation we may want to use

$$W(a) = (a^2 - 1)/8 \quad a \in \text{supp}(P_\mathsf{A}) \tag{4.55}$$

because we decrease the maximum weight that we need to track and have integer weights.

Corollary 4.6. We obtain sequences of least power by minimizing divergence to MB distributions.

This result has a special beauty. One finds that MB distributions are close to optimal for maximizing the single letter mutual information on discrete signal points for the AWGN channel. If we now minimize the informational divergence of our f2f length DM to a memoryless source with MB distribution, we find that sequences of least energy accomplish this goal.

The weight function (4.54) is independent of the parameter v. Consequently, according to (4.18) a shell mapper with this weight function implements a minimum divergence DM with fixed code book size 2^m for *all* half MB distributions. If our goal is to approximate (half) MB distributions and to vary the rate, we fix the weight function and output length and choose the input length m accordingly. This significantly facilitates rate adaptation. Rate adaptation is also easy for distribution families we obtain from (4.50) for a fixed function Ω and varying v.

Letter Frequencies $P_{\bar{\mathsf{A}}}$

A soft-input soft-output decoder requires the letter distribution (4.11) in order to calculate the priors on the constellation symbols [11, Sec. VI-B]. The letter distribution depends on both the weight function and how to order sequences of equal weight. Fischer suggests in [73] an algorithm to calculate the letter distribution. The algorithm uses the *partial histogram*, i.e., the letter distribution for code books that consist of *all* sequences up to a certain weight.

4.2.4 Algorithm - Generating Functions

Generating functions build the combinatorial core for some algorithms that order sequences according to a weight function. A general introduction to generating functions can be found in [74]. We use generating functions to reduce sequences to their weight and to track how many code words of an arbitrary weight a code book contains [75]. Furthermore, generating functions are useful to compute how code word weights evolve if we increase the code word length. The generating function for all code words of length n of a code book $\mathcal{C}$ and cost function $W(\cdot)$ is the polynomial

$$G_n(z) = \sum_{a^n \in \mathcal{C}} z^{W_n(a^n)} \tag{4.56}$$

where the subscript n of G_n indicates the length of the considered functions and the weight function is left implicit.

Example 4.4. Let $\mathcal{A} = \{\alpha, \beta, \gamma, \delta\}$ and $W(\alpha) = 0$, $W(\beta) = W(\gamma) = 1$ and $W(\delta) = 3$.

Weight	Sequences	Occurrences
0	$\alpha\alpha$	1
1	$\alpha\beta$, $\alpha\gamma$, $\beta\alpha$, $\gamma\alpha$	4
2	$\beta\beta, \beta\gamma, \gamma\beta, \gamma\gamma$	4
3	$\alpha\delta$, $\delta\alpha$	2
4	$\beta\delta$, $\gamma\delta$, $\delta\beta$, $\delta\gamma$	4
5	-	0
6	$\delta\delta$	1

Table 4.1: Sequences of Example 4.5 ordered by weight.

The generating function for length one words, i.e., $\mathcal{C} = \mathcal{A}$ is

$$G_1(z) = 1 + 2z^1 + z^3. \tag{4.57}$$

An alternative representation is

$$G_n(z) = \sum_{w \in \mathcal{W}_n} b_w z^w \tag{4.58}$$

where $\mathcal{W}_n$ is the set of weights that a length n sequence may have assuming some weight function $W(\cdot)$. We denote the operation of extracting the coefficient b_w of z^w by

$$[z^w]G(z) = b_w. \tag{4.59}$$

We assume integer weights for ease of implementation. The generating function for all length 2 code words $G_2(z)$ can be calculated from the length 1 generating function $G_1(z)$ by

$$[z^w]G_2(z) = \sum_{k=0}^{w} [z^k]G_1(z) \cdot [z^{w-k}]G_1(z) \tag{4.60}$$

which is a convolution of the coefficients. This corresponds to $G_2(z) = G_1(z) \cdot G_1(z)$ or in general

$$G_n(z) = (G_1(z))^n. \tag{4.61}$$

Example 4.5. Consider the generating function G_1 of Example 4.4. In Table 4.1 we order all sequences $\mathcal{A}^2$ according to their weight. The calculation according to (4.61) is straight forward and gives

$$G_2(z) = (1 + 2z^1 + z^3)^2 = 1 + 4z^1 + 4z^2 + 2z^3 + 4z^4 + z^6. \tag{4.62}$$

4.2.5 Algorithm - Enumerative Coding

The rules for ordering sequences according to a per letter weight function with enumerative coding are

1. a sequence $\boldsymbol{c}$ comes before a sequence $\boldsymbol{d}$ if the weight of $\boldsymbol{c}$ is smaller than the weight of $\boldsymbol{d}$.

2. if the sequences $\boldsymbol{c}$ and $\boldsymbol{d}$ have the same weight, then $\boldsymbol{c}$ comes before $\boldsymbol{d}$ if $\boldsymbol{c}$ is first in lexicographical ordering.

Decoding

The decoder $f_{\mathrm{SM},k}^{-1}$ receives a sequence a^n and outputs the binary encoded index N in the ordered list of code words. The encoding and decoding algorithms reflect the ordering rules. Both require two steps for encoding and decoding. In the first step we find the number N_1 of code words of length n that have a smaller weight than the present sequence. In the second step we sequentially process the sequence and determine the number N_2 of code words with same weight that come first in lexicographical ordering. The decoded index is $N = N_1 + N_2$.

We use generating functions to determine the number of sequences that have weight less than the sequence a^n, i.e., we have

$$N_1 = \sum_{w=0}^{W_n(a^n)-1} [z^w] G_n(z). \tag{4.63}$$

Recall that the coefficient $[z^w]G_n(z)$ corresponds to the number of sequences of length n with weight w according to the weight function W.

The number N_2 of sequences with weight $W(a^n)$ that come first in lexicographical ordering can be determined using equation (2.47). The number of sequences with prefix b^i, total weight $W(a^n)$ and length n is the number of sequences that have weight $W(a^n) - W(b^i)$ and length $n-i$, i.e., we have

$$n_{b^i}(\mathcal{C}_w) = [z^{W_n(a^n)-W_i(b^i)}] G_{n-i}(z). \tag{4.64}$$

Encoding

The encoder finds the N-th entry in a list ordered according to the above rules. First, the encoder needs to find the total weight of the sequence, i.e., it finds w such that

$$\sum_{\omega=0}^{w-1} [z^\omega] G_n(z) \leq N < \sum_{\omega=0}^{w} [z^\omega] G_n(z). \tag{4.65}$$

We then run the encoder according to Algorithm 2.1 with $n_{b^i}(\mathcal{C}_w)$ defined in (4.64) with $W_n(a^n) = w$ and index $N_2 = N - N_1 = N - \sum_{\omega=0}^{w-1}[z^\omega]G_n(z)$.

4.2.6 Algorithm - Divide-and-Conquer Enumerative Coding

For the divide-and-conquer approach we consider the following ordering that is also described in [50, 76]. Consider two length n sequences $\boldsymbol{c}$ and $\boldsymbol{d}$. The sequence $\boldsymbol{c}$ is first in order if

1. The weight $W_n(\boldsymbol{c})$ is smaller than the weight $W_n(\boldsymbol{d})$.
2. The weights are the same and the first half of $\boldsymbol{c}$ comes first by applying the ordering recursively on the first part.
3. The first halves are identical and the second half of $\boldsymbol{c}$ comes first by applying the ordering rule recursively.

For the implementation below, we require n to be a power of 2. However, the divide-and-conquer approach can also be applied when splitting a sequences into sequences of different size. If we choose that the length of the first part of $\boldsymbol{c}$ and $\boldsymbol{d}$ is always 1, we recover the sequential approach of enumerative coding.

Decoding

The decoder maps a sequence a^n to the index $N(a^n)$. The index is the sum of the number $\hat{N}$ of sequences that come first because of a lower weight and the number $\tilde{N}(a^n)$ of sequences that come first because of a recursively defined ordering. The first step of decoding is the same as for the sequential algorithm, i.e., it finds the number $\hat{N}$ of sequences with weight below $W_n(a^n)$. We have

$$\hat{N} = \sum_{w=0}^{W_n(a^n)-1} [z^w]G_n(z). \tag{4.66}$$

The ordering is defined recursively and we divide the sequence into subsequences until dthere are only sequences of length 1 left. For length 1 sequences it is easy to remember the number $\tilde{N}(a)$ of sequences that have same weight as a and come first in order. It is usually 0 unless there are multiple letters with the same weight. In this case we need to be aware of an ordering among those letters and we can store $\tilde{I}(a)$ in a LUT.

In the next step, the algorithm continues with the composition (conquer) of two simple problems, i.e., to compute $N([\boldsymbol{c}_1, \boldsymbol{c}_2])$, when $N(\boldsymbol{c}_1)$ and $N(\boldsymbol{c}_2)$ are known. The conquer operation of two subsequences $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, both of length ν, follows again closely to the ordering rules. The second rule can be split up into two parts:

N_1: number of sequences that have smaller weight in the first part

N_2: number of sequences that have the same weight in the first part and the first part comes first in order.

The algorithm calculates N_1 with the knowledge of the weights of the first and second halves, $W_\nu(\boldsymbol{c}_1)$ and $W_\nu(\boldsymbol{c}_2)$, respectively. We have

$$N_1 = \sum_{w=0}^{W_\nu(\boldsymbol{c}_1)-1} [z^w]G_\nu(z) \cdot [z^{W_\nu(\boldsymbol{c}_1)+W_\nu(\boldsymbol{c}_2)-w}]G_\nu(z). \tag{4.67}$$

For calculating N_2 the algorithm must know the order of the first half $\tilde{N}(\boldsymbol{c}_1)$. This was computed in the last step, i.e., we have

$$N_2 = \tilde{N}(\boldsymbol{c}_1) \cdot [z^{W_\nu(\boldsymbol{c}_2)}]G_\nu(z). \tag{4.68}$$

We know how many sequences of weight $W_\nu(\boldsymbol{c}_1)$ come below $\boldsymbol{c}_1$ by solving the subproblem $\tilde{N}(\boldsymbol{c}_1)$ and we multiply this number with the number of possible combinations in the second half, i.e., the number of sequences with weight $W_\nu(\boldsymbol{c}_2)$ of length ν.

The number N_3 of sequences that come first according to the third rule is equal to the number of sequences that have equal weight as the second part and come first, i.e., we have

$$N_3 = \tilde{N}(\boldsymbol{c}_2). \tag{4.69}$$

The position $\tilde{N}([\boldsymbol{c}_1, \boldsymbol{c}_2])$ of the whole sequence $[\boldsymbol{c}_1, \boldsymbol{c}_2]$ within the sequences of same weight is

$$\tilde{N}([\boldsymbol{c}_1, \boldsymbol{c}_2]) = N_1 + N_2 + N_3. \tag{4.70}$$

Example 4.6. Consider the sequence $[\alpha, \gamma, \beta, \delta]$, the weight function $W(\alpha) = 0$, $w(\beta) = W(\gamma) = 1$, $W(\delta) = 3$ from example 4.5, and suppose that β comes before γ in order. For $G_1(z)$, $G_2(z)$ and $G_4(z)$ we have

$$G_1(z) = 1 + 2z + z^3 \tag{4.71}$$
$$G_2(z) = 1 + 4z + 4z^2 + 2z^3 + 4z^4 + z^6 \tag{4.72}$$
$$G_4(z) = 1 + 8z + 24z^2 + 36z^3 + 40z^4 + 48z^5 + 38z^6 + 24z^7 + 24z^8 + 4z^9 + 8z^{10} + z^{12}. \tag{4.73}$$

The sequence weight is 5, and there are $1 + 8 + 24 + 36 + 40 = 109$ sequences of length 4 with weight 0 to 4. The algorithm starts with the length 1 sequences and we obtain

$$\tilde{N}(\alpha) = 0 \tag{4.74}$$
$$\tilde{N}(\beta) = 0 \tag{4.75}$$
$$\tilde{N}(\gamma) = 1 \tag{4.76}$$
$$\tilde{N}(\delta) = 0 \tag{4.77}$$

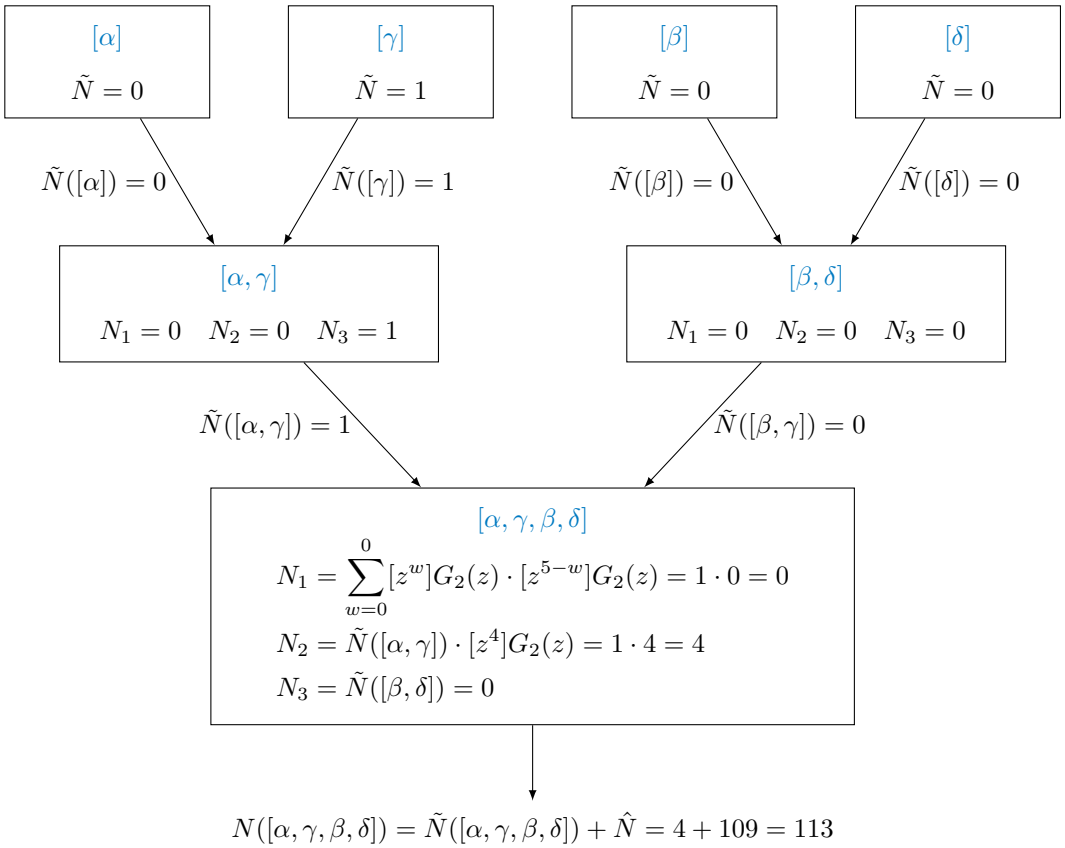


Figure 4.4: Decoding the sequence $[\alpha, \gamma, \beta, \delta]$ with its subproblems in a tree.

because only γ has a predecessor with the same weight.

The first summand N_1 of $\tilde{N}([\alpha,\gamma])$ is zero because the sum of (4.67) does not contain summands. As $\tilde{N}(\alpha)$ is zero, also N_2 of (4.68) is zero. N_3 evaluates to 1 and therefore $\tilde{N}([\alpha,\gamma])$ evaluates to 1. For $\tilde{N}([\beta,\delta])$, N_1 is zero. The sum of (4.67) has only the summand $[z^0]G_1(z)\cdot[z^4]G_1(z)$ which is 0 because there is no letter that has weight 4. As $\tilde{N}(\beta)$ and $\tilde{N}(\delta)$ are zero, N_2 and N_3 evaluate to 0. $\tilde{N}([\alpha,\gamma])$ evaluates to 0. In the last step we determine $\tilde{N}([\alpha,\gamma,\beta,\delta])$. The sum again consists of one summand, i.e., $[z^0]G_2(z)\cdot[z^5]G_2(z)$, which is 0 again, because the coefficient of z^5 does not exist. We cannot find a length 2 sequence that has weight 5 with the respective weight function. N_2 evaluates to 4 because there are 4 sequences of length 2 that have weight 4. We can easily check that those are all sequences that βs and γs on all positions. N_3 is again 0. We obtain $N([\alpha,\gamma,\beta,\delta]) = \tilde{N}([\alpha,\gamma,\beta,\delta]) + \hat{N}([\alpha,\gamma,\beta,\delta]) = 4 + 109 = 113$. All steps are summarized in Fig. 4.4.

Encoding

The encoder finds the N-th entry a^n of the ordered list. Like in the sequential approach, the encoder first finds the total weight of the sequence, i.e., it finds w such that

$$\sum_{\omega=0}^{w-1}[z^\omega]G_n(z) \le N < \sum_{\omega=0}^{w}[z^\omega]G_n(z). \tag{4.78}$$

Analogously to the decoding, we can split the index into $N = \hat{N}(a^n) + \tilde{N}(a^n)$, where $\hat{N}(a^n)$ is the number of sequences that come below a^n because they have lower weight and $\tilde{N}(a^n)$ is the lexicographical ordering among sequences of same weight. The recursive part of the algorithm breaks information about the index $\tilde{N}(\boldsymbol{c})$ within sequences of same weight w and length n into the same information about the first and second half of the sequence $\boldsymbol{c}$. The recursion ends for sequences of length 1, where we can use a LUT. For a detailed description of the recursive part, see Algorithm 4.1.

