Design Techniques for Efficient Sparse Regression Codes

Adam Greig

Selwyn College
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

September 2017

Design Techniques for Efficient Sparse Regression Codes

Adam Greig

Summary

Sparse regression codes (SPARCs) are a recently introduced coding scheme for the additive white Gaussian noise channel, for which polynomial time decoding algorithms have been proposed which provably achieve the Shannon channel capacity. One such algorithm is the approximate message passing (AMP) decoder. However, directly implementing these decoders does not yield good empirical performance at practical block lengths. This thesis develops techniques for improving both the error rate performance, and the time and memory complexity, of the AMP decoder. It focuses on practical and efficient implementations for both single- and multi-user scenarios.

A key design parameter for SPARCs is the power allocation, which is a vector of coefficients which determines how codewords are constructed. In this thesis, novel power allocation schemes are proposed which result in several orders of magnitude improvement to error rate compared to previous designs. Further improvements to error rate come from investigating the role of other SPARC construction parameters, and from performing an online estimation of a key AMP parameter instead of using a pre-computed value.

Another significant improvement to error rates comes from a novel three-stage decoder which combines SPARCs with an outer code based on low-density parity-check codes. This construction protects only vulnerable sections of the SPARC codeword with the outer code, minimising the impact to the code rate. The combination provides a sharp waterfall in bit error rates and very low overall codeword error rates.

Two changes to the basic SPARC structure are proposed to reduce computational and memory complexity. First, the design matrix is replaced with an efficient in-place transform based on Hadamard matrices, which dramatically reduces the overall decoder time and memory complexity with no impact on error rate. Second, an alternative SPARC design is developed, called Modulated SPARCs. These are shown to also achieve the Shannon channel capacity, while obtaining similar empirical error rates to the original SPARC, and permitting a further reduction in time and memory complexity.

Finally, SPARCs are implemented for the broadcast and multiple access channels, and for the multiple description and Wyner-Ziv source coding models. Designs for appropriate power allocations and decoding strategies are proposed and are found to give good empirical results, demonstrating that SPARCs are also well suited to these multi-user settings.

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text.

It is not substantially the same as any that I have submitted, or, is being concurrently submitted for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my dissertation has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University of similar institution except as declared in the Preface and specified in the text.

It does not exceed the prescribed word limit of 65,000 words inclusive of appendices, bibliography, footnotes, tables, and equations, and has fewer than 150 figures.

Adam Greig
September 2017

Acknowledgements

This work was generously funded by a Doctoral Training Award from the Engineering and Physical Sciences Research Council.

My sincere gratitude is owed to Dr. Ramji Venkataramanan, first for offering to take me on as a student, and then for four years of patient, dedicated, and careful supervision and guidance. His enthusiasm and support fostered the ideas herein, while his tireless attention to detail caught my countless missteps along the way.

My thanks also go to my colleagues in the Signal Processing and Communications Laboratory for their camaraderie, and especially to Dr. Jossy Sayir for always having my back.

Thanks finally to friends and family. To friends who left Cambridge: thank you for showing it can be done, and for offering support from the world outside. To Rob and David, for sharing in the graduate experience, and especially David, for also suffering sharing a house for most of it. To Hannah, for their constant encouragement, for putting up with the late nights, and for everything else besides. Last but not least, to my parents and sister, for their unwavering and enduring support over this and all other endeavours.

Preface

Material from Chapters 1 and 2 represents work undertaken in collaboration with Dr. Ramji Venkataramanan and Dr. Cynthia Rush, and has been previously published in [1].

Material from Chapters 3, 4, 5, and 6 represents work undertaken in collaboration with Dr. Ramji Venkataramanan. The author of this thesis was the principal contributor.

Material from Chapters 3 and 4 has been published in [2].

Contents

Chapter 1

Introduction

Claude Shannon's seminal 1948 paper [3] introduced the concepts of channel and source coding, and at the same time presented sharp limits on the possible performance of these techniques. Since then, much work has gone into finding computationally efficient techniques for achieving these bounds. This thesis presents research into the design and implementation of sparse regression codes, which offer efficient solutions to these problems for Gaussian models.

1.1 Channel Coding

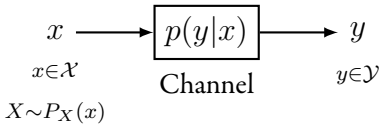


Figure 1.1: A memoryless communications channel

A memoryless communications channel, depicted in Figure 1.1, maps a symbol x from the input alphabet $\mathcal{X}$ to a symbol y from the output alphabet $\mathcal{Y}$, and is defined by its transition probability $p(y|x)$. The *capacity* is defined as the maximum mutual information between X and Y over all possible input distributions: $\mathcal{C} = \sup_{P(x)} I(X;Y)$ [4]. The channel may be discrete, in which case $\mathcal{X}$ and $\mathcal{Y}$ are finite alphabets, or it may be continuous, such as with $\mathcal{X} = \mathcal{Y} = \mathbb{R}$. The memoryless property requires that $p(y|x)$ depends only on the current x and not previous channel inputs or outputs [4]. This channel model is a useful representation of many physical communication mediums where we would like to achieve reliable commu-

nication (that is, with arbitrarily low probability P_e of an erroneous message being received). The channel is *used* by transmitting a symbol x to a receiver who obtains the symbol y. We define the channel *rate* R to be the information communicated per channel use: if it takes n channel uses to communicate B bits of information, then $R = \frac{B}{n}$.

Shannon gave a proof [3] that for any $\epsilon > 0$ and $R < \mathcal{C}$, it is possible to achieve $P_e \leq \epsilon$, and conversely for $R > \mathcal{C}$ the probability of error can not be made arbitrarily small. A stronger version of this important result was later proven for the discrete memoryless channel by Feinstein [5].

Additive White Gaussian Noise Channel

A more specific channel which will be of particular interest is the additive white Gaussian noise (AWGN) channel. The channel generates output sequence y from input x according to

$$y = x + w \tag{1.1}$$

where the independent noise w is a Gaussian random variable with zero mean and variance σ^2. There is an average power constraint P on the input x: if $x_1, \ldots, x_n$ are transmitted over n uses of the channel, then we require that $\frac{1}{n}\sum_{i=1}^{n} x_i^2 \leq P$. The signal-to-noise ratio $\frac{P}{\sigma^2}$ is denoted by snr. The capacity of this channel is $\mathcal{C} = \frac{1}{2}\log(1 + \mathsf{snr})$. The AWGN channel is important both because it is amenable to analysis, and because it is an accurate model for many practical communication channels.

Channel Codes

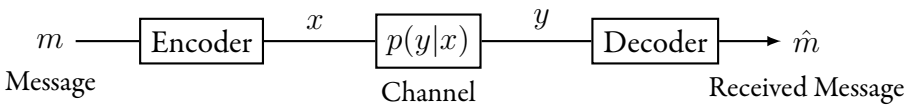


Figure 1.2: The standard channel coding set-up

To achieve these capacity limits, we typically use some form of channel code. The general set up is illustrated in Figure 1.2. Some message m is encoded to a codeword x which is transmitted through the channel, which produces the output y. The decoder must estimate $\hat{m}$ from y. A common form of channel code is the block code, which may be viewed as a codebook of codewords, any of which may be selected for transmission over the channel, and the

index of the chosen codeword represents the message to be communicated. The codewords are sequences (of length n) of symbols from the channel input alphabet, and so each message communicates nR bits.

A simple way to construct these codebooks is the Shannon-style random codebook, where codewords are randomly selected sequences from the input alphabet, drawn according to the input distribution $P(x)$. For the AWGN channel described above, we generate codewords as i.i.d. Gaussian random variates, with each codeword entry distributed as $\mathcal{N}(0, P)$. To decode, we select whichever codeword has the highest likelihood of having being transmitted, given the channel output sequence y. This decoder is called the maximum likelihood decoder and while it is the optimal decoder for this problem [6], it has been largely avoided in practice due to the high decoding complexity. The number of possible codewords is exponential in the block size n, and we require exhaustive search to determine the codeword which is closest to y. This is typically impractical, so in practice alternative code structures are used which permit lower complexity decoding.

Practical Channel Codes

Current state of the art codes can come very close to the Shannon capacity, both theoretically under certain channel models, and in practice in real world communication systems. One such example is low density parity check codes, first invented by Gallager in 1962 [7] and later independently rediscovered by MacKay and Neal in 1993 [8]. LDPC codes have been shown to achieve capacity for the binary erasure channel [9]. These codes work very well on channels with discrete input alphabets, particularly those with binary input. Other examples of capacity-achieving codes for discrete-input channels are turbo codes [10, 11], and polar codes [12, 13].

However, physical communications channels have continuous valued inputs, such as the additive white Gaussian noise (AWGN) channel. To employ an LDPC or Turbo code on such a channel requires the addition of a modulation scheme, such as Phase Shift Keying (PSK) or Quadrature Amplitude Modulation (QAM). This combination of a binary error-correcting code with a standard modulation scheme is known as coded modulation [14, 15]. Though coded modulation schemes have good empirical performance, they have not been proven to be capacity-achieving for the AWGN channel. A survey of progress on coded modulation techniques by Costello and Forney can be found in [16].

Finding an efficient code which achieves capacity while directly using the AWGN channel without a modulation step is a problem for which sparse superposition codes were in-

troduced by Barron and Joseph in [17, 18]. Sparse superposition codes, or sparse regression codes (SPARCs), have been shown to be asymptotically capacity achieving with an appropriate polynomial-time decoder [18]. They combine the encoding and modulation steps into a single process, similar to the Shannon random codebooks, but permit an efficient, low-complexity encoder and decoder, making them promising as an alternative to coded modulation. The SPARC construction is discussed in more detail in Section 1.4.

Existing lattice codes can also be used to code directly on the AWGN channel, but require infeasible computation at the very high dimensions required to come close to capacity [19,20].

1.2 Source Coding

Source coding is the complementary problem to channel coding: rather than attempting to add redundancy to a message so that it may be recovered after sustaining an error, we wish to remove the redundancy already present in some source. This may be performed such that we may still perfectly recover the source at a later date, known as lossless compression, for example when a text document is compressed on a computer; or we may introduce errors, known as lossy compression, typically with a continuous source such as MP3 encoding an audio file.

The latter case is described by rate distortion theory, introduced by Shannon in [3] and developed further in [21]. As in the channel coding situation, we replace the actual source with a codeword selected from a set which is known in advance to both the encoder and decoder. The selected codeword does not perfectly represent the real source, and the error is called the *distortion*. For a source x and our approximation of it $\hat{x}$, the distortion is $d(x, \hat{x})$ for some metric $d(\cdot, \cdot)$. One metric commonly used for distortion is the squared error,

$$d(x, \hat{x}) = (x - \hat{x})^2. \tag{1.2}$$

For a vector source and suitable distortion metric we define the distortion as

$$d(x^n, \hat{x}^n) = \frac{1}{n} \sum_{i=1}^{n} d(x_i, \hat{x}_i). \tag{1.3}$$

Rate distortion theory is concerned with the minimum level of distortion obtainable for a given rate R used to encode a sample. For a specified distortion metric, a rate-distortion pair (R, D) is called *achievable* if a code exists for the rate R that gives an average distortion less

than or equal to D. The set of all such achievable rate-distortion pairs is called the *achievable region*, and the function that maps a target distortion to the required rate is the *rate-distortion function* $R(D)$.

For a vector source with n elements, the index of each codeword will be represented by nR bits. Such a code is called a $(2^{nR}, n)$ rate distortion code. Given an input alphabet $\mathcal{X}$, our encoder and decoder are therefore mappings

$$f_n : \mathcal{X}^n \to \{1, 2, \ldots, 2^{nR}\}$$

$$g_n : \{1, 2, \ldots, 2^{nR}\} \to \mathcal{X}^n, \tag{1.4}$$

and the distortion associated with this code is

$$D = \mathbb{E}[d(X^n, g_n(f_n(X^n)))] = \sum_{x^n} p(x^n) d(x^n, g_n(f_n(x^n))). \tag{1.5}$$

A useful result is the rate distortion theory for a Gaussian source, as these will be of particular interest in the rest of this thesis. Shannon [21] showed that for a Gaussian source with variance σ^2, and $D \leq \sigma^2$,

$$R(D) = \frac{1}{2} \log \frac{\sigma^2}{D}. \tag{1.6}$$

For the case $D > \sigma^2$, the distortion is trivially achievable by encoding all zeros, and no rate is required, $R(D) = 0$. As with the channel codes described previously, one conceptually simple way to achieve this bound is using Shannon-style random codebooks. For a rate R, we construct a codebook by drawing 2^{nR} samples from the input source distribution, and then to encode we select whichever entry has the lowest distortion for the source. The index of that entry is transmitted, requiring nR bits, and the decoder looks up that index to find the approximation to the source. This technique is not practical in reality as the codebook size grows exponentially in n, so alternative techniques must be found. SPARCs provide one option, discussed in more detail in Section 1.4.

Another way to view source coding in the rate-distortion framework is as a process called *quantisation*. The source X is quantised to some representation $\hat{X}$, where all possible $\hat{X}$ are known in advance. The resulting error is called the *quantisation noise*. We can model the process as

$$X = \hat{X} + Z, \tag{1.7}$$

which is known as the test channel, and is very similar to the AWGN channel described above, except that the source X is the *output* of this channel. This quantisation view is especially useful for designing techniques to solve the multiple description problem described in Section 6.3.

1.3 Multiuser Communication

The source and channel coding scenarios described above involve simple point-to-point problems, with a single encoder and decoder. However, there exist many problems involving multiple encoders or decoders, and the study of these problems is called network information theory. Many practical communications problems involve multiple parties, such as multiple mobile telephones connected to the same base station, or submarine communications cables which require amplifiers along their length.

Often for these more complicated problems the general capacity region is not known, and therefore no techniques are known which can achieve capacity. For some specific channels, a capacity region or a bound on the capacity region may be known, and it may then be possible to find techniques to achieve these regions. In many cases, capacity regions are known for Gaussian channels, which often leads to techniques to achieve them based on the same Shannon-style random codebooks discussed previously, which are infeasible for practical purposes.

1.4 The Sparse Superposition Code

Sparse superposition codes, or sparse regression codes (SPARCs) are a recent development, initially proposed by Barron and Joseph [17, 18] for use on the AWGN channel. Unlike with coded modulation, SPARCs directly map the message to be encoded onto a codeword for the AWGN channel. They are similar in essence to Shannon random codebooks: codewords are sequences of normally distributed random variates, but instead of being selected from an unstructured codebook of 2^{nR} such codewords, they are found as a linear combination of codewords from many smaller codebooks. Specifically, they are formed as sparse linear combinations of the columns of a design matrix. This key difference introduces structure to the codes, which enables much lower complexity decoders. SPARCs can be used for both channel and source coding operations: the underlying design matrix and codeword structure is the same, but different encoders and decoders are used.