4.2.7 Code Trellis

The code trellis of a code book that contains all code words up to weight $W_{\max}$ was introduced in [77]. We define a trellis state as tuple (w,n') with $w \in \{0,\dots,W_{\max}\}$ and $n' \in \{0,\dots,n\}$; w tracks the weight and n the length of the code words. The start of the trellis is the state $(0,0)$. Two trellis states (w_1,n_1') and (w_2,n_2') are connected via the letter α if $n_2' = n_1' + 1$, $w_2 = w_1 + W(\alpha)$ and there exists a path from (w_1,n_1') to $(0,0)$. An example trellis can be found in Fig. 4.5.

Note that the trellis contains *all* code words up to weight $W_{\max}$. SMDM with a binary interface indexes only in rare cases all words up to weight $W_{\max}$, and usually only a subset. From the trellis of code words up to weight $W_{\max}$ we can easily construct the trellis of code words with weight exactly $W_{\max}$. We remove states and labels to states

Algorithm 4.1 Divide-and-Conquer Encoder: Recursion

Require: $\tilde{N}(\boldsymbol{c})$, w, n

End of recursion:

if $n == 1$ **then**

return letter of weight w that has $\tilde{N}(\boldsymbol{c})$ predecessors

end if

Recursion step:

$\nu = n/2$

find w_1 such that:

$$\sum_{\omega=0}^{w_1-1} [z^\omega]G_\nu(z) \cdot [z^{w-\omega}]G_\nu(z) \leq \tilde{N}(\boldsymbol{c}) < \sum_{\omega=0}^{w_1} [z^\omega]G_\nu(z) \cdot [z^{w-\omega}]G_\nu(z). \tag{4.79}$$

$w_2 = W(\boldsymbol{c}) - w_1$

$N_1 = \sum_{\omega=0}^{w_1-1} [z^\omega]G_\nu(z) \cdot [z^{w-\omega}]G_\nu(z)$

$\tilde{N}(\boldsymbol{c}_1) = \left\lfloor (\tilde{N}(\boldsymbol{c}) - N_1)/[z^{w_2}]G_\nu(z) \right\rfloor$

$N_2 \overset{(4.68)}{=} \tilde{N}(\boldsymbol{c}_1) \cdot [z^{w_2}]G_\nu(z)$

$N_3 = \tilde{N}(\boldsymbol{c}) - N_1 - N_2$

$\tilde{N}(\boldsymbol{c}_2) \overset{(4.69)}{=} N_3$

return [Recursion($\tilde{N}(\boldsymbol{c}_1)$, w_1, ν), Recursion($\tilde{N}(\boldsymbol{c}_2)$, w_2, ν)]

that do not have a path that end in the state $(W_{\max}, n)$. An example can be found in Fig. 4.6.

4.3 Constant Composition Distribution Matching

4.3.1 Codebook construction

Recall from (2.36) that the empirical distribution of a vector $\boldsymbol{c}$ of length n is defined as

$$P_{\bar{\mathsf{A}},\boldsymbol{c}}(a) = \frac{n_a(\boldsymbol{c})}{n} \tag{4.80}$$

where $n_a(\boldsymbol{c}) = |\{i : c_i = a\}|$ is the number of times symbol a appears in $\boldsymbol{c}$. The authors of [29, Sec. 2.1] call $P_{\bar{\mathsf{A}},\boldsymbol{c}}$ the *type* of $\boldsymbol{c}$. An n-type is a type based on a length n sequence. The type vector $\boldsymbol{t}_{\boldsymbol{c}}$ expresses how often each letter of the alphabet appears, i.e.,

$$\boldsymbol{t}_{\boldsymbol{c}} = [n_{\alpha_1}(\boldsymbol{c}), n_{\alpha_2}(\boldsymbol{c}), \ldots] \quad \forall \alpha_i \in \mathcal{A}. \tag{4.81}$$

A code book $\mathcal{C}_{\text{ccdm}} \subseteq \mathcal{A}^n$ is called a *constant composition code* if all code words are of the same type, i.e., $n_a(\boldsymbol{c})$ does not depend on the code word $\boldsymbol{c}$. We will write n_a in place

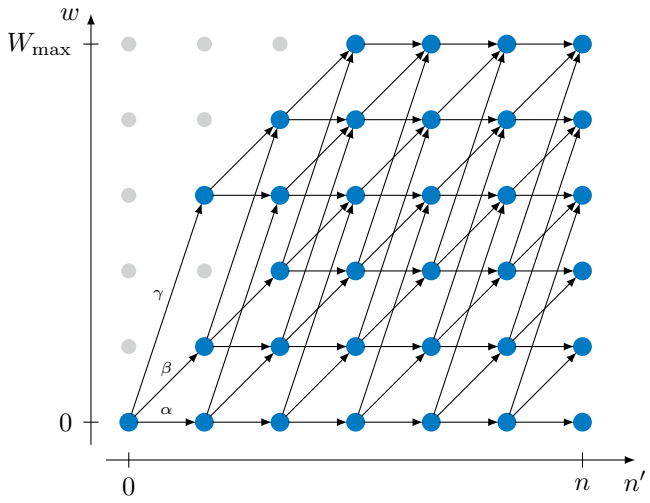


Figure 4.5: Trellis of a shell mapping code book with weight $W(\alpha) = 0, W(\beta) = 1, W(\gamma) = 3$, maximum weight $W_{\max} = 5$ and length $n = 6$.

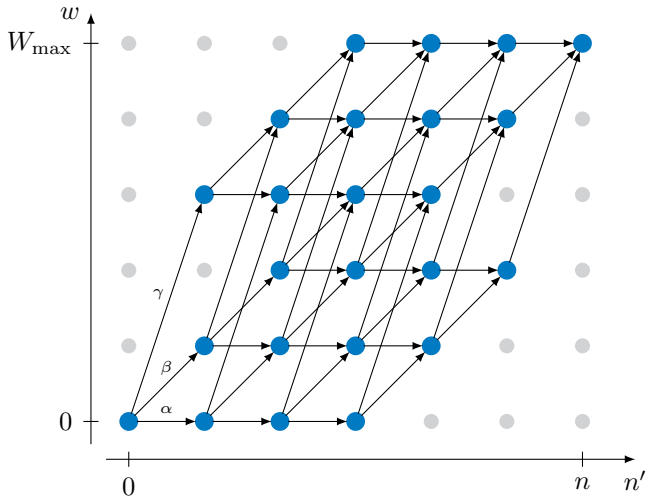


Figure 4.6: Trellis of a code book code words of weight $W_{\max} = 5$ only and length $n = 6$. The weight function is $W(\alpha) = 0, W(\beta) = 1, W(\gamma) = 3$.

of $n_a(\boldsymbol{c})$ for a constant composition code.

We use a constant composition code with $n_a \approx P_{\mathsf{A}}(a)n$ where P_{A} is the target probability. As all n_a need to be integers and add up to n, there are multiple possibilities to choose the n_a. We use the allocation that solves

$$P_{\bar{\mathsf{A}}} = \operatorname*{argmin}_{P_{\bar{\mathsf{A}}'}} \mathbb{D}\left(P_{\bar{\mathsf{A}}'} \| P_{\mathsf{A}}\right) \tag{4.82}$$

$$\text{s.t.: } P_{\bar{\mathsf{A}}'} \text{ is } n\text{-type.} \tag{4.83}$$

The solution of (4.82) can be found efficiently by [78, Algorithm 2]. This allocation provides a clear rule for choosing the n_a and is convenient for analysis. Suppose that the output length n is fixed and our system allows us to choose the input length k according to our target distribution. Let $\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}$ be the set of vectors of type $P_{\bar{\mathsf{A}}}$, i.e., we have

$$\mathcal{T}^n_{P_{\bar{\mathsf{A}}}} = \left\{ \boldsymbol{v} \;\middle|\; \boldsymbol{v} \in \mathcal{A}^n, \frac{n_a(\boldsymbol{v})}{n} = P_{\bar{\mathsf{A}}}(a) \quad \forall a \in \mathcal{A} \right\}. \tag{4.84}$$

For an invertible matcher we need at least as many code words as input blocks. Thus, the input block length must not exceed $\log_2 |\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}|$. We set the input length to $k = \lfloor \log_2 |\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}| \rfloor$ and we define the encoding function

$$f_{\text{ccdm}} : \{0,1\}^k \to \mathcal{T}^n_{P_{\bar{\mathsf{A}}}}. \tag{4.85}$$

The actual mapping f_{ccdm} can be implemented efficiently by arithmetic coding, as we will show in Sec. 4.3.6. We also detail how to use enumerative coding in Sec. 4.3.4 and Sec. 4.3.5. The constant composition code book is now given by the image of f_{ccdm}, i.e.,

$$\mathcal{C}_{\text{ccdm}} = f_{\text{ccdm}}(\{0,1\}^k). \tag{4.86}$$

Since f_{ccdm} is invertible, the code book size is $|\mathcal{C}_{\text{ccdm}}| = 2^k$. Note that the exact code book does not need to be known and the codebook may differ for different algorithms.

4.3.2 Analysis

We show that f_{ccdm} asymptotically achieves all rates satisfying (4.9). We can bound the input length k by

$$k = \left\lfloor \log_2 |\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}| \right\rfloor \geq \log_2 |\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}| - 1 \tag{4.87}$$

and

$$k = \left\lfloor \log_2 |\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}| \right\rfloor \leq \log_2 |\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}|. \tag{4.88}$$

By Lemma 2.3, we have

$$\frac{2^{n\mathbb{H}\left(P_{\bar{\mathsf{A}}}\right)}}{\sqrt{\frac{8^{q-1}}{n}\prod_{i=1}^{q}P_{\bar{\mathsf{A}}}(i)n}} \leq |\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}| \leq \frac{2^{n\mathbb{H}\left(P_{\bar{\mathsf{A}}}\right)}}{\sqrt{\frac{(2\pi)^{q-1}}{n}\prod_{i=1}^{q}P_{\bar{\mathsf{A}}}(i)n}} \tag{4.89}$$

where q enumerates the support of $P_{\bar{\mathsf{A}}}$ and according the definition of $P_{\bar{\mathsf{A}}}$ implies that $nP_{\bar{\mathsf{A}}}$ is always integer.

$$k \geq n\mathbb{H}\left(P_{\bar{\mathsf{A}}}\right) - \frac{3}{2}(q-1) + \frac{1}{2}\log_2\left(\frac{n}{\prod_{i=1}^{q}P_{\bar{\mathsf{A}}}(i)n}\right) - 1 \tag{4.90}$$

$$k \leq n\mathbb{H}\left(P_{\bar{\mathsf{A}}}\right) - \frac{q-1}{2}\log_2 2\pi + \frac{1}{2}\log_2\left(\frac{n}{\prod_{i=1}^{q}P_{\bar{\mathsf{A}}}(i)n}\right). \tag{4.91}$$

Recall that the matcher output distribution is $P_{\tilde{\mathsf{A}}^n}$. We have

$$\begin{aligned}
\mathbb{D}\left(P_{\tilde{\mathsf{A}}^n}\|P^n_{\mathsf{A}}\right) &= \sum_{a^n\in\mathcal{C}_{\mathrm{ccdm}}\subseteq\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}} 2^{-k}\log_2\frac{2^{-k}}{P^n_{\mathsf{A}}(a^n)}\frac{P^n_{\bar{\mathsf{A}}}(a^n)}{P^n_{\bar{\mathsf{A}}}(a^n)} \\
&= \mathbb{D}\left(P_{\tilde{\mathsf{A}}^n}\|P^n_{\bar{\mathsf{A}}}\right) + \sum_{a^n\in\mathcal{C}_{\mathrm{ccdm}}\subseteq\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}} 2^{-k}\log_2\frac{P^n_{\bar{\mathsf{A}}}(a^n)}{P^n_{\mathsf{A}}(a^n)} \\
&= \mathbb{D}\left(P_{\tilde{\mathsf{A}}^n}\|P^n_{\bar{\mathsf{A}}}\right) + |\mathcal{C}_{\mathrm{ccdm}}|2^{-k}\sum_{a\in\mathcal{A}} n_a\log_2\frac{P_{\bar{\mathsf{A}}}(a)}{P_{\mathsf{A}}(a)} \\
&= \underbrace{\mathbb{D}\left(P_{\tilde{\mathsf{A}}^n}\|P^n_{\bar{\mathsf{A}}}\right)}_{\text{Term 1}} + n\underbrace{\mathbb{D}\left(P_{\bar{\mathsf{A}}}\|P_{\mathsf{A}}\right)}_{\text{Term 2}}.
\end{aligned} \tag{4.92}$$

For Term 1 we obtain

$$\begin{aligned}
\mathbb{D}\left(P_{\tilde{\mathsf{A}}^n}\|P^n_{\bar{\mathsf{A}}}\right) &= \sum_{a^n\in\mathcal{C}_{\mathrm{ccdm}}\subseteq\mathcal{T}^n_{P_{\mathsf{A}}}} 2^{-k}\log_2\frac{2^{-k}}{\prod_{i\in\mathcal{A}}P_{\bar{\mathsf{A}}}(i)^{n_i}} \\
&= \sum_{\mathcal{C}_{\mathrm{ccdm}}} 2^{-k}\log_2\frac{2^{-m}}{2^{-n\mathbb{H}(\bar{\mathsf{A}})}} \\
&= n\mathbb{H}(\bar{\mathsf{A}}) - k.
\end{aligned} \tag{4.93}$$

We can bound Term 1 from above and below using (4.90) and (4.91) in (4.93)

$$\mathbb{D}\left(P_{\tilde{\mathsf{A}}^n}\|P^n_{\bar{\mathsf{A}}}\right) \leq \frac{3}{2}(q-1) + \frac{q-1}{2}\log_2(n) + \frac{1}{2}\log_2\left(\prod_{i=1}^{q}P_{\bar{\mathsf{A}}}(i)\right) + 1 \tag{4.94}$$

$$\mathbb{D}\left(P_{\tilde{\mathsf{A}}^n}\|P^n_{\bar{\mathsf{A}}}\right) \geq \frac{q-1}{2}\log_2 2\pi + \frac{q-1}{2}\log_2(n) + \frac{1}{2}\log_2\left(\prod_{i=1}^{q}P_{\bar{\mathsf{A}}}(i)\right). \tag{4.95}$$

Term 1 scales logarithmically with block length n and a pre-log factor of $(q-1)/2$.

Using (4.93) in (4.92) and dividing by n we have

$$\frac{\mathbb{D}\left(P_{\tilde{\mathsf{A}}^n}\|P_{\mathsf{A}}^n\right)}{n} = \mathbb{H}(\bar{\mathsf{A}}) - R + \mathbb{D}\left(P_{\bar{\mathsf{A}}}\|P_{\mathsf{A}}\right). \tag{4.96}$$

The choice (4.82) of $P_{\bar{\mathsf{A}}}$ minimizes the third term on the right-hand side of (4.96) and guarantees (see [78, Proposition 4]) that

$$\mathbb{D}\left(P_{\bar{\mathsf{A}}}\|P_{\mathsf{A}}\right) < \log_2\left(1 + \frac{q}{\min\limits_{a\in\operatorname{supp} P_{\mathsf{A}}} P_{\mathsf{A}}(a)n^2}\right). \tag{4.97}$$

If we fix the target distribution P_{A}, even $n\mathbb{D}\left(P_{\bar{\mathsf{A}}}\|P_{\mathsf{A}}\right)$ goes to zero if n goes to infinity. Consequently, the unnormalized divergence of (4.92) grows logarithmically in n.

We know that Term 2 vanishes as the block length approaches infinity, i.e., we have

$$\lim_{n\to\infty} \mathbb{D}\left(P_{\bar{\mathsf{A}}}\|P_{\mathsf{A}}\right) = 0. \tag{4.98}$$

We now relate the input and output lengths to understand the asymptotic behavior of the rate. For the rate, we obtain the upper and lower bounds

$$\begin{aligned} R = \frac{k}{n} &\overset{(4.88)}{\leq} \frac{\log_2\left|\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}\right|}{n} \\ &\overset{(4.89)}{\leq} \mathbb{H}\left(P_{\bar{\mathsf{A}}}\right) - \frac{(q-1)}{2n}\log_2 2\pi - \sum_{a\in\operatorname{supp} P_{\bar{\mathsf{A}}}} \frac{\log_2 P_{\bar{\mathsf{A}}}(a)}{2n} - \frac{\log_2(n)}{2n}(q-1) \end{aligned} \tag{4.99}$$

and

$$\begin{aligned} R = \frac{k}{n} &\overset{(4.87)}{\geq} \frac{\log_2\left|\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}\right|}{n} - \frac{1}{n} \\ &\overset{(4.89)}{\geq} \mathbb{H}\left(P_{\bar{\mathsf{A}}}\right) - \frac{(q-1)}{2n}\log_2 8 - \sum_{a\in\operatorname{supp} P_{\bar{\mathsf{A}}}} \frac{\log_2 P_{\bar{\mathsf{A}}}(a)}{2n} - \frac{q-1}{2n}\log_2(n) - \frac{1}{n} \end{aligned} \tag{4.100}$$

and in the asymptotic case

$$\lim_{n\to\infty} R = \mathbb{H}\left(P_{\bar{\mathsf{A}}}\right). \tag{4.101}$$

From (4.98) and [69, Proposition 6] we know that $\mathbb{H}\left(P_{\bar{\mathsf{A}}}\right) \to \mathbb{H}\left(\mathsf{A}\right)$, and by (4.98) and (4.101) in (4.96), the normalized divergence approaches zero for $n \to \infty$.

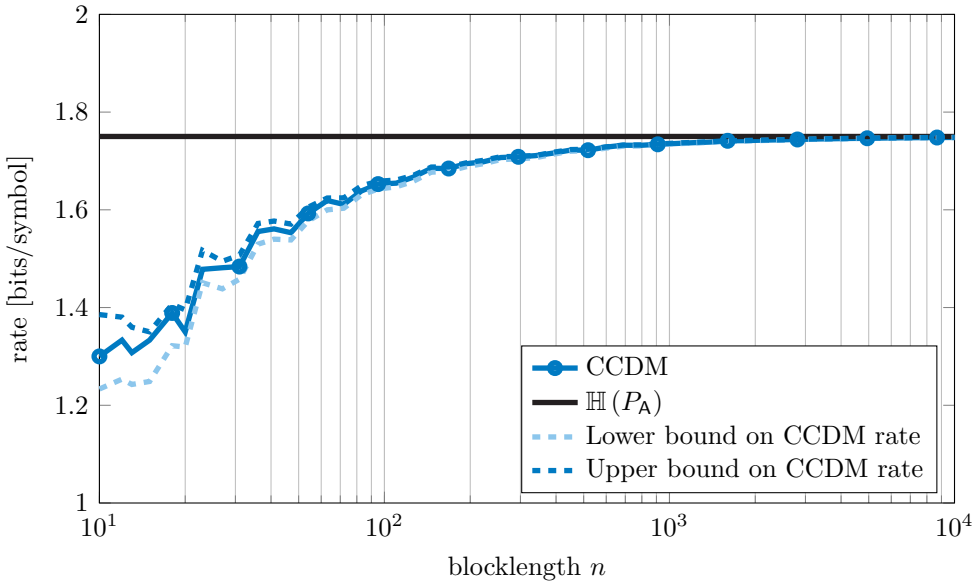


Figure 4.7: Rates of CCDM versus output blocklengths for $P_{\mathsf{A}} = (0.0722, 0.1654, 0.3209, 0.4415)$.

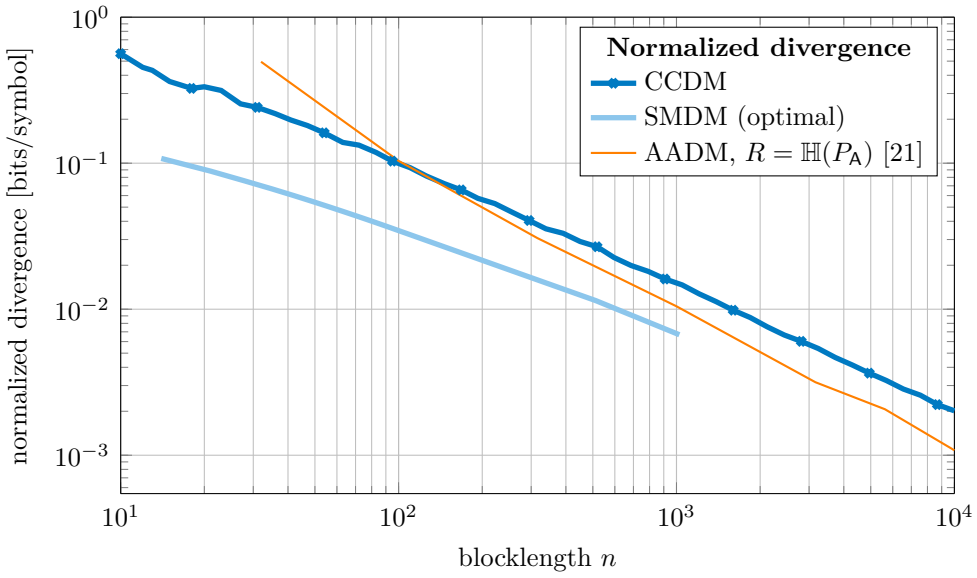


Figure 4.8: Normalized divergence of CCDM versus output blocklengths for $P_{\mathsf{A}} = (0.0722, 0.1654, 0.3209, 0.4415)$. For comparison, the performance of SMDM that is optimal for block codes, and AADM [21] is displayed. Because of limited computational resources, we show the performance of SMDM only up to a blocklengths of $n = 1000$.

CCDM and SMDM Comparison

Divergence

Consider the output alphabet $\mathcal{A} = \{1, 3, 5, 7\}$, rate $R = 1.75$ and an MB distribution. The desired distribution is

$$P_{\mathsf{A}} = (0.0722, 0.1654, 0.3209, 0.4415).$$

Fig. 4.7 and Fig. 4.8 show the rates and normalized divergences of CCDM and SMDM, respectively. Note that SMDM is an optimal distribution matcher for the rate $R = 1.75$. The empirical performance of AADM [21] is also displayed. For optimal f2f and AADM, the rate is fixed to $\mathbb{H}(P_{\mathsf{A}})$ bits per symbol. Fig. 4.8 shows that CCDM needs about 160 symbols to reach an informational divergence of 0.06 bits per symbol, which is about 4 times the blocklengths of SMDM. Fig. 4.7 also shows the lower and upper bounds (4.9) and (4.100), respectively.

Rate Adaptation

Rate adaptation for SMDM is straightforward for distributions of the form (4.50). The number of bits that are interpreted as the index of the ordered list can be easily adapted, and therefore the rate can be easily adapted. The granularity of rate adaption is $1/n$, where n is the output length. This granularity is the best possible. However, n cannot be chosen arbitrarily large. CCDM can achieve a very high granularity because the number of types grows polynomially in block length which corresponds to the number of supported rates. For long block lengths CCDM supports virtually any rate.

Coded Results

We compare the performance of SMDM and CCDM for PAS in a coded scenario. We target an SE of 1.5 bits per channel use with an 8-ASK constellation. We employ LDPC codes from the recent 5G eMBB standard [27] with blocklengths 192, 384 and 768 bits, i.e., 64, 128 and 256 8-ASK constellation points. Note that a 64-QAM constellation can be constructed as the Cartesian product of two (bipolar) 8-ASK constellations, where the latter has four different amplitude values. Consequently we can implicitly consider the sequence of 32, 64 and 128 64-QAM constellation points. The uniform reference curve uses a rate $R_{\mathrm{c}} = 1/2$ code, whereas the shaped scenarios use a rate $R_{\mathrm{c}} = 3/4$ code. Both DM approaches have a 4-ary output alphabet to generate the shaped amplitude sequences for the real and imaginary parts. We need 64, 128 and 256 amplitudes. The target distribution in both cases is the MB family. The CCDMs operate with output blocklengths matched to the code length. The SMDMs use a block length of 32 independent of the code length. This way we keep the algorithmic requirements small. For the block length $n = 64$ the PAS with CCDM performance (blue curve) is similar

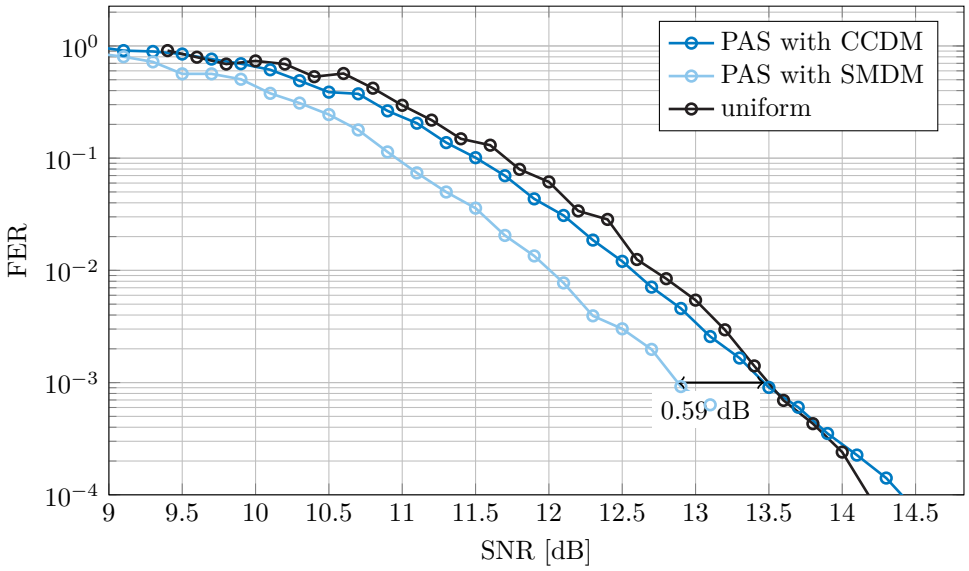


Figure 4.9: Finite length performance for uniform and shaped signaling using CCDM and SMDM. We target a spectral efficiency (SE) of 1.5 bits per dimension with 5G LDPC codes of blocklengths 192. CCDM and SMDM use block lengths $n = 64$ and $n = 32$, respectively.

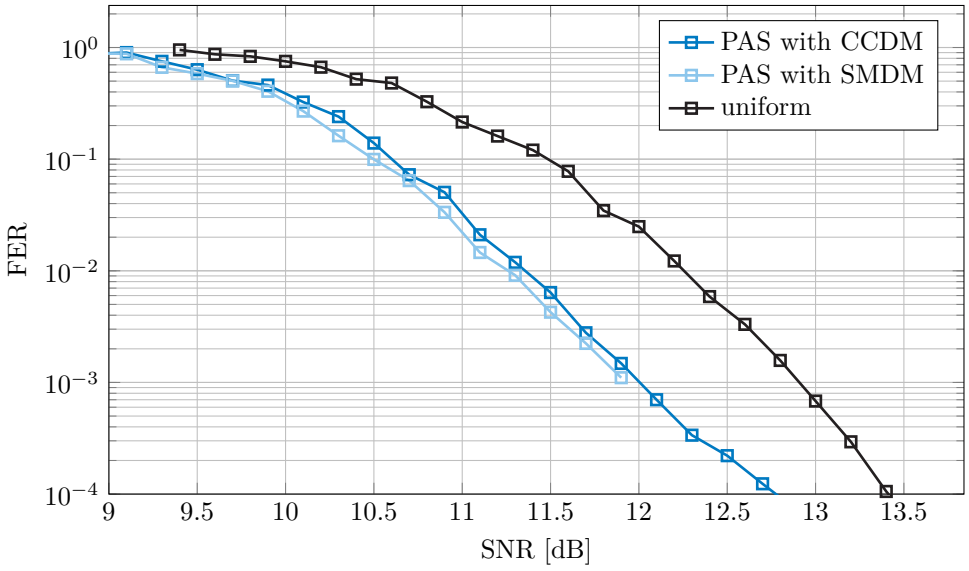


Figure 4.10: Finite length performance for uniform and shaped signaling using CCDM and SMDM. We target a SE of 1.5 bits per dimension with 5G LDPC codes of blocklengths 384. CCDM and SMDM use block lengths $n = 128$ and $n = 32$, respectively.

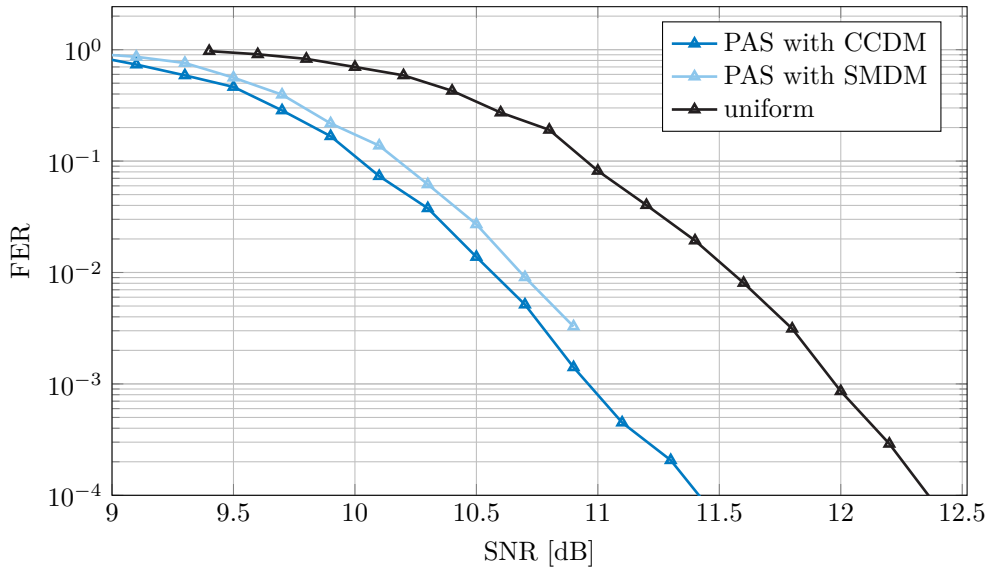


Figure 4.11: Finite length performance for uniform and shaped signaling using CCDM and SMDM. We target a SE of 1.5 bits per dimension with 5G LDPC codes of block length 768. CCDM and SMDM use block lengths $n = 256$ and $n = 32$, respectively.

to the uniform reference (green curve) in Fig. 4.9. The constant composition constraint of CCDM thus causes a significant rate loss for small output blocklengths. In contrast, SMDM operates with an output block length of $n = 32$ (i.e., two SMDM are used in parallel) but gains 0.59dB in power efficiency at a frame error rate of 10^{-3}. For longer block lengths, i.e., $n = 128$ and $n = 256$ CCDM reaches the performance of SMDM or is even better, see Fig. 4.10 and Fig. 4.11. This is due to the restriction of the block length of SMDM to 32.

4.3.3 Code Trellis

The construction of the CC trellis borrows ideas from [34]. The trellis states are tuples

$$\mathcal{S} = \{0, 1, \cdots, n_{\alpha_1}\} \times \{0, 1, \cdots, n_{\alpha_2}\} \times \cdots \{0, 1, \cdots, n_{\alpha_{|\mathcal{A}|}}\}. \tag{4.102}$$

The number of states in the trellis is

$$|\mathcal{S}| = \prod_{\alpha \in \mathcal{A}} (n_\alpha + 1) \tag{4.103}$$

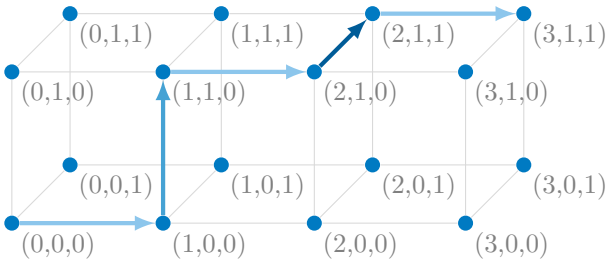


Figure 4.12: Constant composition code trellis for type $\boldsymbol{t} = (3, 1, 1)$. This trellis consists of 16 states and 28 branches and represents 20 different CC code words or paths.

and the number of edges is

$$E = \sum_{\alpha \in \mathcal{A}} n_\alpha \prod_{\alpha' \neq \alpha} (n_{\alpha'} + 1). \tag{4.104}$$

The initial and final states are $(0, \cdots, 0)$ and $(n_{\alpha_1}, \cdots, n_{\alpha_{|\mathcal{A}|}})$, respectively. State $s \in \mathcal{S}$ is connected to an earlier state $s' \in \mathcal{S}$ via symbol α_q if all entries are identical except for the q-th entry of s that is augmented by one.

Example 4.7. Consider a CC code on the alphabet $\mathcal{A} = \{\alpha_1, \alpha_2, \alpha_3\}$ and with type $\boldsymbol{t} = (3, 1, 1)$. The trellis is depicted in Fig. 4.12. It consists of $|\{0, 1, 2, 3\}| \cdot |\{0, 1\}| \cdot |\{0, 1\}| = 16$ states. The colored path corresponds to the sequence $[\alpha_1\alpha_2\alpha_1\alpha_3\alpha_1]$. It includes three increment-steps of α_1, one increment-step of α_2, and one increment-step of α_3, and therefore matches the sequence type.

Note that this code trellis includes the complete set $\mathcal{T}_{\boldsymbol{t}}$ of sequences of type $\boldsymbol{t}$, however $\mathcal{C}_{\text{ccdm}}$ is usually only a subset.

4.3.4 Algorithm - Enumerative Coding

For the implementation of a CC code book with enumerative coding, we need to determine how many code words in the code book start with a prefix b^i. We use algorithms based on (2.47). The prefix b^i tells how often each letter in the alphabet occurred and we can calculate which letters are still missing. It remains to determine how many distinct permutations of the remaining letters are still possible, i.e., we have

$$n_{b^i}(\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}) = \frac{(n-i)!}{\prod_{\alpha \in \mathcal{A}} (P_{\bar{\mathsf{A}}}(\alpha) \cdot n - n_\alpha(b^i))!}. \tag{4.105}$$

If any of the differences $P_{\bar{\mathsf{A}}}(\alpha) \cdot n - n_\alpha(b^i)$ is negative, then b^i is not a valid prefix of the code word and the expression evaluates to zero.