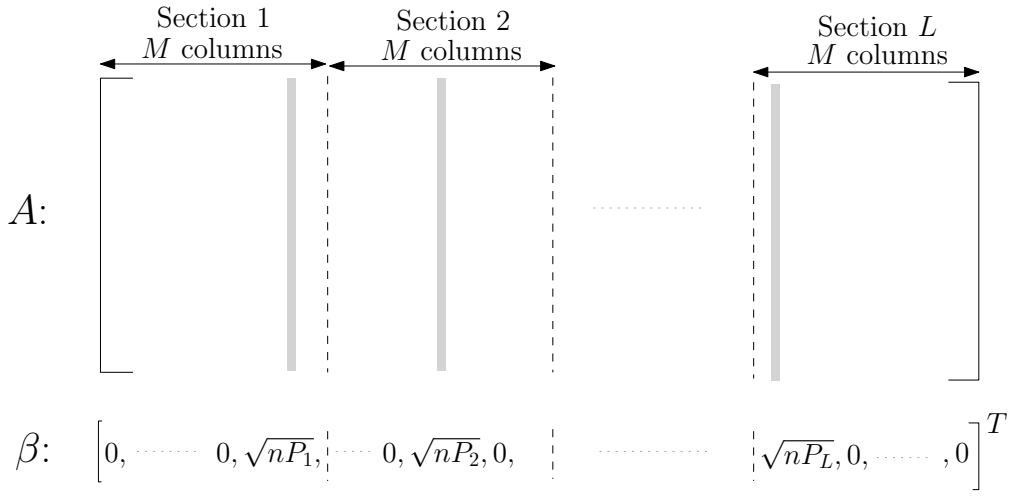


Figure 1.3: A is the $n \times LM$ design matrix, β is an $ML \times 1$ sparse vector with one non-zero in each of the L sections. The length-n codeword is $A\beta$. The message determines the locations of the non-zeros in β, while $P_1, \ldots, P_L$ are fixed a priori. This figure is reproduced from [1].

1.4.1 Construction and Encoding

A SPARC is defined in terms of a design matrix A of dimension $n \times ML$. Here n is the block length, and M, L are integers which are specified below in terms of n and the rate R. As shown in Fig. 1.3, the design matrix A has L sections with M columns each. In the original construction of [17, 18] and in the theoretical analysis in [1, 22–24], the entries of A are assumed to be i.i.d. Gaussian $\sim \mathcal{N}(0, 1/n)$.

Codewords are constructed as sparse linear combinations of the columns of A. In particular, a codeword is of the form $A\beta$, where β is an $ML \times 1$ vector $(\beta_1, \ldots, \beta_{ML})$ with the property that there is exactly one non-zero β_j for the section $1 \leq j \leq M$, one non-zero β_j for the section $M+1 \leq j \leq 2M$, and so forth. The non-zero value of β in each section ℓ is set to $\sqrt{nP_\ell}$, where $P_1, \ldots, P_L$ are pre-specified positive constants that satisfy $\sum_{\ell=1}^{L} P_\ell = P$, the average symbol power allowed.

Both A and the *power allocation* $\{P_1, \ldots, P_L\}$ are known to both the encoder and decoder in advance. The choice of power allocation plays a crucial role in determining the error performance of the decoder. Without loss of generality, we will assume that the power allocation is non-increasing across sections. Two examples of power allocation are:

- Flat power allocation, where $P_\ell = \frac{P}{L}$ for all ℓ. This choice was used in [17] to analyse the error performance with optimal (least-squares) decoding.

- Exponentially decaying power allocation, where $P_\ell \propto 2^{-2C\ell/L}$. This choice was used for the asymptotically capacity-achieving decoders proposed in [1, 18, 23].

Rate: As each of the L sections contains M columns, the total number of codewords is M^L. With the block length being n, the rate of the code is given by

$$R = \frac{\log(M^L)}{n} = \frac{L \log M}{n}. \tag{1.8}$$

In other words, a SPARC codeword corresponding to $L \log M$ input bits is transmitted in n channel uses.

Encoding

The input bitstream is split into chunks of $\log M$ bits. A chunk of $\log M$ input bits can be used to index the location of the non-zero entry in one section of β. Hence L successive chunks determine the message vector β, with the ℓth chunk of $\log M$ input bits determining the non-zero location in section ℓ, for $1 \leq \ell \leq L$. The codeword to be transmitted is then $A\beta$.

1.4.2 Decoders

Having received the channel output sequence $y = A\beta + w$, the decoder must form an estimate of β, denoted by $\hat{\beta}$. We first consider the decoder figures of merit.

Error Rate

We measure the *section error rate* $\mathcal{E}_{\text{sec}}$ as

$$\mathcal{E}_{\text{sec}} = \frac{1}{L} \sum_{\ell=1}^{L} \mathbf{1}\Big\{\widehat{\beta}_\ell \neq \beta_\ell\Big\}, \tag{1.9}$$

where β_ℓ represents the entries in β associated with section ℓ of the SPARC. Assuming a uniform mapping between the input bitstream and the non-zero locations in each section, each section error will cause approximately half of the bits it represents to be incorrect, leading to a *bit error rate* $\mathcal{E}_{\text{ber}} \approx \frac{1}{2}\mathcal{E}_{\text{sec}}$. This bit error rate can also be measured empirically by counting bits that are different between the input bitstream and the decoded output bitstream.

Another figure of merit is the *codeword error rate* $\mathcal{E}_{\text{cw}}$, which estimates the probability $\mathbb{P}(\widehat{\beta} \neq \beta)$. If the SPARC is used to transmit a large number of messages (each via a length n codeword), $\mathcal{E}_{\text{cw}}$ measures the fraction of codewords that are decoded with one or more section errors. The codeword error rate is insensitive to where and how many section errors occur within a codeword when it is decoded incorrectly.

As suggested in [18], having selected a SPARC code and decoder, we can construct a concatenated code using an outer block code such as a Reed-Solomon (RS) code [25]. By choosing M to be a prime power, an RS code defined over a finite field of order M gives a one-to-one mapping between each symbol of the RS code and each section of the SPARC, simplifying concatenation. If the inner SPARC code has rate R and the outer code has a rate $(1 - 2\epsilon)$ then the concatenated code will have rate $R(1 - 2\epsilon)$, with decoding complexity that remains polynomial in n, and obtains exactly the transmitted codeword whenever the SPARC section error rate is less than ϵ. This allows trading some rate loss for the ability to clean up any small residual section error rate from the SPARC code.

Optimal Decoding

Assuming all codewords are equally likely, the optimal decoder is

$$\hat{\beta}_{opt} = \arg\min_{\hat{\beta}} \|y - A\hat{\beta}\|^2, \tag{1.10}$$

where the minimum is over all possible message vectors $\hat{\beta}$.

For rates $R < \mathcal{C}$, this optimal decoder achieves exponential decay of error probability with growing block length n [17]. This important result shows that under optimal decoding, SPARCs are essentially as good as Shannon random codebooks for the AWGN channel. However, as with the maximum likelihood decoder for the Shannon random codebook, the computational complexity is exponential in n, and so this decoder is not feasible in practice.

Feasible Decoders

To give some insight into the operation of feasible decoders, first consider the exponential power allocation described above. With this power allocation, the first section is allocated the highest power, and we might hope to decode it first, by simply finding the column of A in that first section which is best aligned with y. Having found the column for the first section, we

subtract it from y to form a *residual* and repeat the process with the second section, etc. This iterative scheme is similar to successive cancellation decoding, which is a common technique for decoding on the Multiple Access Channel [26].

Unfortunately, this strategy performs poorly with SPARCs. As the number of columns per section grows, we become increasingly likely to make a mistake, and when we do, all subsequent sections are likely to decode in error. Nevertheless, the intuition of successively decoding sections and removing their contribution to the received signal is relevant to the operation of the feasible decoders discussed below.

The first feasible decoder for SPARCs was proposed by Barron and Joseph in [18], called "adaptive successive decoding". Rather than starting with the first section and decoding in strict order, they first compute an inner product of every column in A with the normalised channel output $y/\|y\|$. Columns where this inner product exceeds a pre-determined threshold are considered to have been transmitted, and used to form a first estimate $\hat{\beta}^1$. Subsequently, a residual is generated as $r^1 = y - A\hat{\beta}^1$, and the operation repeated using $r^1/\|r^1\|$ rather than $y/\|y\|$. This continues until either a pre-determined number of iterations is reached, or all sections are decoded, or no further inner products exceed the required threshold.

Barron and Joseph showed [18] that this decoder, when using the exponential power allocation, achieves near-exponential decay of error probability with growing block length, for any fixed rate $R < \mathcal{C}$. However, in practice for feasible block lengths the section error rates are found to be high for rates near capacity.

Subsequently, an adaptive soft-decision successive decoder was proposed by Cho and Barron in [22, 23], which iteratively updates a posterior probability of each entry in β being the true non-zero entry for its section. Since this decoder no longer makes hard decisions at each iteration, it is able to refine its estimates of the posterior probabilities until they converge, at which point final hard decisions are made by selecting the most likely entry in each section. This decoder has also been shown to achieve near-exponential decay of error probability for $R < \mathcal{C}$, and additionally demonstrates improved section error rates compared to the adaptive successive decoder.

Approximate Message Passing Decoding

Approximate Message Passing (AMP) algorithms are a class of algorithms [27–30] which approximate loopy belief propagation on dense factor graphs, where belief propagation would usually be infeasible. An AMP decoder for SPARCs was proposed in [1] and is the focus of

Chapter 2. It is similar in operation to the adaptive soft-decision successive decoder of [22,23], but has simpler update rules to generate its posterior probabilities, leading to faster and easier implementations. As with the preceding decoders, it has been shown to achieve the channel capacity, and obtains excellent section error rates.

Another AMP decoder for SPARCs has also been proposed in [31], which takes advantage of spatial coupling to achieve good section error rates, but will not be discussed in this thesis.

1.4.3 SPARCs for Source Coding

The same underlying SPARC structure and encoder may also be used for source coding [32–34]. An encoder determines which choice of β will give a SPARC codeword $A\beta$ which represents the source X with the lowest possible distortion, the encoded form of the source is the choice of non-zero positions in each section of β, and the decoder simply computes $\hat{X} = A\beta$.

In principle this is a similar operation to channel coding, where we search for the SPARC codeword which was most likely to have been transmitted given the noisy codeword we received. We can view the error between the true source X and our selected codeword $\hat{X}$ as similar to the channel noise which corrupts the channel codeword.

A simple and efficient SPARC source encoder was presented in [32]. It creates an initial residue equal to the source, and proceeds section by section, greedily selecting whichever column in that section is most correlated with the current residue. With a suitable choice of power allocation, it is shown that this encoder is able to asymptotically achieve the rate-distortion region for Gaussian sources.

1.5 Notation

Before proceeding, we establish some notation which will be used throughout this thesis. We use log to denote logarithms with base 2, and ln to denote natural logarithms. For a positive integer N, we use $[N]$ to denote the set $\{1, \ldots, N\}$. The transpose of a matrix A is denoted by A^*. The indicator of an event $\mathcal{E}$ is denoted by $\mathbf{1}\{\mathcal{E}\}$. The ℓ_2-norm of a vector x is denoted by $\|x\|$. Rate is measured in bits. We use sec_ℓ to denote the set of indices in section ℓ, e.g., $j \in \text{sec}_\ell$ corresponds to $j \in \{(\ell-1)M + 1, \ldots, \ell M\}$ where $\ell \in [L]$. We use $\text{sec}(i)$ to denote the set of indices in the section containing i, e.g., $j \in \text{sec}(i)$ corresponds to $j \in \text{sec}_\ell$ where ℓ is such that $i \in \text{sec}_\ell$.

1.6 Contributions of this Thesis

This thesis focuses on design techniques and efficient decoding algorithms for practical sparse regression codes. We seek to obtain performance close to the Shannon limits while using realistic block lengths and with low computational complexity. In particular, we propose techniques to reduce the decoding complexity, outline new power allocations which give significantly improved performance, and extend the basic SPARC structure with new construction techniques and applications to various multiuser models.

The remaining chapters are organised as follows.

- Chapter 2 describes in detail the approximate message passing decoder for SPARCs, discussing its design, giving some intuition into its operation, and summarises the main results.

- Chapter 3 discusses a variety of improvements to the basic AMP design which yield decreased computational complexity and improved error rates at practical block lengths. These include: using Hadamard-based design matrices, new power allocation routines for better finite length performance, and an investigation into how the choice of L and M affects the error performance.

- Chapter 4 investigates the use of outer codes for SPARCs. We demonstrate that applying LDPC outer codes to SPARCs with a new decoding technique can obtain excellent codeword error rates and significantly increased falloff in bit error rate as snr increases.

- Chapter 5 considers a modification to the basic SPARC structure, where multiple possible values are permitted as the non-zero entry in β, which we call *Modulated SPARCs*. We derive a new AMP where the non-zero values are chosen from a symmetric K-ary constellation, and prove that the modulated SPARC with $K = 2$ achieves the AWGN capacity. We also evaluate the empirical performance of these modified SPARCs.

- Chapter 6 implements SPARCs with the AMP decoder for multi-user channels such as the broadcast and multiple access channels, and for multi-user source coding models such as multiple description and Wyner-Ziv. The empirical performance is evaluated.

- Finally, Chapter 7 presents conclusions and directions for future work.

Chapter 2

Approximate Message Passing for SPARCs

When a channel code with equally likely codewords is used over a noisy channel, the optimum decoder is the maximum likelihood decoder, which selects whichever codeword was most likely to have been transmitted given the channel outputs. For SPARCs over the AWGN channel described in Chapter 1, finding the likelihood of any message vector $\hat{\beta}$ given the channel output sequence y is straightforward. From the SPARC encoding and the definition of the AWGN, we have

$$y = A\beta + w, \tag{2.1}$$

where $w = \{w_i\}_{i\in[n]}$ and $w_i \sim \mathcal{N}(0, \sigma^2)$ is additive white Gaussian noise. The likelihood of a particular $\hat{\beta}$ being the transmitted β is

$$\begin{aligned} \mathcal{L}(\beta = \hat{\beta} \mid y) &\propto \mathbb{P}(y \mid \beta = \hat{\beta}) && (2.2)\\ &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - (A\hat{\beta})_i)^2}{2\sigma^2}\right). && (2.3) \end{aligned}$$

This likelihood is maximised by minimising $\|y - A\hat{\beta}\|^2$, and so the maximum likelihood codeword is the solution to

$$\arg\min_{\hat{\beta}} \|y - A\hat{\beta}\|^2. \tag{2.4}$$

However, we cannot hope to compute this over all possible $\hat{\beta}$, and so we must find a computationally feasible decoding algorithm instead. Computationally feasible decoders were first proposed in [17, 18]. In this chapter, we describe the AMP algorithm proposed in [1].

We will first describe the AMP decoder in Section 2.1, then describe its background and outline the derivation in Section 2.2. State evolution is an important concept for predicting the behaviour of the AMP decoder and is described in Section 2.3. Finally, we outline the proof that this decoder can achieve the channel capacity in the large system limit in Section 2.4.

2.1 The AMP Channel Decoder

Given the channel output $y = A\beta + w$, the AMP decoder generates successive estimates of the message vector, denoted $\{\beta^t\}$, $\beta^t \in \mathbb{R}^{LM}$, for $t = 1, 2, \ldots$.