4.3.5 Algorithm - Divide-and-Conquer Enumerative Coding

To implement a CC code book we introduce the ordering of code words within a type. We assume an order among the letters in the alphabet $\mathcal{A}$. Each type set has a *base word* which is the first word in lexicographical ordering. This code word can be easily constructed by concatenating n_α times the letter α with n_β times the letter β and so on with $\alpha < \beta < \ldots$ according to the order. We write $\boldsymbol{b}'(\boldsymbol{c})$ to get the base word of the type set that the sequence $\boldsymbol{c}$ belongs to.

Consider two sequences $\boldsymbol{c}$ and $\boldsymbol{d}$. $\boldsymbol{c}$ is first in order if

1. The base word of the first half of $\boldsymbol{c}$ comes before the base word of the first half of $\boldsymbol{d}$ in lexicographical order.
2. The base word of the first halves of $\boldsymbol{c}$ and $\boldsymbol{d}$ are the same and the first halves are not identical. The first half of $\boldsymbol{c}$ comes before the first half $\boldsymbol{d}$ according to these rules.
3. The first halves of $\boldsymbol{c}$ and $\boldsymbol{d}$ are identical and the second half of $\boldsymbol{c}$ comes before the second half $\boldsymbol{d}$ according to these rules.

In these cases we write $\boldsymbol{c} < \boldsymbol{d}$.

For the implementation below, we require n to be a power of 2.

Decoding

We derive the number $N(\boldsymbol{c})$ of sequences of *the same type* that come before a length n sequence $\boldsymbol{c}$ according to the above rules. In this section we write $\boldsymbol{c}^1$ for the first half of $\boldsymbol{c}$ and $\boldsymbol{c}^2$ for the second half.

The number N_1 of sequences that are below a sequence $\boldsymbol{c}$ according to the first rule is

$$N_1 = \sum_{\boldsymbol{b}<\boldsymbol{b}'(\boldsymbol{c}^1)} \binom{n/2}{\boldsymbol{t}_{\boldsymbol{b}}} \binom{n/2}{\boldsymbol{t}_{\boldsymbol{c}} - \boldsymbol{t}_{\boldsymbol{b}}} \tag{4.106}$$

where the summation is over all base words $\boldsymbol{b} < \boldsymbol{b}'(\boldsymbol{c}^1)$, i.e., all base words that come before the base word of $\boldsymbol{c}^1$ in lexicographical ordering. The first multinomial evaluates how many permutations there are of the base word $\boldsymbol{b}$ and the second multinomial evaluates the number of permutations for the remaining letters such that the type of $\boldsymbol{c}$ is correct. The second multinomial may not exist and becomes 0 in this case.

The number N_2 of sequences that are below a sequence $\boldsymbol{c}$ according to the second rule is

$$N_2 = N(\boldsymbol{c}^1) \cdot \binom{n/2}{\boldsymbol{t}_{\boldsymbol{c}^2}}. \tag{4.107}$$

The number N_3 of sequences that are below a sequence $\boldsymbol{c}$ according to the third rule is

$$N_3 = N(\boldsymbol{c}^2). \tag{4.108}$$

The decoded position $N(\boldsymbol{c})$ in the list or rather the number of sequences that come first is

$$N(\boldsymbol{c}) = N_1 + N_2 + N_3 \tag{4.109}$$

where the summands N_2 and N_3 are defined recursively. We reach the stop criteria for length two. Solving the problem for a length two sequence $[a_1, a_2]$ is easy because there are only a few possible cases. If both letters a_1 and a_2 are the same there is no other permutation, i.e., $N([a_1, a_2]) = 0$. If $a_1 < a_2$ then the permutation $[a_2, a_1]$ is a successor and $N([a_1, a_2]) = 0$ and if $a_1 > a_2$ then $N([a_1, a_2]) = 1$ must hold.

Example 4.8 (Decoding of a CC sequence with divide-and-conquer enumerative coding)**.** Consider the sequence $\boldsymbol{c} = [1, 2, 0, 0, 1, 0, 1, 0]$ and its type vector $\boldsymbol{t_c} = [4, 3, 1]$. This sequence is divided into four sequences of length two, i.e., $N([1, 2])$, $N([0, 0])$, $N([1, 0])$ and $N([1, 0])$. $N([1, 2])$ and $N([0, 0])$ both evaluate to zero, because $[2, 1]$ is second in lexicographical order and $[0, 0]$ does not have a permutation. $N([1, 0])$ evaluates to 1 because the permutation $[0, 1]$ is first in order. With this information, we can compute N_2 and N_3 of $N(1, 2, 0, 0)$ and $N([1, 0, 1, 0])$. N_2 and N_3 of $N([1, 2, 0, 0])$ both evaluate to 0, and N_2 and N_3 of $N([1, 0, 1, 0])$ evaluate to 2 and 1, respectively. For the sequence $[1, 2, 0, 0]$ we consider first the base of $[1, 2]$, which is $[1, 2]$ itself. Base words that come first are $[0, 2]$, $[0, 1]$ and $[0, 0]$. Note that $[1, 1]$ is not included in the list because this sequence violates the type constraint of $[1, 2, 0, 0]$. N_1 then evaluates to

$$N_1 = \sum_{\boldsymbol{b} \in \{[0,0],[0,1],[0,2]\}} \binom{2}{\boldsymbol{t_b}} \binom{2}{\boldsymbol{t} - \boldsymbol{t_b}} \tag{4.110}$$

$$= \binom{2}{2, 0, 0} \binom{2}{0, 1, 1} + \binom{2}{1, 1, 0} \binom{1}{1, 0, 1} + \binom{2}{1, 0, 1} \binom{2}{1, 2, 0} \tag{4.111}$$

$$= 2 + 4 + 4 = 10. \tag{4.112}$$

In a similar way we calculate that N_1 of $N([1, 0, 1, 0])$ is 4. In the final step we calculate N_2 and N_3 from the previous results. The first part of the word is $[1, 2, 0, 0]$. The base word of $[1, 2, 0, 0]$ is $[0, 0, 1, 2]$ and base words that come first and do not violate the type vector $\boldsymbol{t_c}$ are $[0, 0, 0, 0], [0, 0, 0, 1], [0, 0, 0, 2]$ and $[0, 0, 1, 1]$. N_1 of the base problem evaluates to

$$N_1 = \sum_{\substack{\boldsymbol{b} \in \{[0,0,0,0],[0,0,0,1],\\ [0,0,0,2],[0,0,1,1]\}}} \binom{4}{\boldsymbol{t_b}} \binom{4}{\boldsymbol{t} - \boldsymbol{t_b}} \tag{4.113}$$

$$= \binom{4}{4, 0, 0} \binom{4}{0, 3, 1} + \binom{4}{3, 1, 0} \binom{4}{1, 2, 1} + \binom{4}{3, 0, 1} \binom{4}{1, 3, 0} + \binom{4}{2, 2, 0} \binom{4}{2, 1, 1} \tag{4.114}$$

$$= 4 + 48 + 16 + 72 = 140. \tag{4.115}$$

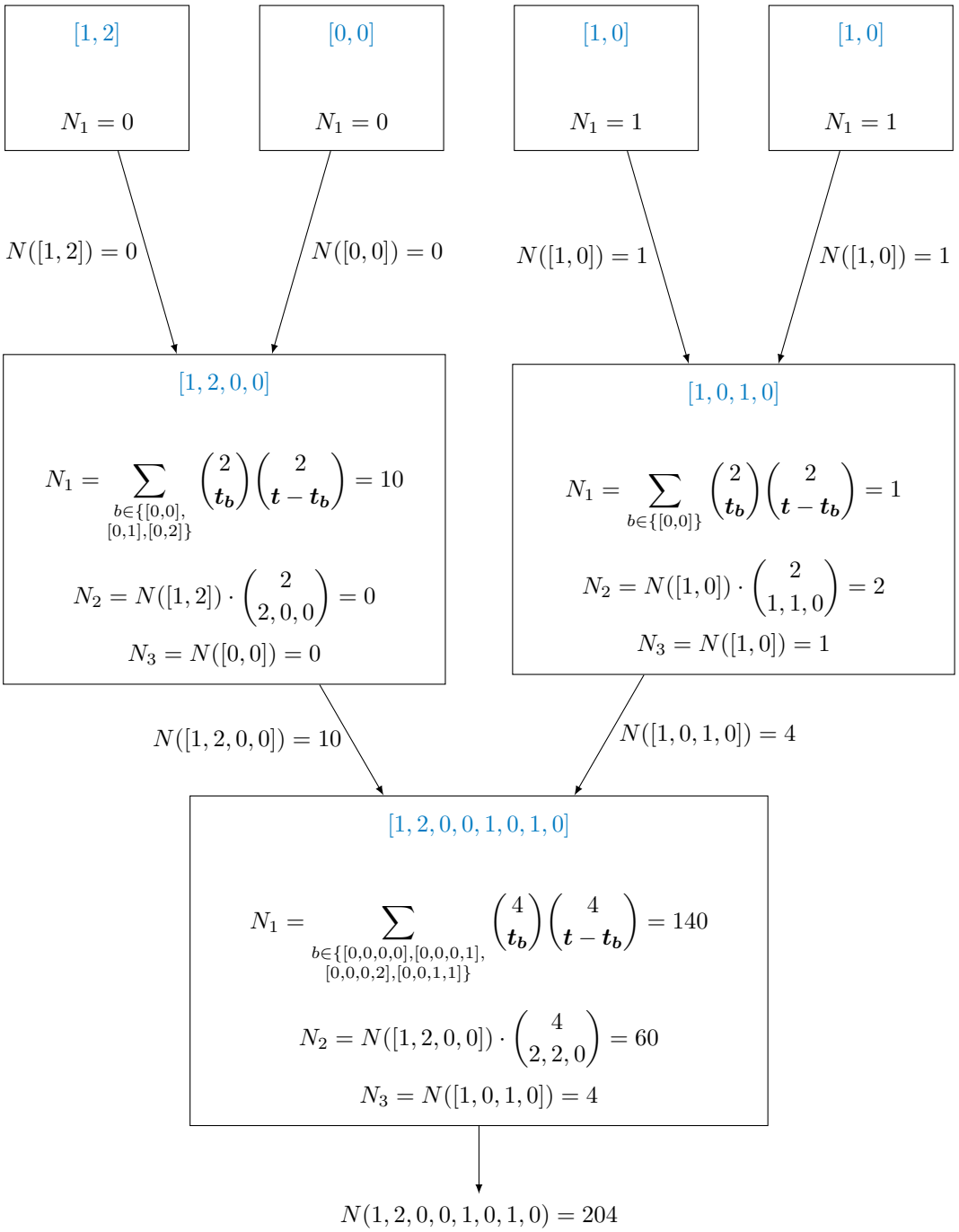


Figure 4.13: Decoding of the sequence $[1, 2, 0, 0, 1, 0, 1, 0]$ in the tree structure.

One can easily check that $N([1,0,1,0])$ evaluates to 4. For $N([1,2,0,0,1,0,1,0])$ we obtain $N_2 = 60$ and $N_3 = 4$ which means that $N([1,2,0,0,1,0,1,0]) = 204$. The steps are shown in Fig. 4.13.

Encoding

Encoding follows a similar strategy as decoding. During decoding we are aware of the types and want to calculate how many sequences there are before in the ordered list, while encoding tries to find the types of subsequences knowing how many sequences came before in the ordered list. Thus $N(\boldsymbol{c})$ is split into three parts N_1, N_2, N_3 and the algorithm determines the type of the first and second parts. At a length of 1, the type directly specifies the letter.

1. Find the base word $\boldsymbol{b}'$ such that

$$\sum_{\boldsymbol{b}<\boldsymbol{b}'} \binom{n/2}{\boldsymbol{t_b}} \binom{n/2}{\boldsymbol{t_c}-\boldsymbol{t_b}} \leq N(\boldsymbol{c}) < \sum_{\boldsymbol{b}\leq\boldsymbol{b}'} \binom{n/2}{\boldsymbol{t_b}} \binom{n/2}{\boldsymbol{t_c}-\boldsymbol{t_b}}. \tag{4.116}$$

2. This base vector $\boldsymbol{b}'$ determines the type of the first part of the sequence and therefore also of the second part.

3. Furthermore we can calculate N_1 of the decoding step and obtain the index of the first and second half via

$$N(\boldsymbol{c}^1) = \left\lfloor \frac{N(\boldsymbol{c}) - N_1}{\binom{n/2}{\boldsymbol{t}_{\boldsymbol{c}^2}}} \right\rfloor \tag{4.117}$$

$$N(\boldsymbol{c}^2) = N(\boldsymbol{c}) - N_1 - N(\boldsymbol{c}^1) \cdot \binom{n/2}{\boldsymbol{t}_{\boldsymbol{c}^2}}. \tag{4.118}$$

4. Apply rules 1 to 3 until we have length 2 sequences. The type vector indicates at most two different letters a_1 and a_2 with $a_1 \leq a_2$. If $N(\boldsymbol{c}) = 0$ then return $[a_1, a_2]$, else $[a_2, a_1]$.

Example 4.9. We want to encode the integer 204 into the type set of the type vector $[4,3,1]$. We try to find the base word such that (4.116) is fulfilled. Possible base words that do not violate the CC constraint are

$$\begin{aligned}\boldsymbol{b'} \in \{&[0,0,0,0],[0,0,0,1],[0,0,0,2],[0,0,1,1],\\ &[0,0,1,2],[0,1,1,1],[0,1,1,2],[1,1,1,2]\}\end{aligned}$$

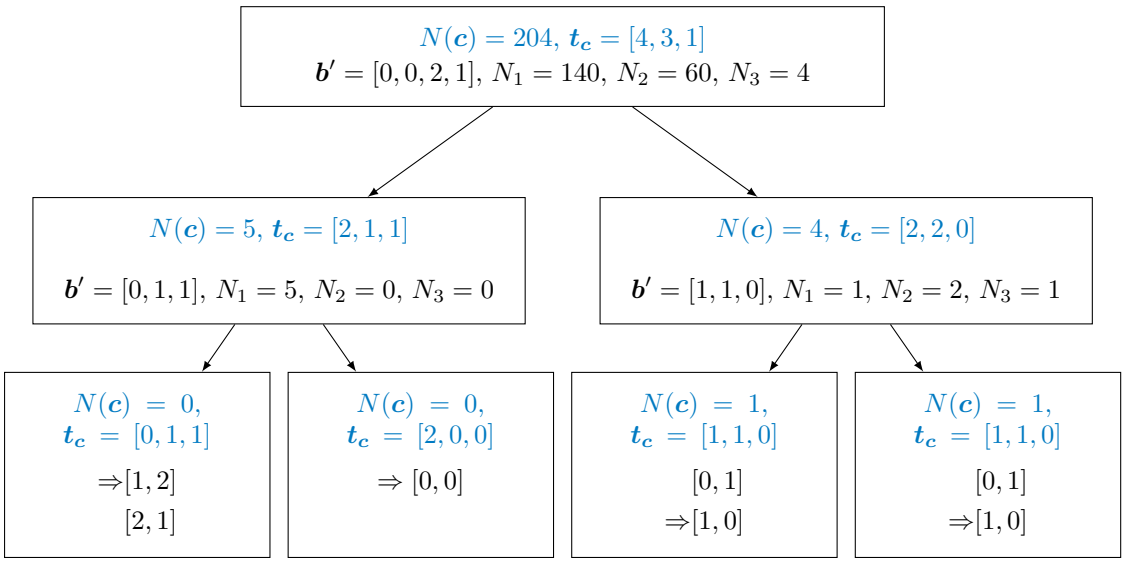


Figure 4.14: Encoding of the index $N(\boldsymbol{c}) = 204$ and type vector $\boldsymbol{t_c} = [4,3,1]$ with its subproblems in a tree.

and they have $[1,4,4,6,12,4,12,4]$ permutations. We find for $\boldsymbol{b'} = [0,0,1,2]$ that (4.116) holds true, i.e., we have

$$\sum_{\boldsymbol{b}<\boldsymbol{b'}} \binom{n/2}{\boldsymbol{t_b}} \binom{n/2}{\boldsymbol{t_c}-\boldsymbol{t_b}} = 140 \leq N(\boldsymbol{c}) < 212 = \sum_{\boldsymbol{b}\leq\boldsymbol{b'}} \binom{n/2}{\boldsymbol{t_b}} \binom{n/2}{\boldsymbol{t_c}-\boldsymbol{t_b}}. \tag{4.119}$$

Consequently the type vector of the second part is $[4,3,1]-[2,1,1] = [2,2,0]$. Using equations (4.117) and (4.118) we obtain for the fist part $N(\boldsymbol{c}^1) = \lfloor 64/12 \rfloor = 5$ and for the second part $N(\boldsymbol{c}^2) = 64 - 60 = 4$. We call the recursive algorithm for the first subsequence with $N(\boldsymbol{c}) = 5$ and $\boldsymbol{t_c} = [2,1,1]$. For the first part of this sequence, all base words of length 2 fulfill the type constraint, i.e., we have

$$\boldsymbol{b'} \in \{[0,0],[0,1],[0,2],[1,2]\}. \tag{4.120}$$

For $\boldsymbol{b'} = [1,2]$ we find that (4.116) holds true and $N_1 = 5$. This implies that the base word of the second part is $[2,1,1]-[0,1,1] = [2,0,0]$ and $N(\boldsymbol{c}^1) = N(\boldsymbol{c}^2) = 0$. Therefore the concatenation of base words $[1,2],[0,0]$ gives the solution. The algorithm runs the recursion with $N(\boldsymbol{c}) = 4$ and $\boldsymbol{t_c} = [2,2,0]$ and finds $\boldsymbol{c} = [1,0,1,0]$. The structure of the algorithm is summarized with all intermediate steps in Fig. 4.14.

4.3.6 Algorithm - Arithmetic Coding

We use arithmetic coding to index sequences efficiently. In [22] an m-out-of-n coding scheme was introduced that uses the same technique. Our arithmetic encoder associates

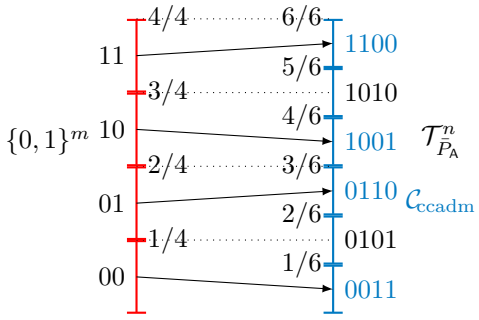


Figure 4.15: Diagram of a constant composition arithmetic encoder with $P_{\bar{\mathsf{A}}}(0) = P_{\bar{\mathsf{A}}}(1) = 0.5$, $m = 2$ and $n = 4$.

an interval to each input sequence in $\{0,1\}^m$ and it associates an interval to each output sequence, see Fig. 4.15 for an example.

The size of an interval is equal to the probability of the corresponding sequence according to the input and output models, respectively. For the input model we choose an iid Bernoulli(1/2) process. We describe the output model by a random vector

$$\bar{\mathsf{A}}^n = \bar{\mathsf{A}}_1 \bar{\mathsf{A}}_2 \dots \bar{\mathsf{A}}_n \tag{4.121}$$

with marginals $P_{\bar{\mathsf{A}}_i} = P_{\bar{\mathsf{A}}}$ and the uniform distribution

$$P_{\bar{\mathsf{A}}^n}(a^n) = \frac{1}{|\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}|} \quad \forall a^n \in \mathcal{T}^n_{P_{\bar{\mathsf{A}}}}.$$

The intervals are ordered lexicographically. All input and output intervals range from 0 to 1 because all probabilities add up to 1.

Example 4.10. Fig. 4.15 shows input and output intervals with output length $n = 4$ and $P_{\bar{\mathsf{A}}}(0) = P_{\bar{\mathsf{A}}}(1) = 0.5$. There are 4 equally probable input sequences and 6 equally probable output sequences. The intervals on the input side are $[0, 0.25)$, $[0.25, 0.5)$, $[0.5, 0.75)$ and $[0.75, 1)$. The intervals on the output side are $[0, \frac{1}{6})$, $[\frac{1}{6}, \frac{2}{6})$, $[\frac{2}{6}, \frac{3}{6})$, $[\frac{3}{6}, \frac{4}{6})$, $[\frac{4}{6}, \frac{5}{6})$ and $[\frac{5}{6}, 1)$.[1]

The arithmetic encoder can link an output sequence to an input sequence if the lower border of the output interval is inside the input interval. In the example (Fig. 4.15) 00 may link to both 0101 and 0011, while for 01 only a link to 0110 is possible. There are at most two possible choices because by (4.87) the input interval size is less than twice the output interval size. At the same time there is at least one choice, because by equation (4.88) the input interval is at least as big as the output interval. Both choices are valid

[1] Please note that in this case no DM is needed. However, this indexing problem is of interest in its own right.

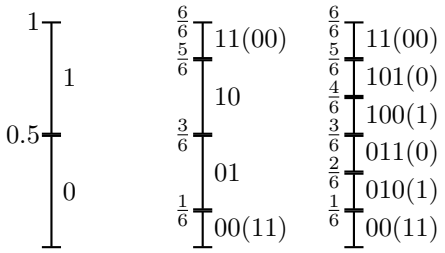


Figure 4.16: Refinement of the output intervals. Round brackets indicate symbols that must follow with probability one.

and we can perform an inverse operation. In our implementation, the encoder decides for the output sequence with the lowest interval border. As a result, the code book $\mathcal{C}_{\text{ccdm}}$ of Example 4.10 is $\{0011, 0110, 1001, 1100\}$. In general $\mathcal{C}_{\text{ccdm}}$ has cardinality 2^m with $2^m \leq |\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}| < 2^{m+1}$ according to (4.87) and (4.88). There are also other mappings that lead to a unique mapping. It is not possible to index the whole set $\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}$ unless $2^m = |\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}|$. The analysis of the code (Sec. 4.3.2) is valid for all code books $\mathcal{C}_{\text{ccdm}} \subseteq \mathcal{T}^n_{P_{\bar{\mathsf{A}}}}$. The actual subset is implicitly defined by the arithmetic encoder.

We now discuss an online algorithm that processes the input sequentially. Initially, the input interval spans from 0 to 1. As the input model is Bernoulli(1/2) we split the interval into two equally sized intervals and continue with the upper interval in case the first input bit is '1'; otherwise we continue with the lower interval. After the next input bit arrives we repeat the last step. After m input bits we reach a size 2^{-m} interval. After every refinement of the input interval the algorithm checks for a sure prefix of the output sequence, e.g., in Fig. 4.15 we see that if the input starts with 1 then the output must start with 1. Every time we extend the sure prefix by a new symbol, we must calculate the probability of the next symbol given the sure prefix. That means we determine the output intervals within the sure interval of the prefix. The model for calculating the conditioned probabilities is based on drawing without replacement. There is a bag with n symbols of k discriminable kinds. n_a denotes how many symbols of kind a are initially in the bag and n'_a is the current number. The probability to draw a symbol of type a is n'_a/n. If we pick a symbol a then both n and n'_a decrement by 1.

Example 4.11. Fig. 4.16 shows a refinement of the output intervals. Initially there are 2 '0's and 2 '1's in the bag. The distribution of the first drawn symbol is $P_{\bar{\mathsf{A}}_1}(0) = P_{\bar{\mathsf{A}}_1}(1) = \frac{1}{2}$. When drawing a '0', there are 3 symbols remaining: one '0' and two '1's. Thus, the probability for a '0' reduces to 1/3 while the probability of '1' is 2/3. If two '0's were picked, two '1's must follow. This way we ensure that the encoder output is of the desired type. Observe that the probabilities of the next symbol depend on the previous symbols, e.g., we have

$$P_{\bar{\mathsf{A}}_2|\bar{\mathsf{A}}_1}(0|0) \neq P_{\bar{\mathsf{A}}_2|\bar{\mathsf{A}}_1}(0|1) \tag{4.122}$$

in general. However, $P_{\bar{\mathsf{A}}^n}(a^n) = \prod_{i=1}^{n} P_{\bar{\mathsf{A}}_i|\bar{\mathsf{A}}^{i-1}}(a_i|a^{i-1})$ is constant on $\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}$ as we show in the following proposition.

Proposition 4.7. After n refinements of the output interval the model used for the refinement step stated above creates equally spaced (equally probable) intervals that are labeled with all sequences in $\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}$.

Proof. All symbols in the bag are chosen at some point. Consequently only sequences in $\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}$ may appear. All possibilities associated with the chosen string are products of fractions n'_a/n, where n takes on all values from the initial value to 1 because every symbol is drawn at some point. Thus for each string we obtain for its probability an expression that is independent of the realization itself:

$$P_{\bar{\mathsf{A}}^n}(a^n) = \frac{n_{a=0}! \cdots n_{a=k-1}!}{n!} = \frac{1}{|\mathcal{T}^n_{P_{\bar{\mathsf{A}}}}|} \quad \forall a^n \in \mathcal{T}^n_{P_{\bar{\mathsf{A}}}}. \tag{4.123}$$

■

Numerical problems for representing the input interval and the output interval occur after a certain number of input bits. For this reason we introduce a *rescaling* each time a new output symbol is known. We explain this next.

Scaling input and output intervals

After we identify a sure prefix, we are no longer interested in code sequences that do not have that prefix. We scale the input and output interval such that the output interval is [0,1). Fig. 4.17 illustrates the mapping of intervals (in_1, out_1) to (in_2, out_2). The refinement for the second symbol works as described in Example 4.11. If the second input bit is 0, we know that 10 must be a prefix of the output. The resulting scaling is shown in Fig. 4.17 as (in_2, out_2) to (in_3, out_3). A more detailed explanation of scaling for arithmetic coding can be found for instance in [79, Chap. 4]. We provide an online implementation of CCDM in [80].

4.4 Product Distribution Matching

This section is based mainly on ideas presented in [70, 81]. A similar idea was presented in [82].

4.4.1 Approach

Consider an M-ary target distribution P_{A} that can be decomposed into a product of distributions, i.e.,

$$P_{\mathsf{A}}(b_A(b_1, b_2, ..., b_m)) = \prod_{i}^{m} P_{\mathsf{B_i}}(b_i) \tag{4.124}$$

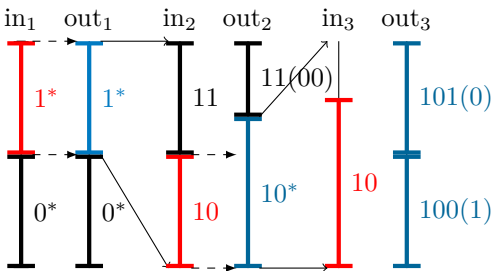


Figure 4.17: Scaling of input and output intervals in case the input interval is a subset of an output interval. The latter interval corresponds to $[0, 1)$ after scaling. A star indicates that this is just a prefix of the complete word. Round brackets indicate symbols that must follow with probability one.

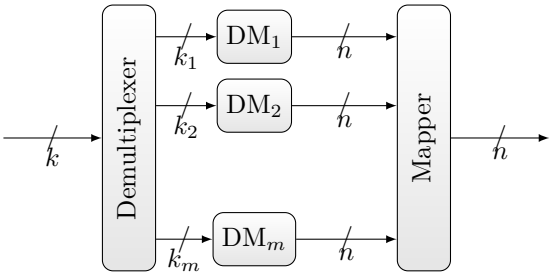


Figure 4.18: Schematic for PDM with m component DMs. A block of k bits is split into parallel blocks of sizes k_1 to k_m. The DMs output m shaped sequences of length n. A mapper recombines the sequences to one length n sequence.

where $b_i \in \mathcal{A}_i$, $\prod_{i=1}^{m} |\mathcal{A}_i| = M$ and $b_A(\cdot)$ is an invertible mapping of m symbols to an M-ary symbol. In this case, one can split one DM into multiple independent *component* DMs and use the mapping to generate those symbols.

Example 4.12. Consider the distribution $P_{\mathsf{A}}(0) = P_{\mathsf{A}}(1) = 0.3$ and $P_{\mathsf{A}}(2) = P_{\mathsf{A}}(3) = 0.2$ with blocks of length 10. A CCDM uses permutations of the sequence $[0, 0, 0, 1, 1, 1, 2, 2, 3, 3]$ to approximate the distribution P_{A}. There are $\left|\mathcal{T}_{P_{\mathsf{A}}}^{10}\right| = 25200$ such sequences. An alternative approach is to use two binary CCDMs with distributions $P_{\mathsf{B}_1} \sim$ Bernoulli(3/5) and $P_{\mathsf{B}_2} \sim$ Bernoulli(1/2) and the mapping

$$b_A(0,0) = 0 \tag{4.125}$$
$$b_A(0,1) = 1 \tag{4.126}$$
$$b_A(1,0) = 2 \tag{4.127}$$
$$b_A(1,1) = 3. \tag{4.128}$$

The number of sequences that that we can address with the PDM is

$$\binom{10}{5} \cdot \binom{10}{4} = 52920. \tag{4.129}$$

This means that we can choose from about 2 times more sequences and therefore possibly use one bit more for indexing. The reason for the increased number of sequences is that not only permutations of

$$[0, 0, 0, 1, 1, 1, 2, 2, 3, 3]$$

can be indexed, but also permutations of

$$[0, 0, 0, 0, 0, 0, 1, 3, 3, 3, 3]$$
$$[0, 0, 0, 0, 1, 1, 3, 3, 3, 2]$$
$$[0, 0, 1, 1, 1, 1, 3, 2, 2, 2]$$
$$[0, 1, 1, 1, 1, 1, 2, 2, 2, 2].$$

4.4.2 P_{A} is a product distribution

We derive a simple bound on the performance of PDM in comparison to CCDM for binary DMs. In this case all distributions P_{B_i} in (4.124) are Bernoulli(p_i) distributed. Suppose that we can index all permutations, i.e., we use an enumerative technique for the DMs. Then the number of bits that we can transmit within one block for CCDM and PDM are the respective

$$k_{\text{ccdm}} = \log_2(\mathcal{T}_{P_{\mathsf{A}}}^{n}) \tag{4.130}$$

$$k_{\text{pdm}} = k_1 + k_2 + \ldots k_m = \sum_{i=1}^{m} \log_2(\mathcal{T}^n_{P_{\mathsf{B_i}}}). \tag{4.131}$$

In Sec. 4.3.2 we derived bounds on the input length k_{ccdm}:

$$n\mathbb{H}(P_{\mathsf{A}}) - \frac{(M-1)}{2}\log_2(n) - \frac{3}{2}(M-1) - \frac{1}{2}\sum_{i=1}^{M}\log_2(P_{\mathsf{A}}(i))) \tag{4.132}$$

$$\leq k_{\text{ccdm}}$$

$$\leq n\mathbb{H}(P_{\mathsf{A}}) - \frac{(M-1)}{2}\log_2(n) - \frac{(M-1)}{2}\log_2(2\pi) - \frac{1}{2}\sum_{i=1}^{M}\log_2(P_{\mathsf{A}}(i)). \tag{4.133}$$

Using Lemma 2.2 we find for k_{pdm} that

$$n\sum_{i=1}^{m}(\mathbb{H}(p_i)) - \frac{1}{2}(m\log_2(8n) + \sum_{i=1}^{m}\log_2(p_i(1-p_i))) \tag{4.134}$$

$$\leq k_{\text{pdm}}$$

$$\leq n\sum_{i=1}^{m}(\mathbb{H}(p_i)) - \frac{1}{2}(m\log_2(2\pi n) + \sum_{i=1}^{m}\log_2(p_i(1-p_i))). \tag{4.135}$$

We can thus bound the difference $\Delta k = k_{\text{pdm}} - k_{\text{ccdm}}$ from above and below by

$$\frac{(M-1)-m}{2}\log_2(n) + \frac{3(M-1)-m\log_2 2\pi}{2} + \frac{M/2-1}{2}\sum_{i=1}^{m}\log_2(p_i(1-p_i)) \tag{4.136}$$

$$\leq \Delta k = k_{\text{pdm}} - k_{\text{ccdm}} \leq \tag{4.137}$$

$$\frac{(M-1)-m}{2}\log_2(n) + \frac{(M-1)\log_2 2\pi - 3m}{2} + \frac{M/2-1}{2}\sum_{i=1}^{m}\log_2(p_i(1-p_i)). \tag{4.138}$$

This means that the extra information that we can transmit with a PDM as compared to CCDM grows logarithmically in block length n. Furthermore, the factor in front of the logarithm grows with the alphabet size. We can see this behavior in Fig. 4.19 and Fig. 4.20.

4.4.3 P_{A} is not a product distribution

Target distributions are in general not product distributions. If we want to use PDM anyway, we need to approximate the distribution, i.e., find a mapping function $b_A(\cdot)$ and the distributions $P_{\mathsf{B_i}}$ that minimize informational divergence

$$\underset{\substack{P_{\mathsf{B_i}}\ i\in\{1,\ldots m\}\\ b_A}}{\operatorname{argmin}} \mathbb{D}(P_{\mathsf{B}_1} \times P_{\mathsf{B}_2} \times \ldots \times P_{\mathsf{B_m}} \| P_{\mathsf{A}}). \tag{4.139}$$

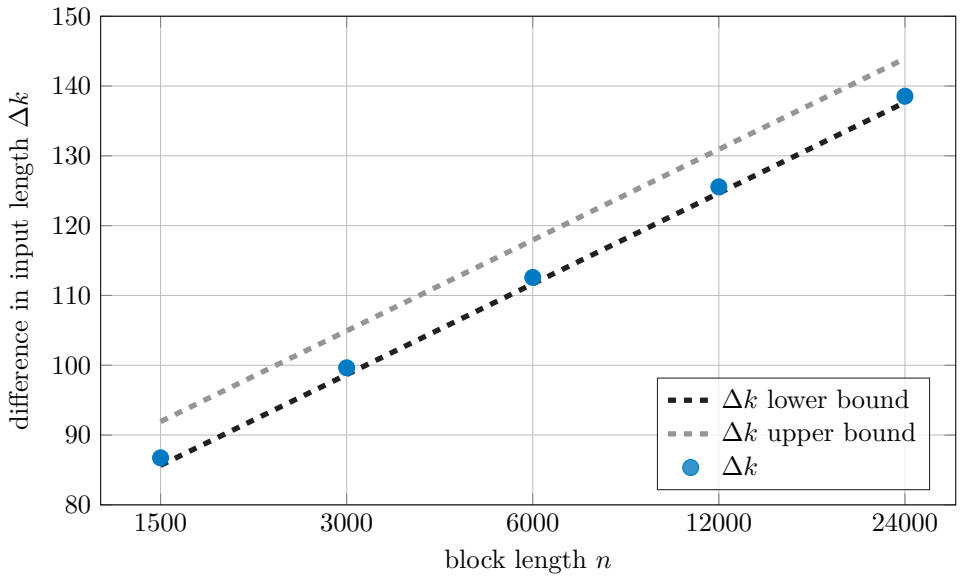


Figure 4.19: Difference of input length $k_{\text{pdm}} - k_{\text{ccdm}}$ for the product distribution $\mathsf{A} \sim$ Bernoulli(1/3) $\cdot$ Bernoulli(1/4) $\cdot$ Bernoulli(2/5) $\cdot$ Bernoulli(2/5) $\cdot$ Bernoulli(1/5) and 5 DMs. The x-axis has a logarithmic scale.