Initialise $\beta^0 = 0$, then compute

$$z^t = y - A\beta^t + \frac{z^{t-1}}{\tau_{t-1}^2}\left(P - \frac{\|\beta^t\|^2}{n}\right), \tag{2.5}$$

$$\beta_i^{t+1} = \eta_i^t(\beta^t + A^* z^t), \quad \text{for } i = 1, \ldots, ML, \tag{2.6}$$

where quantities with negative indices are set equal to zero. The constants $\{\tau_t\}$ and the estimation functions $\eta_i^t(.)$ are defined as follows. First, define

$$\tau_0^2 = \sigma^2 + P, \quad \tau_{t+1}^2 = \sigma^2 + P(1 - x_{t-1}), \quad t \geq 0, \tag{2.7}$$

where

$$x_{t-1} := x(\tau_{t-1}), \tag{2.8}$$

and

$$x(\tau) := \sum_{\ell=1}^{L} \frac{P_\ell}{P}\, \mathbb{E}\left[\frac{\exp\left(\frac{\sqrt{nP_\ell}}{\tau}\left(U_1^\ell + \frac{\sqrt{nP_\ell}}{\tau}\right)\right)}{\exp\left(\frac{\sqrt{nP_\ell}}{\tau}\left(U_1^\ell + \frac{\sqrt{nP_\ell}}{\tau}\right)\right) + \sum_{j=2}^{M} \exp\left(\frac{\sqrt{nP_\ell}}{\tau} U_j^\ell\right)}\right]. \tag{2.9}$$

In (2.9), $\{U_j^\ell\}$ are i.i.d. $\mathcal{N}(0,1)$ random variables for $j \in [M]$, $\ell \in [L]$. The equations (2.7) and (2.9) are together called the *state evolution*, and are discussed in more detail in Section 2.3.

For $i \in \text{sec}_\ell$, define

$$\eta_i^t(s) = \sqrt{nP_\ell}\frac{\exp\left(s_i \frac{\sqrt{nP_\ell}}{\tau_t^2}\right)}{\sum_{j \in \text{sec}(i)} \exp\left(s_j \frac{\sqrt{nP_\ell}}{\tau_t^2}\right)}, \quad 1 \leq i \leq ML. \tag{2.10}$$

Note that $\eta_i^t(s)$ depends on all components of s in $\text{sec}(i)$. For brevity, we will write $s = A^* z^t + \beta^t$, with the understanding that only the components in the section containing i are used to compute β_i^t.

Finally, after T iterations, the decoded message vector $\hat{\beta}$ is produced by setting the maximum value in section ℓ of β^T to $\sqrt{nP_\ell}$ and the remaining entries to zero, for $1 \leq \ell \leq L$.

We view the quantity z_t in (2.5) as a modified residual, tracking what remains of the original channel output y after the current estimate of the codeword $A\beta^t$ has been removed. Our residual also contains an extra term based on the previous value of the residual, which is discussed in Section 2.2. This extra term is a key part of the approximate message passing algorithm.

Once the residual has been found, it is used to form a new estimate β^t in (2.6), using the equation for η_i^t in (2.10). This function η_i^t is in fact the Bayes-optimal estimator of β_i given $A^* z^t + \beta^t = s$, under the assumption that the test statistic s can be expressed as the sum of β and an independent Gaussian noise vector of variance τ_t^2. This estimator η_i^t finds the posterior probabilities of each column $i \in \text{sec}_\ell$ being the one transmitted in section $\ell \in [L]$, and so β^t may also be viewed as containing, for each section, the probability distribution over its columns, weighted by $\sqrt{nP_\ell}$.

By iterating these two update rules, the AMP decoder provides excellent empirical performance and provably achieves the channel capacity in the large system limit. This large system limit is defined as the dictionary size going to ∞, i.e., $L, M, n \to \infty$, while maintaining the relationship of (1.8) ($nR = L \log M$), and defining $M = L^a$ for some constant $a > 0$.

2.2 Background and Derivation

Message passing, also known as belief propagation, has been used to decode channel codes such as LDPC codes [9], as well as to solve various problems in machine learning and statistical physics [35]. This class of algorithms perform inference on a factor graph, which is a bipartite graph which represents a factorisation of a function, typically a posterior probability distribution. In belief propagation, each node computes a *message* to be sent to nodes it is connected to, based on the messages which that node receives. Variable nodes send messages to factors containing a probability distribution over that variable based on messages received from all other connected factors. Factors send messages to variables containing a probability distribution over that variable based on the factor itself and the messages received from the

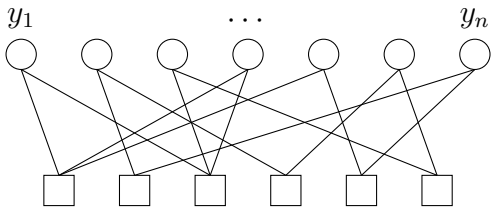


Figure 2.1: An example of a small LDPC factor graph. Factors represent one row of the parity check matrix, while variables represent elements of a codeword. Each variable connected to a parity check is involved in the computation of that parity check. Due to the low density nature of the codes, there are few edges, and so message passing decoding is feasible.

other variables involved in that factor. Note that messages to a given node are always based on all the messages received previously *excluding* any from the node in question. On factor graphs which are trees (i.e., do not contain any loops), this algorithm yields exact marginal distributions for each variable, starting with the leaf nodes and progressing upwards to the root of the tree. In this situation the exclusion mentioned above is immaterial, as nodes will never receive a message from a node they need to send a message to. Refer to [9] for a more thorough treatment of standard belief propagation.

On more general bipartite graphs with loops the same algorithm may be used, but it is no longer guaranteed to terminate, or to provide the exact solution. However, in many circumstances it is found to perform well and eventually converge to the correct solution. This is the case with typical LDPC decoding, where each factor represents a parity check: all variables involved in that factor must sum, $\mod 2$, to 0. Each variable represents one symbol of the codeword. Variables start out with a probability of being 0 or 1 based on the channel outputs, and update this probability based on messages received from the parity checks they are involved in. For example, if one variable had a 0.6 probability of being 1, but was involved in a parity check where all other variables had high probabilities of being 0, the message from the parity check would suggest that this variable is also likely to be a 0, and so the variable lowers its probability of being 1 accordingly in messages sent to other factors. Figure 2.1 shows an example factor graph for a very small LDPC code; since there are few edges the messages along each edge are readily computable.

Unfortunately, this is not the case for SPARCs. We have LM variables, representing each entry in β, and n constraints, representing each row of the SPARC design matrix A. Consider

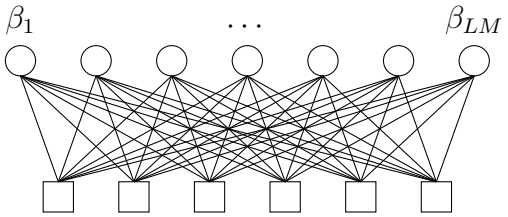


Figure 2.2: The factor graph for a small SPARC code. The factors represent each of the n rows of A, while the variables are the LM elements of β. Each column in A may, depending on the location of the non-zero elements of β, contribute to each element of the codeword, and so the graph is fully connected. Running message passing on a full size SPARC graph would be infeasible as the number of edges is LMn.

the first symbol in a SPARC codeword $A\beta$:

$$(A\beta)_1 = A_{11}\beta_1 + A_{12}\beta_2 + \cdots + A_{1,LM}\beta_{LM}. \tag{2.11}$$

Clearly every variable node is involved in every constraint. The factor graph is dense, as illustrated in Figure 2.2, and running the normal message passing algorithm described above is not feasible. However, by writing down this message passing algorithm as a starting point, it is possible to perform approximations which lead to a computationally efficient algorithm.

Similar relaxations of belief propagation for dense factor graphs were first considered for CDMA multiuser detection [36–38]. Subsequently, approximate message passing algorithms of the form used in this thesis were introduced in the context of compressed sensing by Donoho *et al* [27–29]. Variants of AMP for problems related to compressed sending were proposed in [30, 31, 39]. In this thesis, we will discuss AMP only in the context of SPARCs.

To obtain the AMP described in Equations (2.5) and (2.6), we start by considering a message passing algorithm for decoding SPARCs, which iteratively estimates β from the channel outputs y. Use the indices a, b to denote factor nodes, and i, j to denote variable nodes. For $i \in [N]$ and $a \in [n]$, initialise $\beta^0_{j\to a} = 0$, and iteratively compute the following messages for $t \geq 0$:

$$z^t_{a\to i} = y_a - \sum_{j\in[N]\setminus i} A_{aj}\beta^t_{j\to a}, \tag{2.12}$$

$$\beta^{t+1}_{i\to a} = \eta^t_i(s_{i\to a}), \tag{2.13}$$

where $\eta_i^t(.)$ is given in (2.10), and, for $i \in \text{sec}_\ell$, the entries of the test statistic $s_{i\to a} \in \mathbb{R}^M$ are

$$(s_{i\to a})_i = \sum_{b\in[n]\setminus a} A_{bi} z^t_{b\to i} \tag{2.14}$$

$$(s_{i\to a})_j = \sum_{b\in[n]} A_{bj} z^t_{b\to j}, \quad j \in \text{sec}_\ell \setminus i. \tag{2.15}$$

This message passing algorithm would produce good estimates of β, but because every message is unique and there are $2LMn$ messages, it is not feasible to run. If we could approximate the message $z_{a\to i}$ to not involve i, giving a single z_a, and similarly approximate $\beta_{i\to a}$ to not involve a, then we would only have to compute $LM + n$ messages per iteration, which would be feasible.

The messages' dependence on i and a respectively comes from a single excluded term; in the case of $z_{a\to i}$ we exclude $A_{ai}\beta_{i\to a}$ from the sum, while with $\beta_{i\to a}$ we exclude $A_{ai}z_{a\to i}$ from the argument. If we simply neglected to exclude these terms, we would recover the following simple set of update rules:

$$z^t = y - A\beta^t, \tag{2.16}$$

$$\beta_i^{t+1} = \eta_i^t(\beta^t + A^* z^t), \quad \text{for } i = 1, \dots, ML. \tag{2.17}$$

The update rule for z^t can be seen as forming a residual from the current estimate β^t, then using it to form a new estimate, and iterating. While simple to implement, these rules do not perform well, and have an error which does not vanish in the large system limit. The nature of this error is discussed further at the end of this section.

Instead, [1, Appendix A] finds a less coarse approximation. First we make the dependency on i and a more explicit by writing

$$z^t_{a\to i} = z^t_a + \delta z^t_{a\to i} \quad \text{and} \quad \beta^{t+1}_{i\to a} = \beta^{t+1}_i + \delta\beta^{t+1}_{i\to a}. \tag{2.18}$$

We can now use (2.12) to write

$$z^t_a = y_a - \sum_{j\in[N]} A_{aj}(\beta^t_j + \delta\beta^t_{j\to a}), \quad \delta z^t_{a\to i} = A_{ai}(\beta^t_j + \delta\beta^t_{i\to a}). \tag{2.19}$$

To obtain $\delta\beta_{i\to a}^t$, we perform a Taylor expansion of η_i^t around the argument $\left\{\sum_{b\in[n]} A_{bj} z_{b\to j}^t\right\}_{j\in \text{sec}(i)}$, which does not depend on a. This expansion yields

$$\beta_{i\to a}^{t+1} \approx \eta_i^t\left(\left\{\sum_{b\in[n]} A_{bj} z_{b\to j}^t\right\}_{j\in \text{sec}(i)}\right) - A_{ai} z_{a\to i}^t\, \partial_i \eta_i^t\left(\left\{\sum_{b\in[n]} A_{bj} z_{b\to j}^t\right\}_{j\in \text{sec}(i)}\right), \quad (2.20)$$

where $\partial_i \eta_i^t(.)$ is the partial derivative of η_i^t with respect to the component of the argument corresponding to index i. This partial derivative is

$$\begin{aligned}
\partial_i \eta_i^t(s) &= \eta_i^t(s) \partial_i \ln \eta_i^t(s) && (2.21)\\
&= \eta_i^t(s)\left(\frac{\sqrt{nP_\ell}}{\tau_t^2} - \frac{\sqrt{nP_\ell}}{\tau_t^2} \frac{e^{\frac{s_i\sqrt{nP_\ell}}{\tau_t^2}}}{\sum_{j\in \text{sec}(i)} e^{\frac{s_j\sqrt{nP_\ell}}{\tau_t^2}}}\right) && (2.22)\\
&= \frac{\eta_i^t(s)}{\tau_t^2}\left(\sqrt{nP_\ell} - \eta_i^t(s)\right), && (2.23)
\end{aligned}$$

which gives

$$\begin{aligned}
\beta_{i\to a}^{t+1} \approx\ & \eta_i^t\left(\left\{\sum_{b\in[n]} A_{bj} z_{b\to j}^t\right\}_{j\in \text{sec}(i)}\right)\\
& - \frac{A_{ai} z_a^t}{\tau_t^2} \eta_i^t\left(\left\{\sum_{b\in[n]} A_{bj} z_{b\to j}^t\right\}_{j\in \text{sec}(i)}\right)\left[\sqrt{nP_\ell} - \eta_i^t\left(\left\{\sum_{b\in[n]} A_{bj} z_{b\to j}^t\right\}_{j\in \text{sec}(i)}\right)\right]. \quad (2.24)
\end{aligned}$$

The term $A_{ai} z_{a\to i}^t$ in (2.20) has been replaced by $A_{ai} z_a^t$ as the difference $A_{ai}\delta z_{a\to i}^t$ is $O(\sqrt{\log n}/n)$, and we are only keeping terms greater than $O(\frac{1}{\sqrt{n}})$.