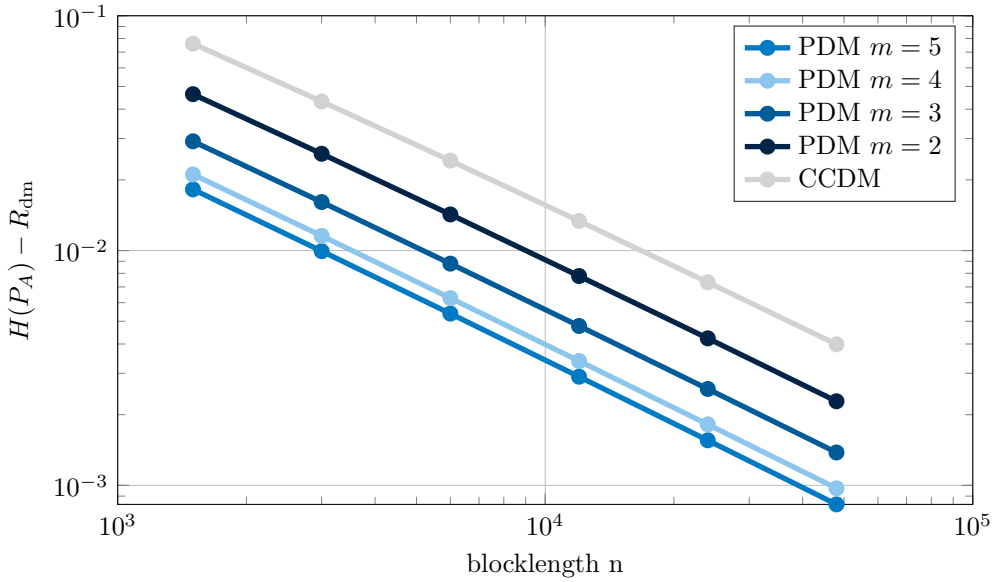


Figure 4.20: Rate loss of PDM. The target distribution is the 32-ary product distribution $P_{\mathsf{A}} =$ Bernoulli(1/3) $\cdot$ Bernoulli(1/4) $\cdot$ Bernoulli(2/5) $\cdot$ Bernoulli(2/5) $\cdot$ Bernoulli(1/5) . The number of component DMs is m. One component DM uses the alphabet size $32/2^{m-1}$, all other component DMs are binary. The case $m = 1$ recovers the CCDM.

We restrict attention to M-ary alphabets (and support) where M is a power of 2, and to binary component DMs. We want to find a good solution that first searches for locally good distributions and then optimizes the mapping function b_A. We iterate both steps. The update rule for the mapping function is:

1. Sort the support of P_A in descending order in a list L_1.
2. Sort the support of the product distribution in descending order in a list L_2.
3. Let the mapping b_A assign the i-th entry of L_2 to the i-th entry of L_1.

For the update rule of the distributions we propose two possibilities.

Inverted Divergence Parameters: Problem (4.139) is easy to solve if we swap the parameters of the divergence operation because the product distributions decouple. In this case, the update rule for the i-th binary pmf $P_{\mathsf{B_i}}$ is

$$P_{\mathsf{B_i}}(0) = \sum_{\substack{b_j \in \{0,1\}, j \neq i \\ b_i = 0}} P_{\mathsf{A}}(b_A(b^m)) \tag{4.140}$$

which means that we add over all letters in the alphabet that are 0 at the i-th position of their binary representation.

Coordinate Descent: When we set the partial derivative of divergence (4.139) of the i-th distribution to zero, we obtain the solution

$$P_{\mathsf{B_i}}(0) = \frac{\exp(W)}{1+\exp(W)} \tag{4.141}$$

with

$$W = \sum_{\substack{b_j \in \{0,1\}, j \neq i \\ b_i = 1}} \prod_{j \neq i} P_{\mathsf{B_j}}(b_j) \log_2 P_{\mathsf{A}}(b_A(b^m)) - \sum_{\substack{b_j \in \{0,1\}, j \neq i \\ b_i = 0}} \prod_{j \neq i} P_{\mathsf{B_j}}(b_j) \log_2 P_{\mathsf{A}}(b_A(b^m)). \tag{4.142}$$

In Fig. 4.21 we compare the divergence of both approaches when we approximate a one-sided MB distribution. Both approaches lead to a similar divergence that is acceptable for energy efficient communication.

4.4.4 Extended Product Distribution Matching

Consider parallel channels with different qualities. [70] specifies a simple extension of PAS that performs shaping for parallel channels. PDM offers a solution for this problem. For example, consider two different channels and suppose we desire a length n sequence for binary and 4-ary alphabets. The PDM needs one binary DM for a first and two binary

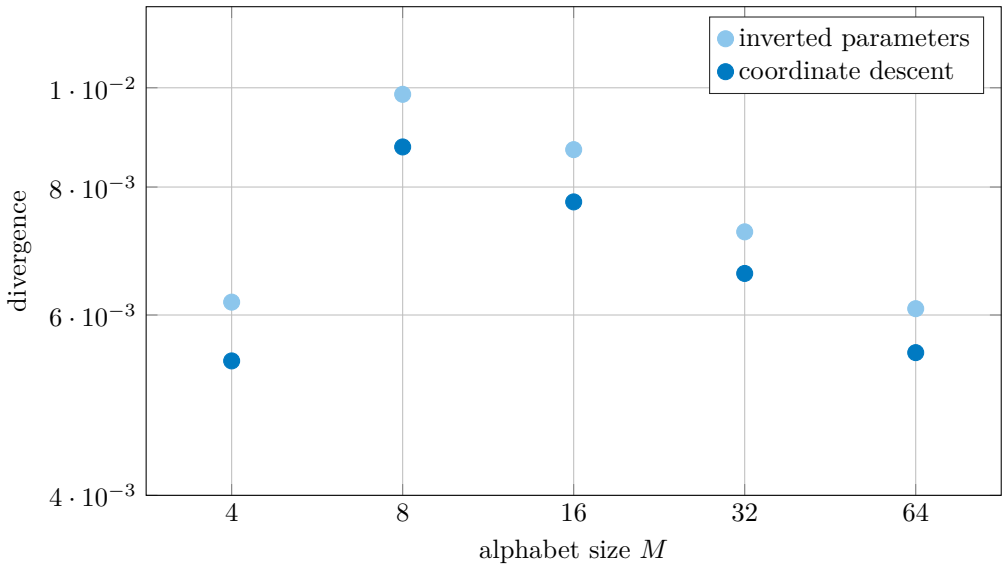


Figure 4.21: Divergence $\mathbb{D}(P_{\bar{\mathsf{A}}} \| P_{\mathsf{A}})$ of PDM when P_{A} is an MB distribution with $\mathbb{H}(P_{\mathsf{A}}) = \log_2(M) - 1$. M is the size of the support of P_{A}.

DMs for a second sequence. As illustrated in Fig. 4.22, we can use one DM with output length $2n$ for both sequences and need one extra for the second sequence with output length n. The potential benefit of this approach is twofold: first, using PDM can reduce the rate loss, and second, replacing two DMs of lengths n by one single DM of length $2n$ should reduce the rate loss even further. Fig. 4.23 shows this extended PDM scheme. It provides the same interface to PAS as the naive approach that uses L individual DMs.

Using one DM for two distributions imposes restrictions on the distribution families that can be generated. We next argue how extended PDM can generate families of Gaussian-like distributions. The maximum constellation size is 2^m and we choose the $m-1$ DM output distributions so that a natural based binary code (NBBC) mapper generates a Gaussian-like distribution. By grouping 2^j neighboring signal points together, the distribution of these signal point groups is still Gaussian-like, and is given by the product distribution generated by the first $m-1-j$ DMs. This suggests that by using only $m-1$ DMs, we can simultaneously generate Gaussian-like distributions on $4, 8, \ldots, 2^m$-ASK constellations. An example is shown in Fig. 4.22.

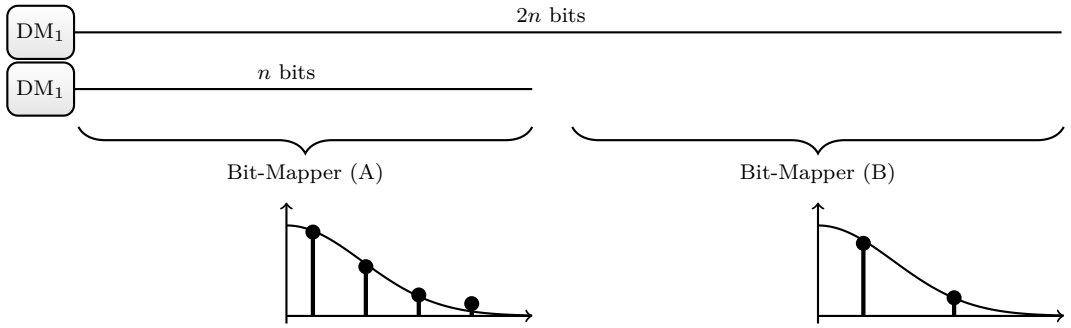


Figure 4.22: Simultaneously generating two Gaussian-like amplitude distributions for alphabet sizes 2 and 4 by reusing a DM.

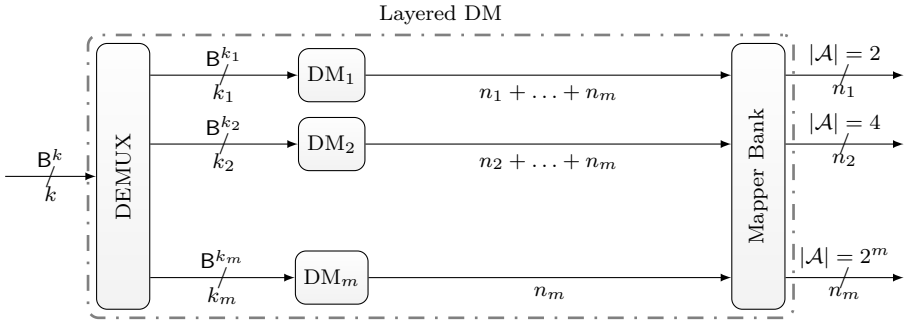


Figure 4.23: Extended PDM encodes k bits into amplitude sequences of predefined lengths. Internally, the extended PDM uses m binary component DMs and 2^m is the largest supported alphabet.

5

Joint Decoding of Shaping and Error Control Codes

In Sec. 3.2 we gave a short introduction to PAS [11] and saw that DMs are an integral part of PAS. In Sec. 4.3.2 CCDM introduces dependencies over all symbols in a block, and a per symbol demodulator cannot consider these dependencies. For very long blocks, the PAS rate is not affected by these dependencies, but systems with short length DMs suffer in transmission rate [71]. In [57, 58, 60, 82], DMs with smaller rate-loss are proposed. In [83] the dependencies introduced by an extremely short 4-D shell mapping (SMDM) [58, 60] are resolved by a 4-D demodulator. The authors of [25] use polar codes with list decoding and check if the code word candidates fulfill the CC constraint.

PAS uses a systematic FEC encoder to preserve a constrained sequence which is similar to the Bliss scheme [12] for a part of the symbol sequence. To improve the Bliss scheme's performance, [84] and [85] use a supplementary soft input soft output (SISO) decoder and iterate with the usual FEC decoder. We adopt this approach for PAS and let the LDPC decoder iterate with a SISO CC code decoder based on the forward backward (BCJR) algorithm to improve performance. For this purpose, we introduce the trellis of a CC code. The resulting decoder is a generalized LDPC (GLDPC) decoder [86] with a non-linear constraint.

Next, we briefly specify some components of the PAS transceiver that are important for joint decoding.

Channel Model For transmission we consider M-ASK over the AWGN channel, i.e., the output symbols of the channel are

$$\mathsf{Y} = \mathsf{X} + \mathsf{Z} \tag{5.1}$$

where Z is a Gaussian random variable with zero mean and variance σ^2. The

signal-to-noise ratio (SNR) is

$$\mathrm{SNR} = \frac{\mathbb{E}\left[\mathsf{X}^2\right]}{\sigma^2}. \tag{5.2}$$

Labeling Function An invertible labeling function b_X converts m bits to an M-ASK symbol. We use a BRGC [52] where b_1 decides the symbol's sign, i.e., we have

$$\beta_A(b_2, \cdots, b_m) = |\beta(0, b_2, \cdots, b_m)| = |\beta(1, b_2, \cdots, b_m)|. \tag{5.3}$$

The notation $b_{i,j}$ refers to the j-th bit of the i-th symbol x_i. We write $\boldsymbol{B}$ to refer to all bits $b_{i,j}$, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$.

Demodulation We consider a symbol-wise demodulator that is aware of the signal statistics P_{A}, P_{B_j}. The LLs $\tilde{L}_i(x)$ of the i-th transmitted symbol are

$$\tilde{L}_i(x) = \log(p_{\mathsf{Y}|\mathsf{X}}(\mathsf{Y}_i|\mathsf{X}_i = x) \cdot P_{\mathsf{X}}(x)), \forall x \in \mathcal{X}. \tag{5.4}$$

The demodulator calculates the bit-wise LLs

$$\tilde{L}_{i,j}(b) = \log(p_{\mathsf{Y}|\mathsf{B}_j}(\mathsf{Y}_i|\mathsf{B}_{i,j} = b) \cdot P_{\mathsf{B}_j}(b)), \forall b \in \{0, 1\}. \tag{5.5}$$

Thus, one symbol-channel splits into m parallel bit-channels [11]. The LLR of the j-th bit in the i-th transmitted symbol is

$$L_{i,j} = \tilde{L}_{i,j}(0) - \tilde{L}_{i,j}(1). \tag{5.6}$$

For convenience, we collect LLs and LLRs in the matrices $\tilde{\boldsymbol{L}}$ and $\boldsymbol{L}$, respectively. The (i, j)-th entry of the LL matrix $\tilde{\boldsymbol{L}}$ corresponds to $\tilde{L}_i(x_j)$. The (i, j)-th entry of the LLR matrix $\boldsymbol{L}$ corresponds to $L_{i,j}$.

LDPC Codes and Belief Propagation (BP) Decoding A (n, k) LDPC code [87] is a binary linear block code described by an $r \times n$ parity-check matrix $\boldsymbol{H}$ with entries $h_{i,j}$, $i = 1, 2, \ldots, r$, $j = 1, 2, \ldots, n$, where $r \geq n - k$. LDPC codes can be visualized through a bipartite graph also known as the Tanner graph $\mathcal{G}$. This graph consists of a set $\mathcal{V} = \{V_1, V_2, \cdots, V_n\}$ of n variable nodes, a set $\mathcal{C} = \{C_1, C_2, \cdots, C_r\}$ of r check nodes and a set $\mathcal{E} = \{e_{j,i}\}$ of edges. The check node C_j is connected to the variable node V_i through the edge $e_{j,i}$ if the entry $h_{i,j}$ of the parity-check matrix is one.

An LDPC BP decoder operates on LLRs [11]. Based on the channel observations $\boldsymbol{L}^{\mathsf{CH}}$, the LDPC decoder outputs the APP LLRs:

$$\boldsymbol{L}^{\mathsf{APP}} = \boldsymbol{L}^{\mathsf{CH}} + \boldsymbol{L}^{\mathsf{E,LDPC}} \tag{5.7}$$

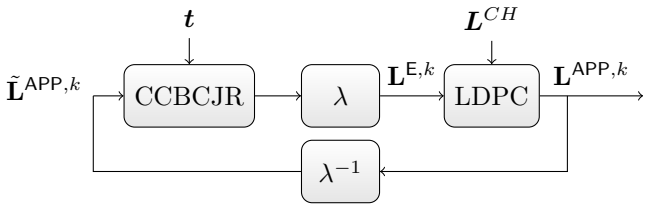


Figure 5.1: Symbol-based decoder.

where $\boldsymbol{L}^{\mathsf{E,LDPC}}$ denotes the extrinsic information.

5.1 Forward-Backward Algorithm for Constant Composition Codes

The BCJR algorithm [88], also known as the forward-backward algorithm, is a SISO algorithm that calculates the a posteriori symbol probabilities

$$P^{\mathsf{APP}}(a_i) = P(a_i|\tilde{\boldsymbol{L}}) \tag{5.8}$$

where $\tilde{\boldsymbol{L}}$ is a vector of LLs and a_i is the i-th transmitted symbol. From these probabilities, we can compute the extrinsic LLs $\tilde{\boldsymbol{L}}^{\mathrm{E}}$ [88]. For binary codes, the input interface may be LLRs, because we can convert easily from LLRs to LLs and vice versa. The constant composition BCJR (CCBCJR) decoder builds the code trellis from the type vector $\boldsymbol{t}$, i.e., it is a function

$$\text{CCBCJR} : \tilde{\boldsymbol{L}} \times \boldsymbol{t} \mapsto \tilde{\boldsymbol{L}}^{\mathsf{E}}. \tag{5.9}$$

5.2 Joint Decoder

We study how the decoder can exploit CC code properties to decrease the error probability.