Since only the second term of (2.24) depends on a, we can write

$$\beta_i^{t+1} = \eta_i^t\left(\left\{\sum_{b\in[n]} A_{bj}(z_b^t + \delta z_{b\to j}^t)\right\}_{j\in \text{sec}(i)}\right), \quad (2.25)$$

$$\delta\beta_{i\to a}^{t+1} = -\frac{A_{ai}z_a^t}{\tau_t^2}\,\eta_i^t\left(\left\{\sum_{b\in[n]} A_{bj}(z_b^t + \delta z_{b\to j}^t)\right\}_{j\in \text{sec}(i)}\right) \cdot\left[\sqrt{nP_\ell} - \eta_i^t\left(\left\{\sum_{b\in[n]} A_{bj}(z_b^t + \delta z_{b\to j}^t)\right\}_{j\in \text{sec}(i)}\right)\right]. \tag{2.26}$$

Note that $\delta\beta_{i\to a}^{t+1} = O(\log n/\sqrt{n})$, and therefore $A_{ai}\delta\beta_{i\to a}^t = O(\log n/n)$, so we can write

$$\delta z_{a\to i}^t = A_{ai}\beta_i^t. \tag{2.27}$$

This gives

$$\beta_i^{t+1} = \eta_i^t\left(\left\{\sum_{b\in[n]} A_{bj}z_b^t + A_{bj}^2\beta_j^t\right\}_{j\in \text{sec}(i)}\right) \tag{2.28}$$

$$\overset{(a)}{=} \eta_i^t\left(\left\{(A^*z^t + \beta^t)_j\right\}_{j\in \text{sec}(i)}\right), \tag{2.29}$$

where (a) is because $\sum_b A_{bj}^2 \to 1$ as $n\to\infty$, giving the final update rule for β_i^{t+1}, as in (2.6). For z^t, we use (2.27) in (2.26) to find

$$\delta\beta_{i\to a}^{t+1} = \frac{-A_{ai}z_a^t}{\tau_t^2}\eta_i^t\left(\left\{\sum_{b\in[n]}(A^*z^t + \beta^t)_j\right\}_{j\in \text{sec}(i)}\right) \cdot\left[\sqrt{nP_\ell} - \eta_i^t\left(\left\{\sum_{b\in[n]}(A^*z^t + \beta^t)_j\right\}_{j\in \text{sec}(i)}\right)\right], \tag{2.30}$$

and by substituting into (2.19), we can write

$$
\begin{aligned}
z_a^t &= y_a - \sum_{k\in[N]} A_{ak}(\beta_k^t + \delta\beta_{k\to a}^t) \\
&= y_a - \sum_{k\in[N]} A_{ak}\,\eta_k^{t-1}\left(A^* z^{t-1} + \beta^{t-1}\right) \\
&\quad + \frac{A_{ak}^2 z_a^{t-1}}{\tau_{t-1}^2}\,\eta_k^{t-1}\left(A^* z^{t-1} + \beta^{t-1}\right)\cdot\left[\sqrt{nP_{\mathsf{sec}(k)}} - \eta_k^{t-1}\left(A^* z^{t-1} + \beta^{t-1}\right)\right] \\
&\overset{(b)}{=} y_a - (A\beta^t)_a + \frac{z_a^{t-1}}{n\tau_{t-1}^2}\,(nP - \|\beta^t\|^2),
\end{aligned}
\tag{2.31}
$$

where (b) is obtained from noting that $A_{ak}^2 \approx \frac{1}{n}$, and from (2.10) that

$$
\sum_{k\in[N]} \sqrt{nP_{\mathsf{sec}(k)}}\,\eta_k^t(s) = \sum_{\ell=1}^{L} nP_\ell = nP,
$$

and finally from (2.29) that

$$
\sum_k (\eta_k^{t-1}\left(A^* z^{t-1} + \beta^{t-1}\right))^2 = \sum_k (\beta_k^t)^2 = \|\beta^t\|^2.
$$

This yields the z_t update rule (2.5).

The extra term compared to (2.16) is called the *Onsager reaction term*. The role of this term is to ensure that asymptotically, the test statistic $s^t = \beta^t + A^* z^t$ is distributed as $\beta + \tau_t Z$, where Z is an i.i.d. $\mathcal{N}(0,1)$ vector independent of β — so that s^t is an observation of the true β in white Gaussian noise. This property is key to the performance of the AMP decoder, where we can show that τ_t^2 will tend to σ^2 such that s^t tends to an observation of β in noise just of power σ^2, and is discussed in more detail in Section 2.3.

To illustrate the necessity of the Onsager reaction term, consider the iteration without it,

described in (2.16), where $z^t = y - A\beta^t$. Expanding s^t, we obtain:

$$\begin{aligned} s^t &= A^* z^t + \beta^t && (2.32)\\ &= A^*(y - A\beta^t) + \beta^t && (2.33)\\ &= A^*(A\beta + w - A\beta^t) + \beta^t && (2.34)\\ &= A^*A\beta + A^*w - A^*A\beta^t + \beta^t && (2.35)\\ &= \beta + A^*w + (I - A^*A)(\beta^t - \beta) && (2.36) \end{aligned}$$

We would like s^t to be asymptotically distributed as $\beta + \tau_t Z$. Recall that

$$\tau_{t+1}^2 = \sigma^2 + P(1 - x_t),$$

and use

$$1 - x_{t+1} = \frac{1}{nP}\mathbb{E}\left[\|\beta - \beta^{t+1}\|^2\right]$$

from (2.43) to obtain

$$\tau_{t+1}^2 = \sigma^2 + \frac{1}{n}\|\beta^t - \beta\|^2. \tag{2.37}$$

In the expansion of s^t above, the term A^*w is a vector of i.i.d. normal variates with variance $\frac{1}{n}\|w\|^2$, which is asymptotically σ^2, and can be shown to be independent of $(I - A^*A)(\beta^t - \beta)$.

The matrix $(I - A^*A)$ has normal entries with variance $\frac{1}{n}$; therefore if β^t is independent of $(I - A^*A)$ the term $(I - A^*A)(\beta^t - \beta)$ will be a vector of normal variates with variance $\frac{1}{n}\|\beta^t - \beta\|^2$, and consequently s^t itself is distributed as $\beta + \tau_t Z$, as desired.

However, the iteration introduces correlations between A and β^t, and consequently the above derivation does not hold and s^t will not have the desired asymptotic distribution. The addition of the Onsager term serves to asymptotically cancel out these correlations, leading to the desired distribution of s^t. For more intuition about the role of the Onsager reaction term in AMP, refer to [40].

The asymptotic distribution of s^t is also responsible for the form of $\eta_i^t(s^t)$ in (2.10). Specif-

ically, η_i^t is derived in [1] as the Bayes-optimal estimate of β given the observation s^t:

$$\begin{aligned}\beta_i^{t+1} &= \eta_i^t(s^t = \beta^t + A^* z^t) && (2.38)\\ &= \mathbb{E}\left[\beta \mid \beta + \tau_t z\right] && (2.39)\\ &= \sqrt{nP_\ell}\frac{\exp\left(s_i \frac{\sqrt{nP_\ell}}{\tau_t^2}\right)}{\sum_{j\in \text{sec}(i)} \exp\left(s_j \frac{\sqrt{nP_\ell}}{\tau_t^2}\right)}, \quad 1 \le i \le ML. && (2.40)\end{aligned}$$

2.3 State Evolution

The AMP decoder defined by (2.5) and (2.6) require a set of $\{\tau_t\}$ coefficients to operate. These coefficients can be found offline by a process known as state evolution. The state evolution equations (2.7) and (2.9) are repeated here for convenience:

$$\tau_0^2 = \sigma^2 + P, \quad \tau_{t+1}^2 = \sigma^2 + P(1 - x_{t-1}), \quad t \ge 0, \tag{2.41}$$

with

$$x(\tau) := \sum_{\ell=1}^{L} \frac{P_\ell}{P}\, \mathbb{E}\left[\frac{\exp\left(\frac{\sqrt{nP_\ell}}{\tau}\left(U_1^\ell + \frac{\sqrt{nP_\ell}}{\tau}\right)\right)}{\exp\left(\frac{\sqrt{nP_\ell}}{\tau}\left(U_1^\ell + \frac{\sqrt{nP_\ell}}{\tau}\right)\right) + \sum_{j=2}^{M} \exp\left(\frac{\sqrt{nP_\ell}}{\tau} U_j^\ell\right)}\right], \tag{2.42}$$

where $\{U_j^\ell\}$ are i.i.d. $\mathcal{N}(0,1)$ random variables for $j \in [M],\ \ell \in [L]$.

These constants have a physical interpretation in the context of the AMP decoder. We first address τ_t^2. As mentioned at the end of Section 2.2, the test statistic $s^t = \beta^t + A^* z^t$ is asymptotically distributed as $\beta + \tau_t Z$, where $Z \sim \mathcal{N}(0,1)$. See Section 2.4 and [1] for further details. This means that (in the large system limit) τ_t^2 is the variance of the Gaussian noise in which, asymptotically, our test statistic observes β. We see from its definition above that this noise starts out with power $\sigma^2 + P$, which reflects the initial condition of $s^0 = A^* z^0 = A^* y$ having variance equal to $\sigma^2 + P$. That is, the initial test statistic is corrupted by not only the full channel noise σ^2, but also by the entire message power P. Since the codeword $A\beta$ is made up of one column of A from each section $\ell \in [L]$ weighted by a power P_ℓ, we can view each of these unknown columns as contributing P_ℓ power to the noise, totalling P. From the successive cancellation decoding viewpoint described in Chapter 1, we have not yet decoded any sections, and so the interfering power is at the maximum.

As decoding progresses, some of these columns are decodable with high probability, and we can imagine removing their contribution to the noise power. This is done in the update rule for z^t. As a result, τ_t^2 will decrease. We can see this in (2.41) for τ_t^2, which decreases as x_t increases from its initial condition of $x_0 = 0$. If decoding is completely successful, x_t reaches a value of 1, or, equivalently, τ_t^2 reaches a value of σ^2. Figure 2.3 shows this trajectory for τ_t^2.

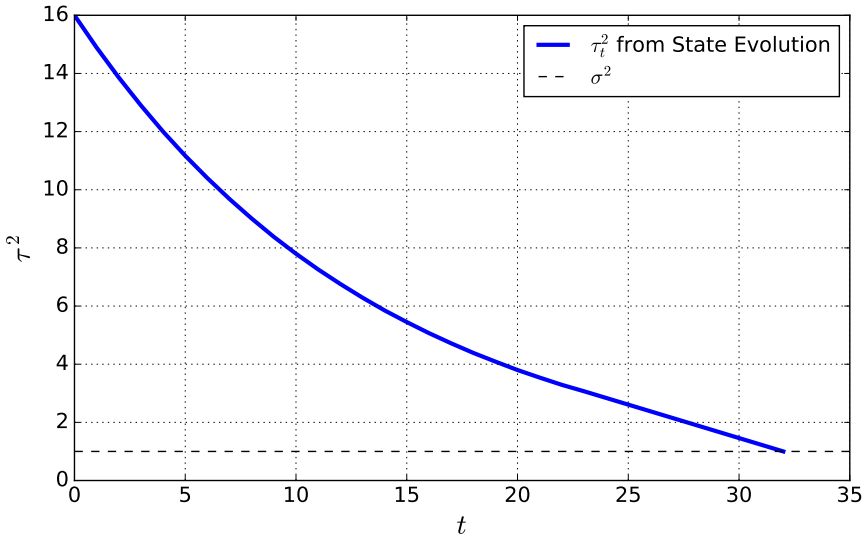


Figure 2.3: The state evolution predicted trajectory for τ_t^2, with $\sigma^2 = 1$ and $P = 15$.

The parameter x_t represents (again, in the large system limit) the power-weighted expected fraction of sections decoded correctly at time t, or in other words, the expected fraction of the message power which has now been successfully decoded. This interpretation is made clear from the following relationship, established in [1], which leads to the definition of x_t in (2.9):

$$1 - x_{t+1} = \frac{1}{nP}\mathbb{E}\left[\|\beta - \beta^{t+1}\|^2\right] \tag{2.43}$$

This quantity, and its non-power-weighted equivalent, are useful predictive tools to estimate the performance of an AMP decoder in advance. Since the state evolution equations do not depend on the actual message or channel realisation, they can be computed offline for any given set-up, and the resulting τ_t^2 and x^t used to predict the performance of the AMP decoder. We find that the state evolution very accurately predicts the empirical behaviour of the AMP. For example, Figure 2.4 shows in black the value $1 - x_t$, and in green, the actual value of $1 - x_t$

for 200 empirical trials. This empirical value is found by computing the quantity $1 - \frac{\beta^* \beta^t}{nP}$ at each iteration t, where β is the true message vector used to create the message.

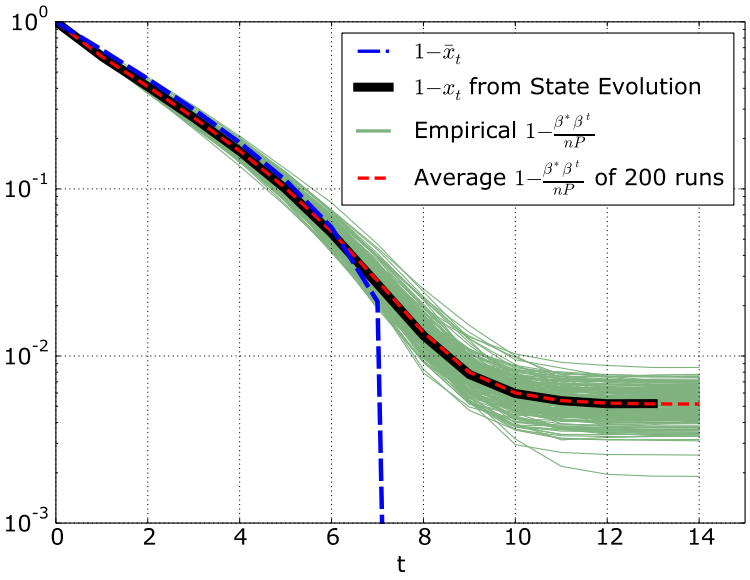


Figure 2.4: Comparison of state evolution predictions to empirical AMP performance. The SPARC parameters are $M = 512$, $L = 1024$, snr $= 15$, $R = 0.7\mathcal{C}$, and $P_\ell \propto 2^{-2\mathcal{C}\ell/L}$. The average of the 200 trials (green curves) is the dashed red curve, which is almost indistinguishable from the state evolution prediction (black curve).

It is clear from the figure that the actual AMP decoder closely tracks the state evolution prediction. In particular, x_t closely tracks the performance metric $\frac{\beta^* \beta^t}{nP}$. This allows us to use the state evolution to make predictions about the behaviour of the AMP, such as estimating the error rate or deciding how many iterations to run the decoder for. More importantly, by showing that the AMP decoder follows the state evolution asymptotically, we will be able to prove that the AMP decoder asymptotically achieves the channel capacity. To investigate this effect, we first consider the asymptotic state evolution.

2.4 Asymptotic State Evolution

We are interested in the behaviour of τ_t^2 and x_t asymptotically. First, we write

$$\bar{x}(\tau) := \lim x(\tau), \tag{2.44}$$

where lim is understood to mean the large system limit, where $L, M, n \to \infty$ while maintaining (1.8) ($nR = L \log M$), and with $M = L^a$ for some constant $a > 0$. We then consider [1, Lemma 1], which is reproduced here as Lemma 2.1.

Lemma 2.1. *[1, Lemma 1] For any power allocation $\{P_\ell\}_{\ell=1,\dots,L}$ that is non-increasing with ℓ, we have*

$$\bar{x}(\tau) = \lim \sum_{\ell=1}^{\lfloor \xi^*(\tau) L \rfloor} \frac{P_\ell}{P}, \tag{2.45}$$

where $\xi^*(\tau)$ is the supremum of all $\xi \in (0, 1]$ that satisfy

$$\lim L P_{\lfloor \xi L \rfloor} > 2(\ln 2) R \tau^2.$$

If $\lim L P_{\lfloor \xi L \rfloor} \leq 2(\ln 2) R \tau^2$ for all $\xi > 0$, then $\bar{x}(\tau) = 0$.

Proof. See [1, Appendix B]. □

Note that since the entries of A are i.i.d, the assumption over $\{P_\ell\}$ is made without loss of generality: the $\{P_\ell\}$ coefficients could simply be reordered to meet this assumption.

We can interpret this lemma as follows: in the large system limit, all sections ℓ where $\ell \leq \lfloor \xi^*(\tau_t) L \rfloor$ will be correctly decoded in step $(t+1)$, while all sections where this is not the case will not be decodable. Consequently, $\xi^*(\tau_t)$ tracks the overall proportion of sections which will be decoded at time $(t+1)$. This represents a key measure of the performance of the AMP decoder.

We now specify a specific power allocation. Set

$$P_\ell = P \cdot \frac{2^{2\mathcal{C}/L} - 1}{1 - 2^{-2\mathcal{C}}} \cdot 2^{-2\mathcal{C}\ell/L}, \quad \ell \in [L]. \tag{2.46}$$

We can now find the limiting value of $L P_\ell$ as

$$\lim L P_{\lfloor \xi L \rfloor} = \sigma^2 (1 + \mathsf{snr})^{1-\xi} \ln(1 + \mathsf{snr}). \tag{2.47}$$

We can combine this with Lemma 2.1 to find an expression for $\bar{x}$, in Lemma 2.2.

Lemma 2.2. *[1, Lemma 2] For the power allocation $\{P_\ell\}$ given in (2.46), we have for $t = 0, 1, \ldots$:*

$$\bar{x}_t := \lim x_t = \frac{(1+\mathit{snr}) - (1+\mathit{snr})^{1-\xi_{t-1}}}{\mathit{snr}}, \tag{2.48}$$

$$\bar{\tau}_t^2 := \lim \tau_t^2 = \sigma^2 + P(1-\bar{x}_t) = \sigma^2 (1+\mathit{snr})^{1-\xi_{t-1}} \tag{2.49}$$

where $\xi_{-1} = 0$, and for $t \geq 0$,

$$\xi_t = \min\left\{ \left(\frac{1}{2\mathcal{C}} \log\left(\frac{\mathcal{C}}{R}\right) + \xi_{t-1} \right),\ 1 \right\}. \tag{2.50}$$

Proof. See [1, Appendix C]. □

This lemma has a useful interpretation: ξ_t increases at each iteration by $\frac{1}{2\mathcal{C}} \log\left(\frac{\mathcal{C}}{R}\right)$ until it reaches 1, which means that the state evolution, for this power allocation, and so long as $\log\left(\frac{\mathcal{C}}{R}\right) > 0$ (i.e., so long as $R < \mathcal{C}$), predicts that all sections will eventually become decodable. Furthermore, the number of steps until this happens is given by

$$T^* = \left\lceil \frac{2\mathcal{C}}{\log(\mathcal{C}/R)} \right\rceil. \tag{2.51}$$

We have established that, with the exponentially decaying power allocation in (2.46), the asymptotic state evolution describes a process where all sections are decoded in a finite number of iterations. The final step is to show that the AMP decoder itself follows this state evolution, which is stated in the following theorem.

Theorem 2.1. *[1, Theorem 1] Fix any rate $R < \mathcal{C}$, and $a > 0$. Consider a sequence of rate R SPARCs $\{\mathcal{S}_n\}$ indexed by block length n, with design matrix parameters L and $M = L^a$ determined according to $nR = L \log L$, and an exponentially decaying power allocation given by (2.46). Then the section error rate of the AMP decoder (described in (2.5)–(2.6), and run for T^* steps) converges to zero almost surely, i.e., for any $\epsilon > 0$,*

$$\lim_{n_0 \to \infty} P\left(\mathcal{E}_{sec}(\mathcal{S}_n) < \epsilon,\ \forall n \geq n_0\right) = 1. \tag{2.52}$$

Proof. See [1, Section V]. □

This strong theoretical result shows that the AMP decoder asymptotically achieves the channel capacity with the exponentially decaying power allocation in (2.46). In fact, the theorem shows that for a given $R < \mathcal{C}$, any power allocation for which $\bar{x}_t$ converges to 1 in a finite number of iterations will enable reliable decoding in the large system limit, i.e., for any $\epsilon > 0$, the probability $\mathbb{P}(\mathcal{E}_{\text{sec}} > \epsilon)$ will go to zero. A result converting the asymptotic result of Theorem 2.1 to a large deviations bound that shows how the probability of error decays with L, M, n is given in [24, 41].

Despite these strong theoretical guarantees for very large block lengths, we will see that the exponential power allocation does not yield very small section error rates (e.g., better than 10^{-3}) at rates with practical block lengths on the order of a few thousands. We will see in the next chapter that a judicious choice of power allocation can yield several orders of magnitude improvement in the section error rates at such finite block lengths.

Chapter 3

Design Techniques to Improve Finite Length Performance

3.1 Introduction

The AMP decoder for SPARCs was introduced in Chapter 2, but the focus of that chapter was on performance in the limit of large block length. In this chapter, we instead focus on design techniques for improved *finite length* performance. Two main objectives are decreased computational complexity, so that the AMP decoder might be feasibly implemented for practical block lengths, and a small section error rate $\mathcal{E}_{\text{sec}}$. We focus on section error rate instead of the codeword error rate $\mathcal{E}_{\text{cw}}$ because if a good $\mathcal{E}_{\text{sec}}$ can be achieved, a high-rate outer code could be used to give good $\mathcal{E}_{\text{cw}}$, as described in Chapter 4. This chapter is organised as follows:

- We first consider, in Section 3.2, the use of Hadamard design matrices. These replace the normally distributed and i.i.d. matrices of the original design with a construction based on deterministic Hadamard matrices, as the Hadamard matrix product may be computed using a fast transform, similar to the Fast Fourier Transform. This provides a substantial reduction in time and memory complexity, with no impact on error rates.
- In Section 3.3 we briefly discuss an implementation challenge associated with the large exponentials involved in computing η_i^t, and suggest a solution which preserves sufficient accuracy while being efficient to compute.
- In Section 3.4 we investigate alternative power allocations to the simple exponential

allocation used to prove that the AMP decoder is capacity-achieving. While the exponential allocation is optimal in the large system limit, at finite block lengths it does not give a sharp reduction in error rates as we back off the rate from capacity. Two alternatives are proposed, which obtain greatly improved error performance for practical block lengths.

- In Section 3.5 we analyse some further effects of the power allocation on the *concentration* of error performance. Some power allocations often give excellent performance, with many trials showing no errors at all, but at the cost of very rare high-error decodes. Other choices of power allocation instead give worse average performance, but do not exhibit occasional high-error-count decodes. This permits a trade-off between $\mathcal{E}_{\text{cw}}$ and $\mathcal{E}_{\text{sec}}$. We also investigate a similar effect based on the choice of the design matrix parameters L and M. This analysis leads to strategies to find optimum choices of L, M for low section error rates.

- In Section 3.6 we describe an online estimator of the key SE parameter τ_t^2, which improves error performance and allows an early-stopping criterion which can often lead to dramatic reductions in computational time required to decode a codeword. Furthermore, this estimator enables us to accurately estimate the actual section error rate at the end of the decoding process.

- Finally, in Section 3.7, we derive simple expressions to estimate $\mathcal{E}_{\text{sec}}$ and $\mathcal{E}_{\text{cw}}$ without requiring the full SE recursions. These new expressions accurately track the empirical performance as long as a good concentration is obtained, but are not able to capture the effects of poor concentration on the error performance.

Taken together, these improvements represent multiple orders of magnitude reduction in computational time and memory complexity, while providing orders of magnitude improvement in error performance at rates moderately away from capacity. Table 3.1 shows the cumulative improvement for a typical operating point.

Category	Before	After
CPU Time	12 600 s	4.21 s
Memory	25.7 GB	<0.01 GB
Bit Error Rate	1.11×10^{-2}	4.52×10^{-7}

Table 3.1: Comparison of naïve SPARC-AMP (before) with the improvements described in this chapter (after). The SPARC parameters are $L = 1024$, $M = 512$, $R = 1.4$, $n = 6583$, $P = 15$, $\sigma^2 = 1$.

3.2 Reducing Decoding Complexity via Random Hadamard Design Matrices

In theoretical analyses of sparse regression codes, the design matrix A is chosen to have zero-mean i.i.d. entries, either Gaussian $\sim \mathcal{N}(0, \frac{1}{n})$ as in [1, 22, 42], or Bernoulli entries drawn uniformly from $\pm\frac{1}{\sqrt{n}}$ [43]. With such matrices, the computational complexity of the AMP decoder in (2.5)–(2.10) is $O(LMn)$ when the matrix-vector multiplications $A\beta$ and A^*z^t are performed in the usual way. Additionally, storing A requires $O(LMn)$ memory, which is prohibitive for reasonable code lengths. For example, $L = 1024$, $M = 512$, $n = 9216$ ($R = 1$ bit) requires 18 gigabytes of memory using a 4-byte floating point representation, all of which must be accessed twice per iteration.

To reduce decoding complexity, we replace the i.i.d. design matrix with a structured Hadamard-based design matrix, which we denote in this section by A_{H}. With A_{H}, the key matrix-vector multiplications can be performed via a fast Walsh-Hadamard Transform (FWHT) [44]. Moreover, A_{H} can be implicitly defined which greatly reduces the memory required.

Construction of A_{H}

We first consider the underlying Hadamard matrix H which is defined as follows. Let N be a power of 2, and let $m = \log_2 N$. With $H_0 = 1$, recursively define the $N \times N$ matrix H_m as

$$H_m = \begin{pmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{pmatrix}. \tag{3.1}$$

This matrix is square, with ± 1 entries and mutually orthogonal rows. It can be viewed as an analogue of the Fourier matrix, with each row and column representing a basis vector at a different digital frequency. The first row and column are all $+1$, a complication which we will

need to account for. All other rows and columns have equal counts of 1 and -1 entries, and consequently a mean of 0 and a norm of N. We have defined an H_m which is $2^m \times 2^m$, but we require an $n \times LM$ matrix for A_{H}. Additionally we require that A_{H} is random and each column has norm 1. One option to form A_{H} is to set $m = \lceil \log_2(\max(LM+1, n+1)) \rceil$, and select n truncated rows uniformly at random from the $2^m \times 2^m$ Hadamard matrix H_m to define A_{H}. To avoid the all-1 initial row, we avoid ever selecting the first row when defining A_{H}. By truncating the left side of the rows, we avoid ever including entries from the first column. These two measures ensure all columns of A_{H} will have zero mean. Finally we consider each entry to be divided by $\sqrt{n}$ to give each column a norm of 1. In practice, this division is omitted, and all equations involving A are modified accordingly, which often removes a few additional computational steps. See Figure 3.1 for an illustration.

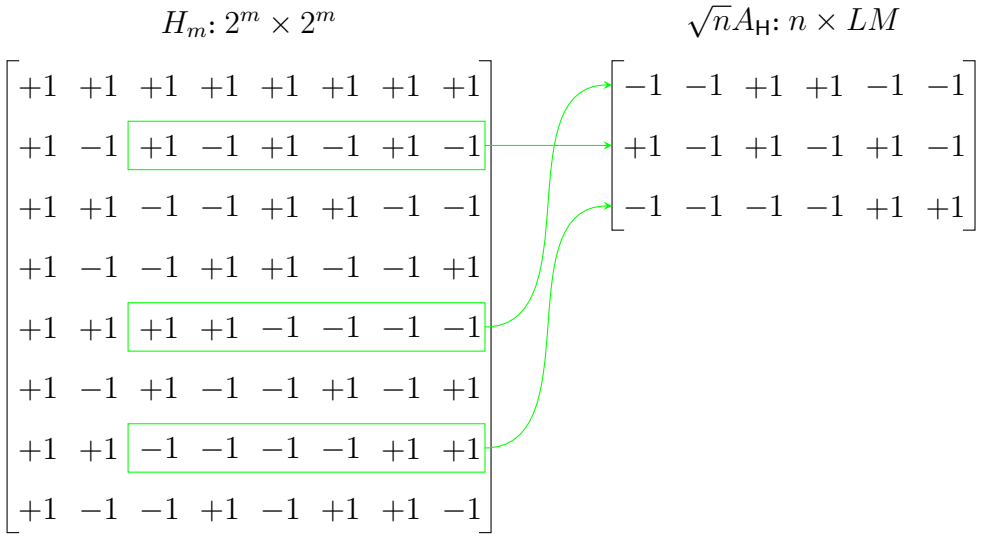


Figure 3.1: The first strategy for forming A_{H} from the Hadamard matrix H_m. Here we use a single H_m and select A_{H} as random truncated rows, excluding the first row as it is all $+1$. Note that by truncating on the left hand side we also avoid including entries from the first column of H_m, which is also all $+1$.

Computation of $A_{\mathsf{H}}\beta$ and $A_{\mathsf{H}}^{*}z$

As mentioned above, we do not actually form A_{H} explicitly. Instead, to compute $A_{\mathsf{H}}\beta$, we form the 2^m long vector $\tilde{\beta}$ by prepending 0 to β; evaluate $H_m\tilde{\beta}$ using the FWHT; and then select the same rows from the 2^m-long result as were used to define A_{H}. Finally we divide each entry by $\sqrt{n}$ to provide for A_{H} having a column norm of 1. This gives an n-long result, as required, which is equal to $A_{\mathsf{H}}\beta$. Figure 3.2 demonstrates this strategy.

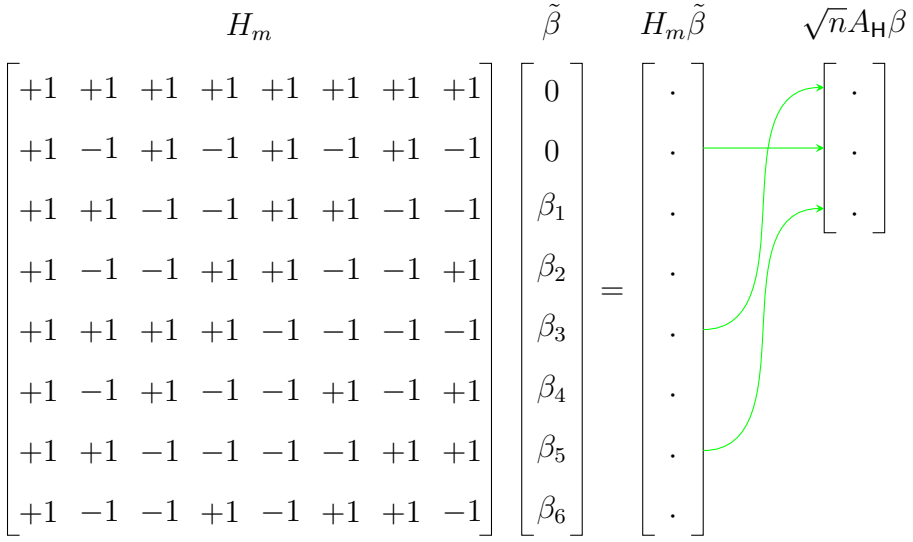


Figure 3.2: Finding $A_{\mathsf{H}}\beta$ using the FWHT rather than explicit computation. $H_m\tilde{\beta}$ is computed as $\text{FWHT}(\tilde{\beta})$.

Taking a similar approach for $A_{\mathsf{H}}^{*}z$, we first embed z into an 2^m-long vector $\tilde{z}$ by setting the entries in $\tilde{z}$ which correspond to the rows used to define A_{H} to the respective entries in z, and set the remaining entries to 0. Then we compute $H_m^{*}\tilde{z}$ using the FWHT, divide each entry by $\sqrt{n}$, and select the final LM entries to find $A_{\mathsf{H}}^{*}z$. This is illustrated in Figure 3.3. Note that by *prepending* the 0s for $A_{\mathsf{H}}\beta$, and then taking the *final* entries when finding $A_{\mathsf{H}}^{*}z$, we avoid any entries which involve the first column of H_m, reflecting the construction of A_{H}. Note also that since H_m is symmetric, we find $H_m^{*}\tilde{z}$ by simply taking the regular FWHT of $\tilde{z}$.