5.2.1 Symbol-Based Decoder

The symbol-based decoder consists of a CCBCJR decoder and a LDPC decoder that exchange messages iteratively, see Fig. 5.1. The BCJR decoder has a symbol based interface, while the LDPC decoder has a bit based interface. The demodulator provides the LLs of the symbols and bit levels. The symbol-wise LLs are passed to the CCBCJR decoder. The LDPC decoder and the CCBCJR decoder iterate extrinsic information $\boldsymbol{L}^{\mathsf{E}}$. We use functions β and β^{-1} to convert from symbol based LLs to bit based LLRs

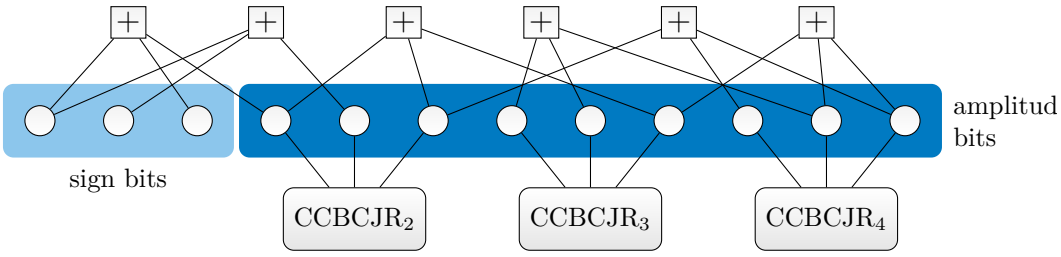


Figure 5.2: Tanner graph of naive bit-based decoder for $m = 4$ and $n = 3$. The amplitude bit variable nodes are connected to the respective BCJR node. The sign bit variable nodes are not connected to a BCJR node.

and vice versa. The function β converts LL into LLRs via

$$L_{i,j} = \log \left(\frac{\sum\limits_{x:\beta_j^{-1}(x)=0} \exp\left(\tilde{L}_i(x)\right)}{\sum\limits_{x:\beta_j^{-1}(x)=1} \exp\left(\tilde{L}_i(x)\right)} \right). \tag{5.10}$$

The function β^{-1} converts from bit-based LLRs to symbol-based LLs. For simplicity, we assume for a fixed i that the $\mathsf{B}_{i,j}$, $j = 1, 2, \ldots, m$, are pairwise independent given Y_i. The conversion is then

$$\tilde{L}_i(x) = \log \left(\prod_{j=2}^{m} \frac{\exp(L_{i,j} \cdot (1 - 2\beta_{A,j}^{-1}(x)))}{1 + \exp(L_{i,j} \cdot (1 - 2\beta_{A,j}^{-1}(x)))} \right). \tag{5.11}$$

Note that $1 - 2\beta_{A,j}^{-1}(x)$ is 1 for the bit 0 and -1 for the bit 1.

5.2.2 Bit-Based Decoder

The number of states and edges in the CCBCJR decoder increases exponentially with the alphabet size and polynomially in n. One idea to decrease complexity is to replace one $|\mathcal{A}|$-ary CCBCJR decoder by $\log_2 |\mathcal{A}|$ binary CCBCJR decoders. Additionally, the conversion functions β, β^{-1} become obsolete.

Consider a transmission sequence x^n with type constraint $\boldsymbol{t}$ on the amplitudes and its binary representation $\boldsymbol{B} \in \{0,1\}^{n \times m}$ according to the labeling function β, where the entry $b_{i,j}$ corresponds to the j-th bit of the i-th symbol. Let $\boldsymbol{b}_j^{\mathsf{I}} = b_{1,j}, b_{2,j}, \cdots b_{n,j}$ be the j-th column of $\boldsymbol{B}$, i.e., the j-th bit-level of the binary representation of the symbol sequence. Since x^n has a type $\boldsymbol{t}$ constraint on the amplitudes only, the sign bits are unconstrained. All other bit levels j, $2 \leq j \leq m$ are constrained. We derive the type constraint for each bit-level depending on the type $\boldsymbol{t}$ of the sequence x^n and the labeling function β. The number of zeros in bit-level j is equal to the number of amplitudes in

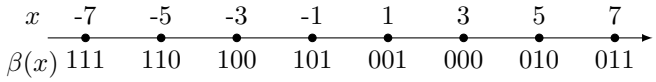


Figure 5.3: 8-QAM constellation with BRGC labeling β.

the sequence x^n whose binary representation is zero in the j-th position, i.e., we have

$$n_b(\boldsymbol{b}_j^{\downarrow}) = \sum_{\alpha\in\mathcal{A},\, \beta_j^{-1}(\alpha)=b} n_\alpha(\mathrm{amp}(x^n)) \tag{5.12}$$

where $\mathrm{amp}(x^n)$ is the element-wise absolute value of x^n, $b \in \{0,1\}$, and $2 \le j \le m$. Thus for one amplitude type constraint $\boldsymbol{t}$, we obtain $m-1$ bit constraints $\boldsymbol{t}_2, \cdots, \boldsymbol{t}_m$, where the index denotes the respective bit-level with

$$\boldsymbol{t}_j = \left[n_0(\boldsymbol{b}_j^{\downarrow}), n_1(\boldsymbol{b}_j^{\downarrow})\right]. \tag{5.13}$$

Example 5.1. Consider a sequence x^n with amplitude constraint $\boldsymbol{t} = [37, 20, 6, 1]$, i.e., 37 ones, 20 threes, 6 fives and 1 sevens, and the BRGC labeling β as shown in Fig. 5.3. We find $n_1(\boldsymbol{b}_2^{\downarrow}) = 7$ because the second bit of the labeling β is '1' for amplitudes 5 and 7 and they appear 6 times and once, respectively. The corresponding bit types $\boldsymbol{t}_2$ and $\boldsymbol{t}_3$ are

$$\boldsymbol{t}_2 = [57, 7] \tag{5.14}$$
$$\boldsymbol{t}_3 = [26, 38]. \tag{5.15}$$

For decoding, we add $m-1$ BCJR nodes into the Tanner graph, as shown in Fig. 5.2. Note that the bit-based CCBCJR decoders run independently. Their combined trellises allow sequences that do not fulfill the type constraint $\boldsymbol{t}$.

5.2.3 Improved Bit-Based Decoder

Each of the $m-1$ BCJR nodes is connected to n/m nodes. This suggests that the girth, i.e., the shortest cycle in the graph, is small. Loopy BP for small-girth was investigated in [89] and may lead to oscillations. There are two basic approaches to deal with this issue. First, we may filter the beliefs and thereby attenuate oscillations. Second, we could introduce multiple short length CC constraints on a bit-level, i.e., introduce lower degree CCBCJR nodes which increases both the girth and the rate-loss. Here, we consider only the first approach.

The LDPC decoder outputs the a posteriori LLRs $\boldsymbol{L}_j^{\mathsf{APP}}$. Based on the channel observation, the type vector $\boldsymbol{t}_j$ and a posteriori information, the j-th BCJR decoder CCBCJR_j generates the extrinsic information $\boldsymbol{L}_j^{\mathsf{E}}$. The outputs of the $m-1$ CCBCJRs are collected

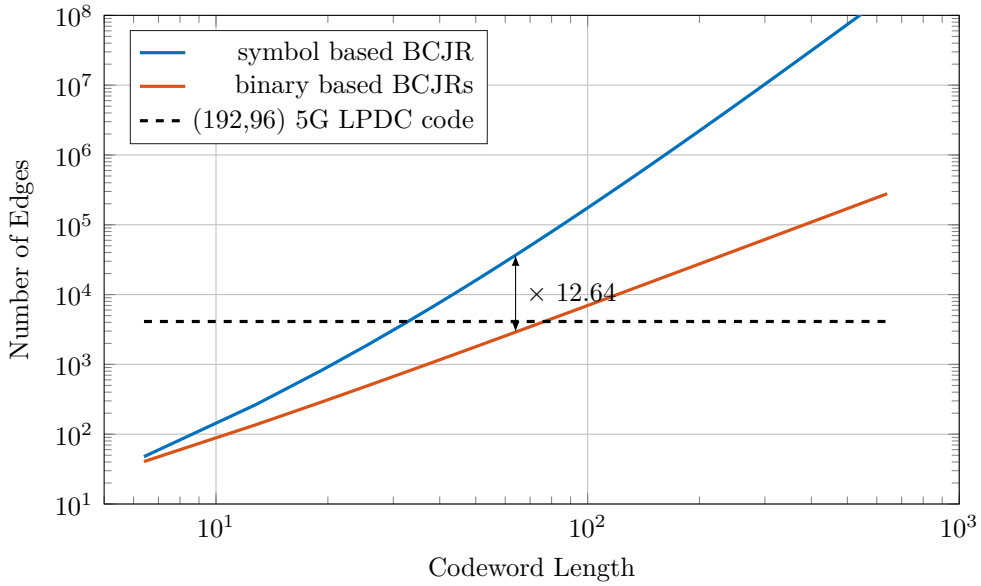


Figure 5.4: Number of branches to compute for the bit-based and symbol-based BCJR algorithms. The empirical distribution is $[37, 20, 6, 1]/64$. We interpret (5.17) and (5.18) as continuous functions. At an output length of 64 symbols, the symbol-based BCJR algorithm needs about 12.5 times more states than the binary-based BCJR algorithm. We compare with the number of branches of an iterative LDPC decoder using the BCJR algorithm.

in the matrix $\boldsymbol{L}^{\mathsf{E}}$. $\boldsymbol{L}$ and $\boldsymbol{L}^{\mathsf{E}}$ are then processed by the function

$$\mathrm{g}\left(\boldsymbol{L}^{\mathsf{CH}}, \boldsymbol{L}^{\mathsf{E}}, \boldsymbol{L}^{\mathsf{APP}}, k\right) \approx \boldsymbol{L}^{\mathsf{CH}} + \underbrace{\left(\mu \cdot \boldsymbol{L}^{\mathsf{E},k-1} + (1-\mu) \cdot \boldsymbol{L}^{\mathsf{E},k}\right)}_{\text{prior information}} \tag{5.16}$$

with $k \geq 1$ and $\mu \in [0, 1]$. After a number of iterations, the LDPC decoder outputs new a posteriori information, which is sent back to the CCBCJR decoders. The optimal parameter μ is found by grid search.

5.2.4 Computational Complexity Comparison

For the computational complexity analysis, we focus on the number E of edges in the code trellises, since the BCJR complexity is $\Theta(E)$[1] [90]. This analysis depends on the trellis representation of the CC code.

[1]We write $f(x) = \Theta(g(x))$ to express that $f(x)$ is bounded starting from x_0 on from below by $g(x)$ times a positive constant k_1 and from above by $g(x)$ times a positive constant k_2.

Symbol-Based Decoder

For a type $\boldsymbol{t} = [n_1, \cdots, n_{M/2}]$ constraint, we have

$$E_{\text{symb}} = \sum_{i=1}^{M/2} n_i \prod_{j \neq i} (n_j + 1) \tag{5.17}$$

branches. An increasing alphabet size even for the same block length may result in a large increase in the number of states and therefore the computational complexity. For a given empirical distribution, the number of states scales with the power of the support of the empirical distribution.

Bit-Based Decoder

For the bit-based decoder, we split one amplitude type constraint $\boldsymbol{t}$ into $m-1$ bit constraints $\boldsymbol{t}_2, \cdots, \boldsymbol{t}_m$. The number of edges is then

$$E_{\text{bit}} = \sum_{j=2}^{m} 2n_0(\boldsymbol{b}_j^{\downarrow}) n_1(\boldsymbol{b}_j^{\downarrow}) + n_0(\boldsymbol{b}_j^{\downarrow}) + n_1(\boldsymbol{b}_j^{\downarrow}). \tag{5.18}$$

In Fig. 5.4 we show the number of branches vs. the code word length for the empirical distribution $[37, 20, 6, 1]/64$. We also add the number of branches that are evaluated during one iteration of LDPC decoding of an (192,96) 5G LDPC code, i.e., we compute the number of branches of all single parity check and repetition nodes. Single parity check and repetition nodes have 4 times and 2 times the number of branches as their degree edges, respectively.

5.3 Simulation Results

We compare the performance of PAS with the bit-level decoder proposed in [11] with the symbol-based and the heuristically improved bit-based decoders with supplementary CC constrained nodes. We target a spectral efficiency of 1.5 bits per channel use with the 8-ASK constellation.

For encoding, we use a DM with type $\boldsymbol{t} = [37, 20, 6, 1]$ from Example 5.1 and a rate 3/4 code from the 5G eMBB standard [27] with block length 192. The reference LDPC decoder [11] is biased with the empirical distribution of the FEC input. The symbol-based decoder uses $\boldsymbol{t}$ and the bit-based decoder has two CCBCJRs with $\boldsymbol{t}_2 = [7, 57]$ and $\boldsymbol{t}_3 = [38, 26]$.

Simulation results in Fig. 5.5 show that the LDPC decoder with a linear combination of $\boldsymbol{L}^{\mathsf{E},\mathsf{LDPC},k-1}$ and $\boldsymbol{L}^{\mathsf{E},k}$ outperforms the LDPC decoder with $\boldsymbol{L}^{\mathsf{E},k}$ as prior information only. We include the performance of a (192,96) 5G LDPC code with an optimized interleaver as a non-shaped baseline with the same spectral efficiency. The bit-based

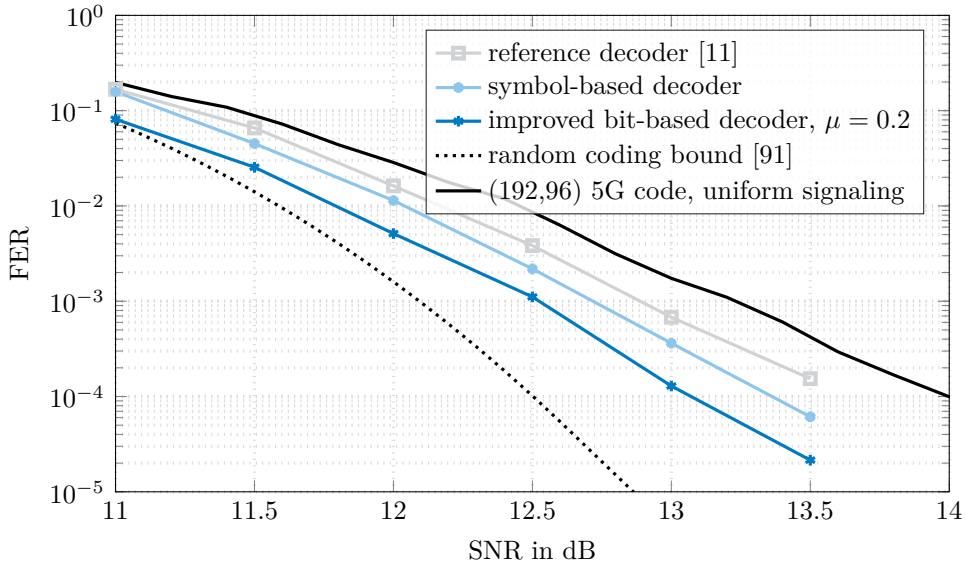


Figure 5.5: FER of the different strategies for 24 outer-iterations and 100 inner-iterations. We collected 100 erroneous frames per simulation point. The scheme is implemented by using 8−ASK with code rate 3/4 and block length $n = 192$. The rate-loss R_{loss} is about 0.145 bit/symbol.

decoding strategy gains 0.5dB in the simulation setup as compared to the LDPC decoder in [11].

6

Conclusions

We have studied distribution matchers (DMs) for energy efficient coded modulation (CM) and in particular probabilistic amplitude shaping (PAS). We summarize the main contributions of this thesis:

Chapter 3 focused on low energy sequences and showed achievable shaping gains for fixed-to-fixed length distribution matchers with finite block length. In particular, constraints on the order of an ASK modulation lead to a degradation of the shaping gain and the shaping improves for high transmission rates.

Chapter 4 introduced DMs formally and explained differences of variable length and fixed length DMs. Two DMs were introduced, i.e., constant composition DM (CCDM) and divergence optimal fixed length DM (SMDM). SMDMs approximate an independent and identically distributed (iid) target source with logarithmically growing divergence in the block length. We observed that divergence grows logarithmically for CCDM as well, but the constant term is worse so that CCDM is recommended only for long block lengths.

For fixed-to-fixed length DMs with the same code book, the divergence is independent of the mapping from messages to code words. It was shown how to use enumerative coding and shell mapping to index the optimal code book. In both cases generating functions build the foundation of the algorithms. Enumerative coding and a divide-and-conquer method are also used to instantiate CCDMs. Moreover, arithmetic coding is an elegant method to map messages to code words that permits encoding long block lengths. Since CCDM has a poor performance for short block lengths, product distribution matching (PDM) was proposed as a method that uses multiple binary DMs to approximate a distribution with large support. For product distributions, the pre-log factor of divergence can be reduced

significantly. For non-product distributions, two algorithms are proposed that approximate these distributions. Unfortunately, asymptotically, the divergence grows linearly in the block length.