Improved Construction of A_{H}

A downside to the simple construction of A_{H} from H_m is that since A is typically much wider than it is tall (i.e., $LM \gg n$), the required H_m contains many more rows than are required,

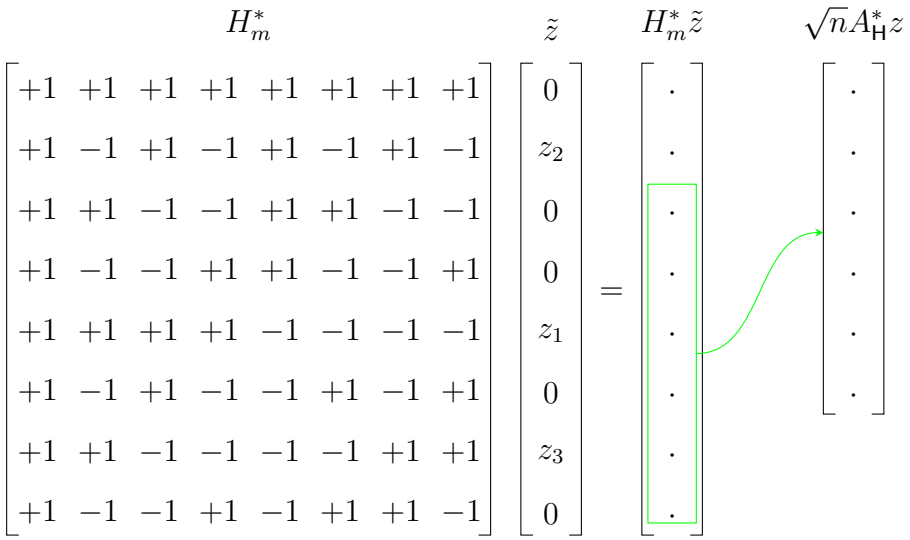


Figure 3.3: Finding A_{H}^*z using the FWHT rather than explicit computation. $H_m^*\tilde{z}$ is computed as $\mathrm{FWHT}(\tilde{z})$.

and so the FWHT computations are larger than necessary and most of the result is discarded each time. To improve on this, we can split the definition of A_{H} into many smaller Hadamard matrices, and compute the relevant products by combining many smaller FWHT operations. This is done as follows.

Take $k = \lceil \log_2(\max(n+1, M+1)) \rceil$. Each section of A_{H} is constructed independently by choosing a permutation of n distinct rows from H_k uniformly at random. We do not pick the first row of H_k as it is all-ones. The $n+1$ in the definition of k ensures that we still have enough rows left to pick n at random after removing the first, all-one, row; the $M+1$ ensures that we can always have one leading 0 when embedding β so that the first, all-one, column is also never picked.

The multiplications $A_{\mathsf{H}}\beta$ and A_{H}^*z are performed by computing $A_{\mathsf{H}\ell}\beta_\ell$ and $A_{\mathsf{H}\ell}^*z$, for $\ell \in [L]$, where the $n \times M$ matrix $A_{\mathsf{H}\ell}$ is the ℓth section of A_{H}, and $\beta_\ell \in \mathbb{R}^M$ is the ℓth section of β. Each section is computed in the same way as before: to compute $A_{\mathsf{H}\ell}\beta_\ell$, zero-prepend β_ℓ to length 2^k, perform the FWHT, then choose n entries corresponding to the rows in $A_{\mathsf{H}\ell}$. Sum the n-length result from each section to obtain $A_{\mathsf{H}}\beta$. Note that we prepend with 0 because the first column of H_k must always be ignored as it is always all-ones. For $A_{\mathsf{H}\ell}^*z$, embed entries from z into a 2^k long vector again corresponding to the rows in A_{H}, with all other entries set to zero, perform the FWHT, and return the last M entries. Concatenate the result from each

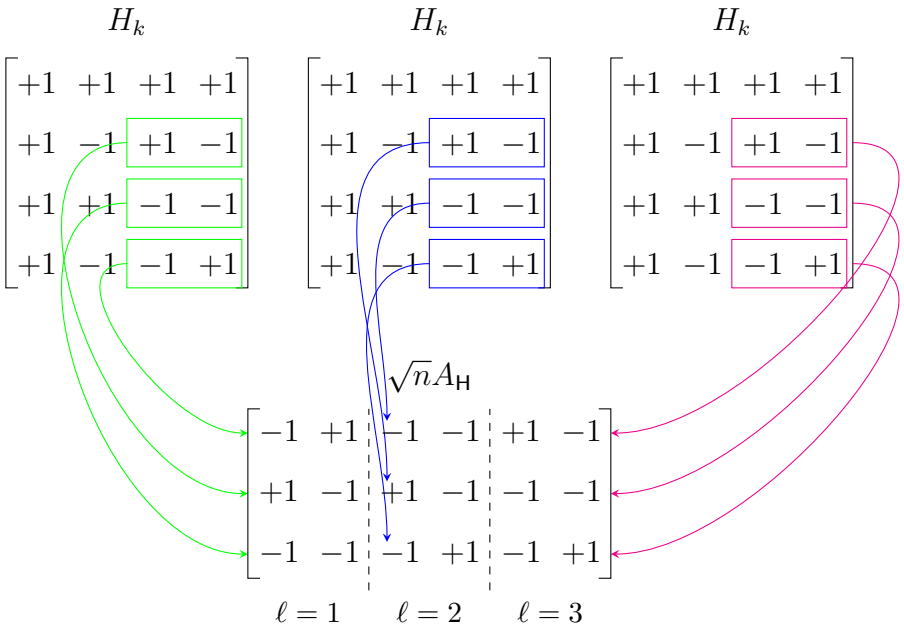


Figure 3.4: The improved strategy for forming A_{H} from many smaller Hadamard matrices H_k. We use a different set of random rows for each section $\ell \in [L]$ of A_{H}, all drawn from the same smaller H_k, again excluding the first row and truncating the rows to exclude the first column. Note that in this very small example there are only three usable rows, so the available permutations are limited, but in practical situations there are thousands of rows available.

section to form $A_{\mathsf{H}}^* z$.

It would be possible to extend this concept even further, by constructing each section of A_{H} out of multiple even smaller Hadamard matrices, with $k = \lceil \log_2(M+1) \rceil$ and using $\lceil \frac{n}{2^k} \rceil$ such matrices. The same techniques to perform the required multiplications apply, and this technique is even more efficient computationally. However, it introduces additional implementation complexity, and care must be taken that the smaller Hadamard matrices still have a sufficient number of row permutations so that the columns of A_{H} remain independent.

The Fast Walsh-Hadamard Transform

The FWHT, described in [44], is similar to the Fast Fourier Transform [45], and amenable to similar efficient implementations in software and in hardware. However, unlike the Fast Fourier Transform, no multiplications are required, and no twiddle factors are necessary to re-index the input at each stage, permitting the efficient Algorithm 3.1 to be used.

Algorithm 3.1 In-place Fast Walsh-Hadamard Transform

Require: A length-N vector x, where $N = 2^m$, which will be transformed in-place.
  for $i = 2^{m-1}, 2^{m-2}, ..., 2^0$ **do**
    for $k = 0, 2i, \ldots, N$ **do**
      for $j = k, k+1, \ldots, k+i$ **do**
        $u \leftarrow x_j$
        $v \leftarrow x_{i|j}$
        $x_j \leftarrow x_j + v$
        $x_{i|j} \leftarrow u - v$
      end for
    end for
  end for
  return x

In the above algorithm $i|j$ is taken to mean "the bit-wise logical OR of the binary representations of i and j".

Performance of A_{H}

SPARC codewords should have entries which are well described by a Gaussian distribution. To verify that this remains the case when A_{H} is used, an $n = 8192$ codeword was generated using both a full Gaussian A matrix and using the Fast Walsh-Hadamard Transform described above. Figure 3.5 shows a probability plot for the entries of each codeword. Both are extremely well matched to a Guassian distribution and there is nothing to discern the Hadamard-based codeword from the full Gaussian. This similarly and goodness of fit remains true even for atypically small values of L and M such as $L = 32$, $M = 16$.

The error rate performance of the AMP decoder with A_{H} constructed as described above is likewise indistinguishable from the full i.i.d. $\sim \mathcal{N}(0, \frac{1}{n})$ matrix, once $\max(n+1, M+1)$ exceed a small threshold. This has been verified across numerous simulations, where the resulting difference in performance between the two matrices is smaller than the difference between successive random trials. The use of Hadamard matrices therefore does not come at any cost to error performance.

The computational complexity of the decoder is reduced to $O(Ln \log n)$ (in the common case where $n > M$, otherwise, it is $O(LM \log M)$). The memory requirements are reduced to $O(LM)$, typically a few megabytes. In comparison, for i.i.d. design matrices, the complexity and memory requirements scale as $O(LMn)$. For reasonable code lengths, this represents

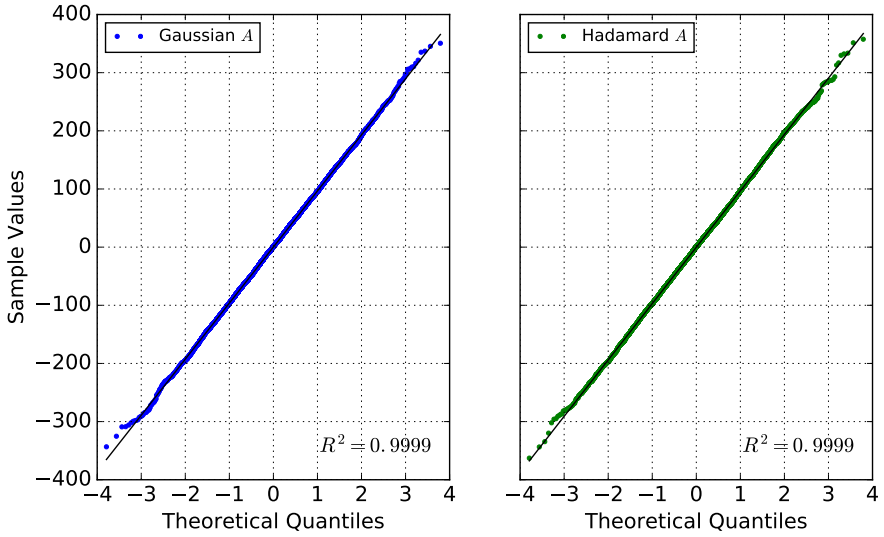


Figure 3.5: Probability plot comparison of codeword entries from full Gaussian A-matrix compared to the Hadamard A_{H} matrix. Ideal Gaussian variates would lie on the straight black lines. The correlation coefficient for the straight-line fit is also plotted. For this codeword, $L = 1024, M = 512, n = 8192$.

around a thousandfold improvement in both time and memory. Furthermore, the easily parallelised structure would enable a hardware implementation to trade off between a slower and smaller series implementation and a faster though larger parallel implementation, potentially leading to significant practical speedups.

3.3 Numerical Regularisation in η_i^t

The AMP decoder is defined by two recursions, (2.5) for z^t and (2.6) for β_i^{t+1}, which must be computed at each iteration. The expression for β_i^{t+1} requires the computation of η_i^t, given in (2.10), and repeated here for convenience:

$$\eta_i^t(s = A^* z^t + \beta^t) = \sqrt{nP_\ell}\frac{\exp\left(s_i \frac{\sqrt{nP_\ell}}{\tau_t^2}\right)}{\sum_{j\in \mathrm{sec}(i)} \exp\left(s_j \frac{\sqrt{nP_\ell}}{\tau_t^2}\right)}, \quad 1 \le i \le ML. \tag{3.2}$$

The quantities $\{s_i\}_{i\in[ML]}$ may be as large as $\sqrt{nP_\ell}$ towards the end of decoding. The exponentials in (3.2) may therefore be as large as $\frac{nP_\ell}{\tau_t^2}$, which is unbounded. Using a typical IEEE 754

double precision floating point representation, the largest integer x such that $\exp(x)$ is representable is 709. Values of $s_i \frac{\sqrt{nP_\ell}}{\tau_t^2}$ which exceed this are often encountered when running numerical AMP simulations. If unhandled, this may either cause an error in the simulation implementation, or may generate a unrepresentable value which propagates through the rest of the simulation.

One possible solution is to use higher precision floating point representations, which permit an extended range of exponents, but come at a cost of significantly reduced simulation throughput and are often non-standard implementations. Another option is to saturate the arguments to exp, assuming that whenever one s_i in a section would require saturating, the remaining s_j for that same section will all be small, thus the saturation would have minimal impact on accuracy. Since the elements of each section represent a probability distribution, this will generally be the case. The computational cost of performing this saturation depends on the target architecture, and if the cost is low, this may be an acceptable method.

Algorithm 3.2 Non-Overflowing Computation of $\eta_i^t(s)$ for section ℓ

Require: Input vector $\{s_j\}_{j \in \text{sec}_\ell}$
  Initialise S to 0
  for $j = 1, \ldots, M$ **do**
    $u_j \leftarrow s_j \frac{\sqrt{nP_\ell}}{\tau_t^2}$
    $\mathsf{S} \leftarrow \max(\mathsf{S}, u_j)$
  end for
  Initialise e_Σ to 0
  for $j = 1, \ldots, M$ **do**
    $\mathsf{e}_j \leftarrow \exp(u_j - \mathsf{S})$
    $\mathsf{e}_\Sigma \leftarrow \mathsf{e}_\Sigma + \mathsf{e}_j$
  end for
  for $j = 1, \ldots, M$ **do**
    $\eta_j^t \leftarrow \sqrt{nP_\ell} \cdot \mathsf{e}_j / \mathsf{e}_\Sigma$
  end for
  return $\{\eta_j^t\}_{j \in \text{sec}_\ell}$

A third option is to rewrite (3.2) so that the arguments to exp never exceed 0. We do this as follows. First, compute

$$\mathsf{S} = \max_{j \in \text{sec}(i)} s_j \frac{\sqrt{nP_\ell}}{\tau_t^2}, \tag{3.3}$$

and then use $e^x = e^{x-\mathsf{S}}e^{\mathsf{S}}$ to write (3.2) as

$$\eta_i^t(s = A^* z^t + \beta^t) = \frac{\sqrt{nP_\ell}\exp(\mathsf{S})\exp\left(s_i\frac{\sqrt{nP_\ell}}{\tau_t^2} - \mathsf{S}\right)}{\exp(\mathsf{S})\sum_{j\in \mathrm{sec}(i)}\exp\left(s_j\frac{\sqrt{nP_\ell}}{\tau_t^2} - \mathsf{S}\right)}, \quad 1 \le i \le ML, \qquad (3.4)$$

noting that the e^{S} terms cancel out. Any arguments which were previously very small compared to S will become too small to represent after exponentiation and be rounded to 0. Since those terms were much smaller than S to begin with, this loss of accuracy does not impact the result. Effectively, we have saturated the lower range instead of the upper range discussed above, which maintains better accuracy around the larger arguments. Furthermore, note that the denominator is the sum of all the numerators for each s_j in that section. These terms can be computed once per section to further reduce computation time. Our resulting algorithm to compute $\eta^t(s)$ for each section $\ell \in [L]$ is given in Algorithm 3.2.

3.4 Power Allocation

Figure 3.6 shows the section error rate for the AMP decoder using the exponential power allocation given in (2.46). This allocation achieves the channel capacity asymptotically, but for finite block lengths the error rates decrease very slowly as the rate backs off from capacity, and the overall error rate performance is poor. We would like to find better power allocations for rates away from capacity, for example around $R = 0.7\mathcal{C}$.

Before introducing the power allocation scheme, we briefly give a reminder of some intuition about the AMP update rules (2.5)–(2.10), and the SE recursion in (2.7)–(2.9). The update step (2.6) to generate each estimate of β is underpinned by the following key property: after step t, the "effective observation" $\beta^t + A^* z^t$ is approximately distributed as $\beta + \tau_t Z$, where Z is a standard normal random vector independent of β. Thus τ_t^2 is the effective noise variance at the end of step t.

We see from (2.7) that the effective noise variance τ_t^2 is the sum of two terms. The first is the channel noise variance σ^2. The other term $P(1 - x(\tau_{t-1}))$ can be interpreted as the interference due to the undecoded sections in β^t. Equivalently, $x(\tau_{t-1})$ is the expected power-weighted fraction of sections which are correctly decodable at the end of step t.

The starting point for our power allocation design is the following result from [24], which gives analytic upper and lower bounds for $x(\tau)$ of (2.7).

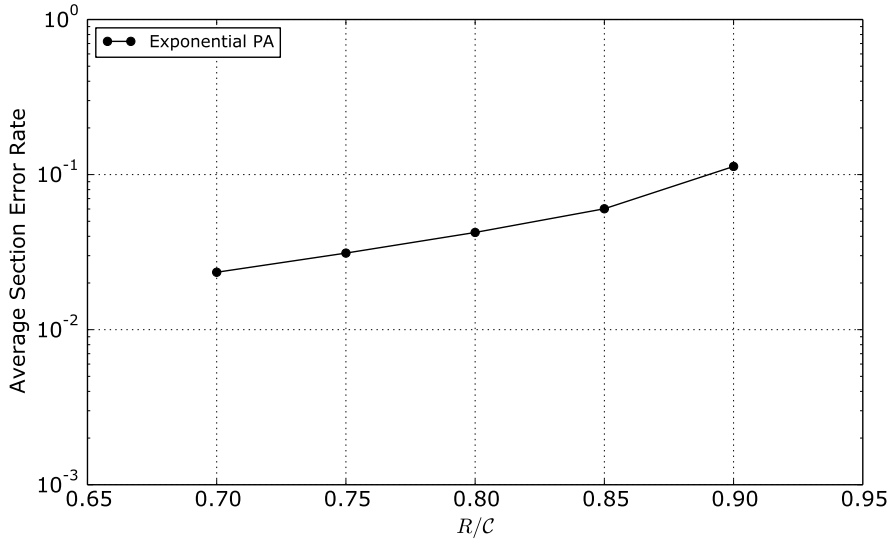


Figure 3.6: AMP section error rate $\mathcal{E}_{\text{sec}}$ vs $R/\mathcal{C}$ at $\mathsf{snr} = 15, \mathcal{C} = 2$ bits. The SPARC parameters are $M = 512, L = 1024$. The black curve shows the $\mathcal{E}_{\text{sec}}$ with $P_\ell \propto 2^{-2\mathcal{C}\ell/L}$.

Lemma 3.1. *[24, Lemma 1(b)] Let* $\nu_\ell := \frac{LP_\ell}{R\tau^2 \ln 2}$. *For sufficiently large* M, *and for any* $\delta \in (0, \frac{1}{2})$,

$$x(\tau) \leq \sum_{\ell=1}^{L} \frac{P_\ell}{P} \mathbf{1}\left\{\nu_\ell > 2 - \delta\right\} + M^{-\kappa_1 \delta^2} \sum_{\ell=1}^{L} \frac{P_\ell}{P} \mathbf{1}\left\{\nu_\ell \leq 2 - \delta\right\}, \tag{3.5}$$

$$x(\tau) \geq \left(1 - \frac{M^{-\kappa_2 \delta^2}}{\delta\sqrt{\ln M}}\right) \sum_{\ell=1}^{L} \frac{P_\ell}{P} \mathbf{1}\left\{\nu_\ell > 2 + \delta\right\}. \tag{3.6}$$

where κ_1, κ_2 *are universal positive constants.*

We note that when in the large system limit, as $M \to \infty$, Lemma 3.1 reduces to the asymptotic result Lemma 2.1. As the constants κ_1, κ_2 in (3.5)–(3.6) are not precisely specified, for designing power allocation schemes, we use the following approximation for $x(\tau)$:

$$x(\tau) \approx \sum_{\ell=1}^{L} \frac{P_\ell}{P} \mathbf{1}\left\{LP_\ell > 2R\tau^2 \ln 2\right\}. \tag{3.7}$$

This approximate version, which is increasingly accurate as L, M grow large, is useful for

gaining intuition about suitable power allocations. If the effective noise variance after step t is τ_t^2, then (3.7) says that any section ℓ whose normalised power LP_ℓ is larger than the threshold $2R\tau_t^2 \ln 2$ is likely to be decodable correctly in step $(t+1)$, i.e., in β^{t+1}, the probability mass within the section will be concentrated on the correct non-zero entry. For a given power allocation, we can iteratively estimate the SE parameters $(\tau_t^2, x(\tau_t^2))$ for each t using the approximation in (3.7). This provides a way to quickly check whether or not a given power allocation will lead to reliable decoding in the large system limit. For reliable decoding at a given rate R, the effective noise variance given by $\tau_t^2 = \sigma^2 + P(1 - x(\tau_{t-1}))$ should decrease with t until it reaches a value close to σ^2 in a finite number of iterations. Equivalently, $x(\tau_t)$ in (3.7) should increase to a value very close to 1.

For a rate $R < \mathcal{C}$, there are infinitely many power allocations for which (3.7) predicts successful decoding in the large system limit. However, as illustrated below, their finite length error performance may differ significantly. Thus the key question addressed in this section is: *how do we choose a power allocation that gives the lowest section error rate*?

Let us first consider the exponentially-decaying power allocation given by

$$P_\ell = \frac{P(2^{2\mathcal{C}/L} - 1)}{1 - 2^{-2\mathcal{C}}} 2^{-2\mathcal{C}\ell/L}, \quad \ell \in [L], \tag{3.8}$$

This power allocation was proven in [1] to be asymptotically capacity-achieving in the large system limit, i.e., it was shown that the section error rate $\mathcal{E}_{\text{sec}}$ of the AMP decoder converges almost surely to 0 as $n \to \infty$, for any $R < \mathcal{C}$. However, it does not perform well at practical block lengths, which motivated the search for alternatives. We now evaluate it in the context of (3.7) to better explain the development of a new power allocation scheme.

Given a power allocation, using (3.7) one can compute the minimum required power for any section $\ell \in [L]$ to decode, assuming that the sections with higher power have decoded correctly. The dashed lines in Figure 3.7 shows the minimum power required for each section to decode (assuming the exponential allocation of (3.8) for the previous sections), for $R = \mathcal{C}$ and $R = 0.7\mathcal{C}$. The figure shows that the power allocation in (3.8) matches (up to order $\frac{1}{L}$ terms) with the minimum required power when $R = \mathcal{C}$. However, for $R = 0.7\mathcal{C}$, we see that the exponentially-decaying allocation allocates significantly more power to the earlier sections than the minimum required, compared to later sections. This leads to relatively high section error rates, as shown in Figure 3.6.

Figure 3.7 also shows that the total power allocated by the minimal power allocation at

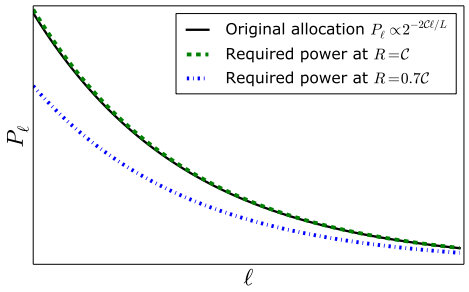


Figure 3.7: The dashed lines show the minimum required power in section for successful decoding when $R = \mathcal{C}$ (upper), and $R = 0.7\mathcal{C}$ (lower), where $\mathcal{C} = 2$ bits. The solid line shows the exponentially-decaying power allocation in (3.8).

$R = 0.7\mathcal{C}$ is significantly less than the available power P. Therefore, the key question is: how do we balance the allocation of available power between the various sections to minimise the section error rate? Allocating excessive power to the earlier sections ensures they decode reliably early on, but then there will not be sufficient power left to ensure reliable decoding in the final sections. This is the reason for the poor finite length performance of the exponentially-decaying allocation. Conversely, if the power is spread too evenly then no section particularly stands out against the noise, so it is hard for the decoding to get started, and early errors can cause cascading failures as subsequent sections are also decoded in error.

3.4.1 Modified Exponential Power Allocation

This trade-off motivated the following modified exponential power allocation, originally proposed in [1]. We set

$$P_\ell = \begin{cases} \kappa \cdot 2^{-2a\mathcal{C}\ell/L} & 1 \le \ell \le fL, \\ \kappa \cdot 2^{-2a\mathcal{C}f} & fL+1 \le \ell \le L, \end{cases} \tag{3.9}$$

where the normalizing constant κ, chosen to ensure that $\sum_{\ell=1}^{L} P_\ell = P$, is

$$\kappa = \frac{P\left(2^{2a\mathcal{C}/L} - 1\right)}{1 - 2^{-2a\mathcal{C}f}\left(1 - L(1-f)(2^{2a\mathcal{C}/L} - 1)\right)}. \tag{3.10}$$

In (3.9), the parameter a controls the steepness of the exponential allocation, while the

parameter f flattens the allocation after the first fraction f of the sections. Smaller choices of a lead to less power allocated to the initial sections, making a larger amount available to the later sections. Similarly, smaller values of f lead to more power allocated to the final sections, while preventing the final few sections from having negligible power. See Figure 3.8 for an illustration.

In order to run the SPARC AMP decoder with this power allocation, closed-form estimates for τ_t^2 can be obtained using Lemma 2.1. Using the definition of $x_t = x(\tau_t)$ from (3.7), initialise $\tau_0^2 = \sigma^2 + P$, and for $t \geq 0$ we have:

$$\tau_{t+1}^2 = \sigma^2 + P(1 - x_{t+1}), \qquad x_{t+1} = \frac{1 - 2^{-2a\mathcal{C}\xi_t}}{1 + 2^{-2a\mathcal{C}f}(2\ln 2 a\mathcal{C}(1-f) - 1)}, \tag{3.11}$$

where

$$\xi_t = \frac{1}{2a\mathcal{C}} \log_2 \left(\frac{2^{2a\mathcal{C}f} \cdot a\mathcal{C}P}{R\tau_t^2(2^{2a\mathcal{C}f} + (1-f)(2a\mathcal{C}\ln 2) - 1)} \right). \tag{3.12}$$

While this allocation improves the section error rate by up to a few orders of magnitude (see Figure 3.11), it requires costly numerical optimization of a and f. A good starting point is to use $a = f = R/\mathcal{C}$, but further optimization is generally necessary. This motivates the need for a fast power allocation algorithm with fewer tuning parameters.

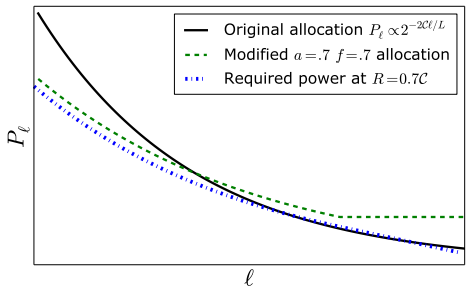


Figure 3.8: The modified power allocation with $a = f = 0.7$ results in slightly more than the minimum power required for the first 70% of sections; the remaining available power is allocated equally among the last 30% of sections. The original allocation with $P_\ell \propto 2^{-2\mathcal{C}\ell/L}$ is also shown for comparison.

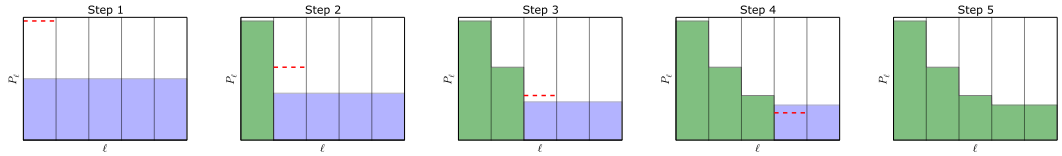


Figure 3.9: Example illustrating the iterative power allocation algorithm with $B = 5$. In each step, the height of the blue region represents the allocation that distributes the remaining power equally over all the remaining sections. The dashed red line indicates the minimum power required for decoding the current block of sections. The green bars represent the power that has been allocated at the beginning of the current step.

3.4.2 Iterative Power Allocation

We now describe a simple parameter-free iterative algorithm to design a power allocation. The L sections of the SPARC are divided into B *blocks* of L/B sections each. Each section within a block is allocated the same power. For example, with $L = 512$ and $B = 32$, there are 32 blocks with 16 sections per block. The algorithm sequentially allocates power to each of the B blocks as follows. Allocate the minimum power to the first block of sections so that they can be decoded in the first iteration when $\tau_0^2 = \sigma^2 + P$. Using (3.7), we set the power in each section of the first block to

$$P_\ell = \frac{2R\tau_0^2 \ln 2}{L}, \quad 1 \le \ell \le \frac{L}{B}.$$

Using (3.7) and (2.7), we then estimate $\tau_1^2 = \sigma^2 + (P - BP_1)$. Using this value, allocate the minimum required power for the second block of sections to decode, i.e., $P_\ell = 2 \ln 2R\tau_1^2/L$ for $\frac{L}{B} + 1 \le \ell \le \frac{2L}{B}$. If we sequentially allocate power in this manner to each of the B blocks, then the total power allocated by this scheme will be strictly less than P whenever $R < \mathcal{C}$. We therefore modify the scheme as follows.

For $1 \le b \le B$, to allocate power to the bth block of sections assuming that the first $(b-1)$ blocks have been allocated, we compare the two options and choose the one that allocates higher power to the block: i) allocating the minimum required power (computed as above) for the bth block of sections to decode; ii) allocating the remaining available power equally to sections in blocks $b, \ldots, B$, and terminating the algorithm. This gives a flattening in the final blocks similar to the allocation in (3.9), but without requiring a specific parameter that determines where the flattening begins. The iterative power allocation routine is described in Algorithm 3.3.

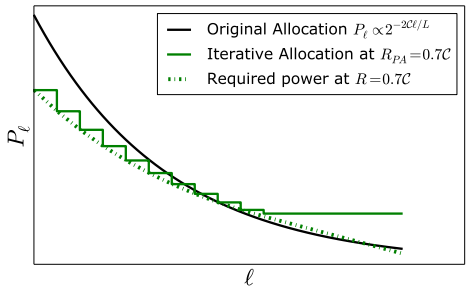


Figure 3.10: Iterative allocation, with $L = 512$, and $B = 16$ blocks. Flattening occurs at the 11th block.