Chapter 5 tackles the performance of CCDM at the receiver side. The CCDM code book is interpreted as a supplementary code that can assist in the decoding process by adding a further node to the Tanner graph of an LDPC code. This additional node is computationally complex, hence, it is split into multiple binary nodes which reduces complexity significantly. A heuristically improved decoder gains up to 0.5 dB in transmit power as compared to a decoder that uses only a bias that reflects the amplitude distribution.

All presented methods, except PDM that approximates non-product distributions, have a vanishing normalized divergence to the target distribution for long blocks. This was required for energy efficient communications. The PDM that approximates non-product dsitributions has a penalty that depends on the target distribution and is negligible for energy efficient communications. However, for stealth communication, even the the optimal distribution matcher (SMDM) is infeasible because divergence grows logarithmically and therefore the communication channel can be discovered. It remains to relax some restrictions of the DMs such that they remain practical, but their divergence decreases. We suggest to allow many-to-one mappings into disjoint sets with a supplementary source of randomness that does not carry information.

Bibliography

[1] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423 and 623–656, Jul. and Oct. 1948.

[2] J. Forney, G., R. Gallager, G. Lang, F. Longstaff, and S. Qureshi, "Efficient modulation for band-limited channels," *IEEE J. Sel. Areas Commun.*, vol. 2, no. 5, pp. 632–647, Sep. 1984.

[3] G. Ungerböck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inf. Theory*, vol. 28, no. 1, pp. 55–67, Jan. 1982.

[4] A. K. Khandani and P. Kabal, "Shaping multidimensional signal spaces. I. Optimum shaping, shell mapping," *IEEE Trans. Inf. Theory*, vol. 39, no. 6, pp. 1799–1808, 1993.

[5] ——, "Shaping multidimensional signal spaces. ii. shell-addressed constellations," *IEEE Trans. Inf. Theory*, vol. 39, no. 6, pp. 1809–1819, 1993.

[6] F. R. Kschischang and S. Pasupathy, "Optimal nonuniform signaling for Gaussian channels," *IEEE Trans. Inf. Theory*, vol. 39, no. 3, pp. 913–929, May 1993.

[7] P. Fortier, A. Ruiz, and J. M. Cioffi, "Multidimensional signal sets through the shell construction for parallel channels," *IEEE Trans. Commun.*, vol. 40, no. 3, pp. 500–512, 1992.

[8] ITU, "A modem operating at data signalling rates of up to 33 600 bit/s for use on the general switched telephone network and on leased point-to-point 2-wire telephone-type circuits," 1998.

[9] *Digital Video Broadcasting (DVB); 2nd Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and Other Broadband Satellite Applications (DVB-S2)*, European Telecommun. Standards Inst. (ETSI) Std. EN 302 307, Rev. 1.2.1, 2009.

[10] *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part 2: DVB-S2 Extensions (DVB-S2X)*, European Telecommun. Standards Inst. (ETSI) Std. EN 302 307-2, Rev. 1.1.1, 2014.

[11] G. Böcherer, F. Steiner, and P. Schulte, "Bandwidth efficient and rate-matched low-density parity-check coded modulation," *IEEE Trans. Commun.*, vol. 63, no. 12, pp. 4651–4665, Dec. 2015.

[12] W. G. Bliss, "Circuitry for performing error correction calculations on baseband encoded data to eliminate error propagation," *IBM Techn. Discl. Bull.*, no. 23, Mar. 1981.

[13] A. R. Calderbank and L. H. Ozarow, "Nonequiprobable signaling on the gaussian channel," *IEEE Trans. Inf. Theory*, vol. 36, no. 4, pp. 726–740, Jul. 1990.

[14] G. Ungerböck, "Huffman shaping," in *Codes, Graphs, and Systems*, R. Blahut and R. Koetter, Eds. Springer, 2002, ch. 17, pp. 299–313.

[15] G. Böcherer and R. Mathar, "Matching dyadic distributions to channels," in *Proc. Data Compression Conf.*, Mar. 2011, pp. 23–32.

[16] R. A. Amjad and G. Böcherer, "Fixed-to-variable length distribution matching," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2013.

[17] N. Cai, S.-W. Ho, and R. Yeung, "Probabilistic capacity and optimal coding for asynchronous channel," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Sep. 2007, pp. 54–59.

[18] S. Baur and G. Böcherer, "Arithmetic distribution matching," in *Proc. Int. ITG Conf. Syst. Commun. Coding*, Hamburg, Germany, Feb. 2015, pp. 1–6.

[19] R. A. Amjad, "Algorithms for simulation of discrete memoryless sources," Master's thesis, Technical University of Munich, Institute for Communications Engineering, 2013.

[20] M. Mondelli, S. H. Hassani, and R. Urbanke, "How to achieve the capacity of asymmetric channels," in *Proc. Allerton Conf. Commun., Contr., Comput.*, Monticello, IL, USA, Sep. 2014, pp. 789–796.

[21] P. Schulte, "Zero error fixed length distribution matching," Master's thesis, Technical University of Munich, Institute for Communications Engineering, 2014.

[22] T. V. Ramabadran, "A coding scheme for m-out-of-n codes," *IEEE Trans. Commun.*, vol. 38, no. 8, pp. 1156–1163, Aug. 1990.

[23] M. Pikus, W. Xu, and G. Kramer, "Finite-precision implementation of arithmetic coding based distribution matchers," *arXiv preprint arXiv:1907.12066*, 2019.

[24] T. Fehenberger, D. S. Millar, T. Koike-Akino, K. Kojima, and K. Parsons, "Parallel-amplitude architecture and subset ranking for fast distribution matching," *arXiv preprint arXiv:1902.08556*, 2019.

[25] P. Yuan, G. Böcherer, P. Schulte, G. Kramer, R. Böhnke, and W. Xu, "Error detection using symbol distribution in a system with distribution matching and probabilistic amplitude shaping," German EP Application, 10 31, 2016.

[26] K. Niu and K. Chen, "Crc-aided decoding of polar codes," *IEEE Commun. Lett.*, vol. 16, no. 10, pp. 1668–1671, 2012.

[27] T. Richardson and S. Kudekar, "Design of Low-Density Parity Check Codes for 5G New Radio," *IEEE Commun. Mag.*, vol. 56, no. 3, pp. 28–34, Mar. 2018.

[28] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. John Wiley & Sons, Inc., 2006.

[29] I. Csiszár and P. C. Shields, "Information theory and statistics: A tutorial," *Foundations and Trends® in Commun. Inf. Theory*, vol. 1, no. 4, pp. 417–528, 2004.

[30] R. B. Ash, *Information Theory*. New York: Dover, 1965.

[31] R. R. Bahadur, "Some approximations to the binomial distribution function," *Ann. Math. Statistics*, pp. 43–54, 1960.

[32] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, 2nd ed. AIP, 1989.

[33] T. Cover, "Enumerative source encoding," *IEEE Trans. Inf. Theory*, vol. 19, no. 1, pp. 73–77, Jan. 1973.

[34] J. Schalkwijk, "An algorithm for source coding," *IEEE Trans. Inf. Theory*, vol. 18, no. 3, pp. 395–399, May 1972.

[35] T. J. Lynch, "Sequence time coding for data compression," *Proc. IEEE*, vol. 54, no. 10, pp. 1490–1491, 1966.

[36] J. Sayir, "On coding by probability transformation," Ph.D. dissertation, ETH Zürich, 1999.

[37] R. C. Pasco, "Source coding algorithms for fast data compression," Ph.D. dissertation, Stanford University CA, 1976.

[38] J. J. Rissanen, "Generalized Kraft inequality and arithmetic coding," *IBM Journal of research and development*, vol. 20, no. 3, pp. 198–203, May 1976.

[39] M. Hoshi *et al.*, "Interval algorithm for homophonic coding," *IEEE Trans. Inf. Theory*, vol. 47, no. 3, pp. 1021–1031, 2001.

[40] J. Honda and H. Yamamoto, "Variable-to-fixed length homophonic coding with a modified shannon-fano-elias code," in *Proc. Int. Symp. Inf. Theory and its Applicat. (ISITA)*. IEEE, Oct. 2016, pp. 11–15.

[41] R. Blahut, "Computation of channel capacity and rate-distortion functions," *IEEE Trans. Inf. Theory*, vol. 18, no. 4, pp. 460–473, Jul. 1972.

[42] S. Arimoto, "An algorithm for computing the capacity of arbitrary discrete memoryless channels," *IEEE Trans. Inf. Theory*, vol. 18, no. 1, pp. 14–20, Jan. 1972.

[43] J. Hou and G. Kramer, "Effective secrecy: Reliability, confusion and stealth," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2014, pp. 601–605.

[44] F. Steiner and G. Böcherer, "Comparison of geometric and probabilistic shaping with application to ATSC 3.0," in *Proc. Int. ITG Conf. Syst. Commun. Coding*. VDE, Feb. 2017, pp. 1–6.

[45] *Physical Layer Protocol*, Advanced Television Systems Committee, Washington, DC, USA, Dec. 2016.

[46] A. Martinez, A. Guillén i Fàbregas, G. Caire, and F. Willems, "Bit-interleaved coded modulation revisited: A mismatched decoding perspective," *IEEE Trans. Inf. Theory*, vol. 55, no. 6, pp. 2756–2765, Jun. 2009.

[47] U. Wachsmann, R. F. H. Fischer, and J. B. Huber, "Multilevel codes: theoretical concepts and practical design rules," *IEEE Trans. Inf. Theory*, vol. 45, no. 5, pp. 1361–1391, Jul. 1999.

[48] H. Imai and S. Hirakawa, "A new multilevel coding method using error-correcting codes," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 371–377, May 1977.

[49] E. Zehavi, "8-PSK trellis codes for a Rayleigh channel," *IEEE Trans. Commun.*, vol. 40, no. 5, pp. 873–884, May 1992.

[50] R. F. H. Fischer, *Precoding and Signal Shaping for Digital Transmission*. John Wiley & Sons, Inc., 2002.

[51] G. Böcherer, "Capacity-achieving probabilistic shaping for noisy and noiseless channels," Ph.D. dissertation, RWTH Aachen University, 2012. [Online]. Available: http://www.georg-boecherer.de/capacityAchievingShaping.pdf

[52] F. Gray, "Pulse code communication," U.S. Patent 2 632 058, 1953.

[53] F. Buchali, F. Steiner, G. Böcherer, L. Schmalen, P. Schulte, and W. Idler, "Rate adaptation and reach increase by probabilistically shaped 64-QAM: An experimental demonstration," *J. Lightw. Technol.*, vol. 34, no. 8, Apr. 2016.

[54] J. Cho and P. J. Winzer, "Probabilistic constellation shaping for optical fiber communications," *J. Lightw. Technol.*, vol. 37, no. 6, pp. 1590–1607, Feb. 2019.

[55] M. Pikus and W. Xu, "Arithmetic coding based multi-composition codes for bit-level distribution matching," *arXiv preprint arXiv:1904.01819*, 2019.

[56] T. Fehenberger, D. Millar, T. Koike-Akino, K. Kojima, and K. Parsons, "Partition-based distribution matching," *arXiv preprint*, 2018.

[57] T. Fehenberger, D. S. Millar, T. Koike-Akino, K. Kojima, and K. Parsons, "Multiset-partition distribution matching," *IEEE Trans. Commun.*, vol. 67, no. 3, pp. 1885–1893, Nov. 2018.

[58] Y. C. Gültekin, W. J. van Houtum, S. Şerbetli, and F. M. Willems, "Constellation shaping for IEEE 802.11," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE, Oct. 2017, pp. 1–7.

[59] Y. C. Gültekin, W. van Houtum, and F. Willems, "On constellation shaping for short block lengths," in *Proc. Symp. Inf. Theory Signal Process. Benelux*, Jun. 2018, pp. 1–11.

[60] Y. C. Gültekin, F. M. Willems, W. J. van Houtum, and S. Şerbetli, "Approximate enumerative sphere shaping," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*. IEEE, Aug. 2018, pp. 676–680.

[61] T. Yoshida, M. Karlsson, and E. Agrell, "Hierarchical distribution matching for probabilistically shaped coded modulation," *J. Lightw. Technol.*, vol. 37, no. 6, pp. 1579–1589, Jan. 2019.

[62] J. Cho, "Prefix-free code distribution matching for probabilistic constellation shaping," *IEEE Trans. Commun.*, 2019, early access.

[63] J. Honda and H. Yamamoto, "Polar coding without alphabet extension for asymmetric models," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 7829–7838, Dec. 2013.

[64] O. İşcan, R. Böhnke, and W. Xu, "Shaped polar codes for higher order modulation," *IEEE Communications Letters*, vol. 22, no. 2, pp. 252–255, Oct. 2017.

[65] N. Stolte, "Rekursive Codes mit der Plotkin-Konstruktion und ihre Decodierung," Ph.D. dissertation, TU Darmstadt, 2002.

[66] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.

[67] T. Wiegart, F. Steiner, P. Schulte, and P. Yuan, "Shaped on-off keying using polar codes," *IEEE Communications Letters*, vol. 23, no. 11, pp. 1922–1926, Nov. 2019.

[68] T. S. Han and S. Verdu, "Approximation theory of output statistics," *IEEE Trans. Inf. Theory*, vol. 39, no. 3, pp. 752–772, Jan. 1993.

[69] G. Böcherer and R. A. Amjad, "Informational divergence and entropy rate on rooted trees with probabilities," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aug. 2014, pp. 176–180.

[70] F. Steiner, P. Schulte, and G. Böcherer, "Approaching waterfilling capacity of parallel channels by higher order modulation and probabilistic amplitude shaping," in *Proc. IEEE Conf. on Inf. Sci. and Sys.*, May 2018, pp. 1–6.

[71] G. Böcherer, P. Schulte, and F. Steiner, "High throughput probabilistic shaping with product distribution matching," *arXiv preprint*, 2017. [Online]. Available: http://arxiv.org/abs/1702.07510

[72] R. Laroia, N. Farvardin, and S. A. Tretter, "On optimal shaping of multidimensional constellations," *IEEE Trans. Inf. Theory*, vol. 40, no. 4, pp. 1044–1056, Jul. 1994.

[73] R. F. H. Fischer, "Calculation of shell frequency distributions obtained with shell-mapping schemes," *IEEE Trans. Inf. Theory*, vol. 45, no. 5, pp. 1631–1639, Jul. 1999.

[74] P. Flajolet and R. Sedgewick, *Analytic combinatorics*. Cambridge University Press, 2009.

[75] G. R. Lang and F. M. Longstaff, "A Leech lattice modem," *IEEE J. Sel. Areas Commun.*, vol. 7, no. 6, pp. 968–973, Aug. 1989.

[76] S. A. Tretter, *Constellation Shaping, Nonlinear Precoding, and Trellis Coding for Voiceband Telephone Channel Modems: With Emphasis on ITU-T Recommendation*. Springer Science & Business Media, 2012, vol. 34.

[77] F. Willems and J. Wuijts, "A pragmatic approach to shaped coded modulation," in *Proc. IEEE Symp. Commun. Veh. Technol. Benelux*, 1993.

[78] G. Böcherer and B. C. Geiger, "Optimal quantization for distribution synthesis," *IEEE Trans. Inf. Theory*, vol. 62, no. 11, pp. 6162–6172, Nov. 2016.

[79] K. Sayood, *Introduction to data compression*. Elsevier, 2006.

[80] P. Schulte, F. Steiner, and G. Böcherer, "shapecomm WebDM: Online constant composition distribution matcher," http://dm.shapecomm.de, Jul. 2017.

[81] G. Böcherer, F. Steiner, and P. Schulte, "Methods of converting or reconverting a data signal and method and system for data transmission and/or data reception," German EP Application EP3 306 821A1, 10 5, 2016.

[82] M. Pikus and W. Xu, "Bit-level probabilistically shaped coded modulation," *IEEE Commun. Lett.*, vol. 21, no. 9, pp. 1929–1932, Sep. 2017.

[83] F. Steiner, F. Da Ros, M. P. Yankov, G. Böcherer, P. Schulte, S. Forchhammer, and G. Kramer, "Experimental verification of rate flexibility and probabilistic shaping by 4D signaling," in *Proc. Optical Fiber Commun. Conf.*, 2018, paper M4E3.

[84] J. L. Fan and J. M. Cioffi, "Constrained coding techniques for soft iterative decoders," in *IEEE Global Telecommun. Conf. (GLOBECOM)*, vol. 1. Rio de Janeireo, Brazil: IEEE, Dec. 1999, pp. 723–727.

[85] A. P. Hekstra, "Use of a *d*-constraint during LDPC decoding in a Bliss scheme," *arXiv preprint arXiv:0707.3925*, 2007.

[86] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.

[87] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[88] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Trans. Inf. Theory*, vol. 20, no. 2, pp. 284–287, Mar. 1974.

[89] K. P. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study," in *Proc. Conf. on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1999, pp. 467–475.

[90] R. J. McEliece, "On the BCJR trellis for linear block codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 4, pp. 1072–1092, Jul. 1996.

[91] G. Liva and F. Steiner, "pretty-good-codes.org: Online library of good channel codes," http://pretty-good-codes.org, Oct. 2017.