Algorithm 3.3 Iterative power allocation routine

Require: L, B, σ^2, P, R such that B divides L.
  Initialise $k \leftarrow \frac{L}{B}$
  for $b = 0$ to $B - 1$ **do**
    $P_{\text{remain}} \leftarrow P - \sum_{\ell=1}^{bk} P_\ell$
    $\tau^2 \leftarrow \sigma^2 + P_{\text{remain}}$
    $P_{\text{block}} \leftarrow 2\ln(2)R\tau^2/L$
    if $P_{\text{remain}}/(L - bk) > P_{\text{block}}$ **then**
      $P_{bk+1}, \ldots, P_L \leftarrow P_{\text{remain}}/(L - bk)$
      break
    else
      $P_{bk+1}, \ldots, P_{(b+1)k} \leftarrow P_{\text{block}}$
    end if
  end for
  return $P_1, \ldots, P_L$

Figure 3.9 shows a toy example building up the power allocation for $B = 5$, where flattening is seen to occur in step 4. Figure 3.10 shows a more realistic example with $L = 512$ and $R = 0.7\mathcal{C}$.

Choosing B: By construction, the iterative power allocation scheme specifies the number of iterations of the AMP decoder in the large system limit. This is given by the number of blocks with distinct powers; in particular the number of iterations (in the large system limit) is of the order of B. For finite code lengths, we find that it is better to use a termination criterion for the decoder based on the estimates generated by the algorithm. This criterion is described in Section 3.6. This data-driven termination criterion allows us to choose the number of blocks B

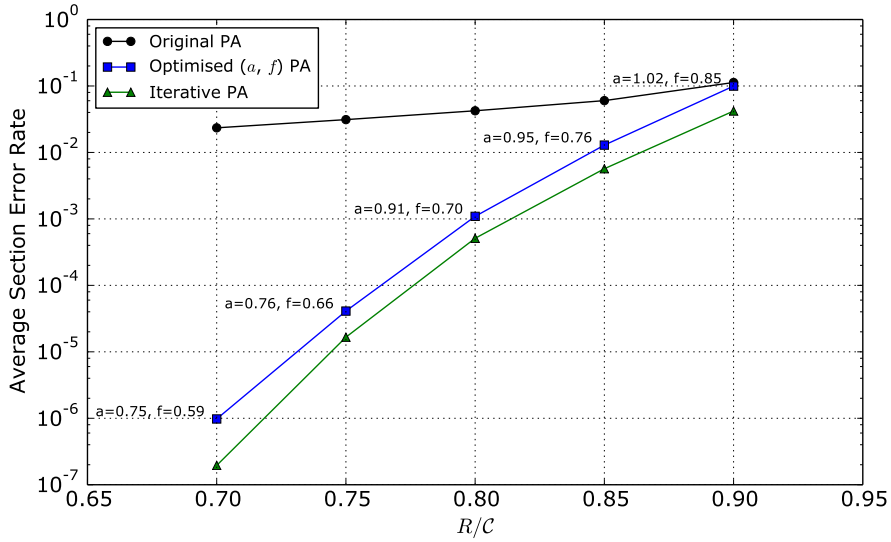


Figure 3.11: AMP section error rate $\mathcal{E}_{\text{sec}}$ vs $R/\mathcal{C}$ at $\mathsf{snr} = 15, \mathcal{C} = 2$ bits. The SPARC parameters are $M = 512, L = 1024$. The top curve shows the $\mathcal{E}_{\text{sec}}$ with $P_\ell \propto 2^{-2\mathcal{C}\ell/L}$, reproduced from Figure 3.6. The middle curve shows $\mathcal{E}_{\text{sec}}$ with the power allocation in (3.9) with the optimised (a, f) values shown. The bottom curve shows $\mathcal{E}_{\text{sec}}$ for the iterative power allocation.

to be as large as L. We found that choosing $B = L$, together with the termination criterion in Section 3.6, consistently gives a small improvement in error performance (compared to other choices of B), with no additional time or memory cost.

Figure 3.11 compares the error performance of the three power allocation schemes discussed above for different values of $R/\mathcal{C}$ at $\mathsf{snr} = 15$. The values of (a, f) for the modified exponential allocation in (3.9) were found via numerical optimization. We observe that the empirical section error performance with the iterative power allocation is the same or better than the optimised (a, f) scheme.

3.5 Error Concentration Trade-offs

In this section, we discuss how the choice of SPARC design parameters can influence the trade-off between the 'typical' value of section error rate and concentration of actual error rates around the typical values. The typical section error rate refers to that predicted by state evolution (SE). Using the SE equations (2.7)–(2.9), we iterate until convergence at iteration T,

obtaining a final value of τ_T. The value of $x(\tau_t)$ represents the power-weighted fraction of sections expected to have decoded correctly at iteration t; by removing the power-weighting $\frac{P_\ell}{P}$ and using τ_T we can instead estimate the overall fraction of sections to have decoded correctly at time T as:

$$\mathcal{E}_{\text{sec}}^{\text{SE}} := 1 - \frac{1}{L}\sum_{\ell=1}^{L} \mathbb{E}\left[\frac{\exp\left(\frac{\sqrt{nP_\ell}}{\tau_T}(U_1^\ell + \frac{\sqrt{nP_\ell}}{\tau_T})\right)}{\exp\left(\frac{\sqrt{nP_\ell}}{\tau_T}(U_1^\ell + \frac{\sqrt{nP_\ell}}{\tau_T})\right) + \sum_{j=2}^{M}\exp\left(\frac{\sqrt{nP_\ell}}{\tau_T}U_j^\ell\right)}\right]. \tag{3.13}$$

The concentration refers to how tightly distributed around the SE prediction $\mathcal{E}_{\text{sec}}^{\text{SE}}$ the observed section error rates are.

As we describe below, the choice of SPARC parameters (L, M) and the power allocation both determine a trade-off between obtaining a low value for $\mathcal{E}_{\text{sec}}^{\text{SE}}$, and concentration of the actual section error rate around $\mathcal{E}_{\text{sec}}^{\text{SE}}$. This trade-off is of particular interest when applying an outer code to the SPARC, as considered in Section 4.2, which may be able to reliably handle only a small number of section errors.

3.5.1 Effect of L and M on Concentration

Recall from (1.8) that the code length n at a given rate R is determined by the choice of L and M according to the relationship $nR = L \log M$. In general, L and M may be chosen freely to meet a desired rate and code length.

To understand the effect of increasing M, consider Figure 3.12 which shows the error performance of a SPARC with $R = 1.5, L = 1024$, as we increase the value of M. From (1.8), the code length n increases logarithmically with M. We observe that the section error rate (averaged over 200 trials) decreases with M up to $M = 2^9$, and then starts increasing. This is in sharp contrast to the SE prediction (3.13) which keeps decreasing as M is increased, plotted using a dashed line in Figure 3.12.

This divergence between the actual section error rate and the SE prediction for large M is due to large fluctuations in the number of section errors across trials. Recent work on the error exponent of SPARCs with AMP decoding shows that the concentration of error rates near the SE prediction is strongly dependent on both L and M. For $R < \mathcal{C}$, [24, Theorem 1] shows

that for any $\epsilon > 0$, the section error rate $\mathcal{E}_{\text{sec}}$ satisfies

$$\mathbb{P}\left(\mathcal{E}_{\text{sec}} > \mathcal{E}_{\text{sec}}^{\text{SE}} + \epsilon\right) \leq K_T \exp\left\{ \frac{-\kappa_T L}{(\log M)^{2T-1}} \left(\frac{\epsilon \sigma^2 \ln(1+\mathsf{snr})}{4} - \tau_0^2 f(M) \right)^2 \right\}, \tag{3.14}$$

where T is the number of iterations until state evolution convergence, κ_T, K_T are constants depending on T, and $f(M) = \frac{M^{-\kappa_2 \delta^2}}{\delta \sqrt{\ln M}}$ is a quantity that tends to zero with growing M. For any power allocation, T increases as R approaches C. For example, $T \propto 1/\log(\mathcal{C}/R)$ for the exponential power allocation. We observe that the deviation probability bound on the RHS of (3.14) depends on the ratio $L/(\log M)^{2T-1}$.

In our experiments, T is generally on the order of a few tens. Therefore, keeping L constant, the probability of large deviations from the SE prediction $\mathcal{E}_{\text{sec}}^{\text{SE}}$ increases with M. This leads to the situation shown in Figure 3.12, which shows that the SE prediction $\mathcal{E}_{\text{sec}}^{\text{SE}}$ continues to decrease with M, but beyond a certain value of M, the observed average section error rate becomes progressively worse due to loss of concentration. This is caused by a small number of trials with a very large number of section errors, even as the majority of trials experience lower and lower error rates as M is increased. This effect can be clearly seen in Figure 3.13, which compares the histogram of section error rates over 200 trials for $M = 64$ and $M = 4096$. The distribution of errors is clearly different, but both cases have the same average section error rate due to the poorer concentration for $M = 4096$.

To summarise, given R, snr, and L, there is an optimal M that minimises the empirical section error rate. Beyond this value of M, the benefit from any further increase is outweighed by the loss of concentration. For a given R, values of M close to L are a good starting point for optimizing the empirical section error rate, but obtaining closed-form estimates of the optimal M for a given L is still an open question.

For fixed L, R, the optimal value of M increases with snr. This effect can be seen in the results of Figure 4.1, where there is an inversion in the order of best-performing M values as E_b/N_0 increases. This is because as snr increases, the number of iterations T for SE to converge decreases. A smaller T mitigates the effect of larger M in the large deviations bound of (3.14). In other words, a larger snr leads to better error rate concentration around the SE prediction, so larger values of M are permissible before the performance starts degrading.

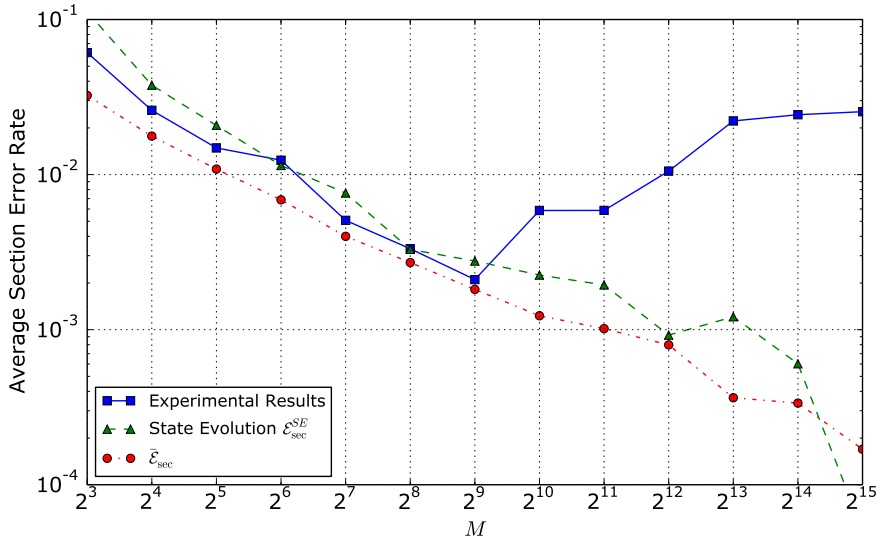


Figure 3.12: AMP error performance with increasing M, for $L = 1024$, $R = 1.5$, and $\frac{E_b}{N_0} = 5.7$ dB (2 dB from Shannon limit). See Section 3.7 for details of $\bar{\mathcal{E}}_{\text{sec}}$.

3.5.2 Effect of Power Allocation on Concentration

The non-asymptotic bounds on $x(\tau)$ in Lemma 3.1 indicate that at finite lengths, the minimum power required for a section ℓ to decode in an iteration may be slightly different than that indicated by the approximation in (3.7). Recall that the iterative power allocation algorithm in Section 3.4.2 was designed based on (3.7). We can compensate for the difference between the approximation and the actual value of $x(\tau)$ by running the iterative power allocation in Algorithm 3.3 using a modified rate R_{PA} which may be slightly different from the communication rate R. The choice of R_{PA} directly affects the error concentration. We now discuss the mechanism for this effect and give guidelines for choosing R_{PA} as a function of R.

If we run the power allocation algorithm with $R_{\text{PA}} > R$, from (3.7) we see that additional power is allocated to the initial blocks, at the cost of less power for the final blocks (where the allocation is flat). Consequently, it is less likely that one of the initial sections will decode in error, but more likely that some number of the later sections will instead. Figure 3.14 (bottom) shows the effect of choosing a large $R_{\text{PA}} = 1.06R$: out of a total of 1000 trials, there were no trials with more than 7 sections decoded in error (the number of sections $L = 1024$); however, relatively few trials (29%) have zero section errors.

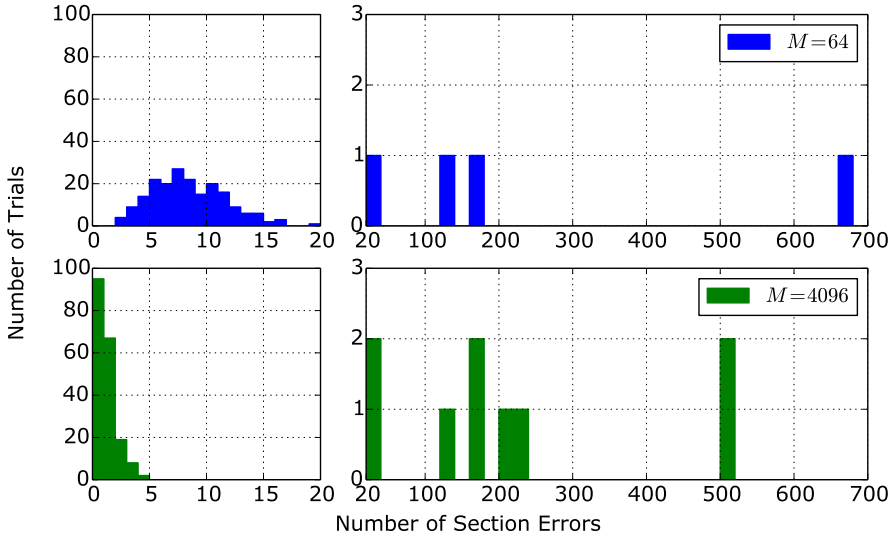


Figure 3.13: Histogram of AMP section errors over 200 trials, $M = 64$ (top) and $M = 4096$ (bottom), with $L = 1024$, $R = 1.5$, $\frac{E_b}{N_0} = 5.7$dB. The left panels highlight distribution of errors around low section error counts, while the right panels show the distribution around high-error-count events. As shown in Figure 3.12, both cases have an average section error rate of around 10^{-2}, but the lower M gives better concentration at a higher expected error rate, while the larger M has many more trials with zero errors, but also more trials with a large number of errors.

Conversely, choosing $R_{\text{PA}} < R$ allocates less power to the initial blocks, and increases the power in the final sections which have a flat allocation. This increases the likelihood of the initial section being decoded in error; in a trial when this happens, there will be a large number of section errors. However, if the initial sections are decoded correctly, the additional power in the final sections increases the probability of the trial being completely error-free. Thus choosing $R_{PA} < R$ makes completely error-free trials more likely, but also increases the likelihood of having trials with a large number of sections in error. In Figure 3.14 (top), the smaller $R_{\text{PA}} = 0.98R$ gives zero or one section errors in the majority (81%) of cases, but the remaining trials typically have a large number of sections in error.

To summarise, the larger the R_{PA}, the better the concentration of section error rates of individual trials around the overall average. However, increasing R_{PA} beyond a point just increases the average section error rate because of too little power being allocated to the final sections.

For different values of the communication rate R, we empirically determined an R_{PA} that

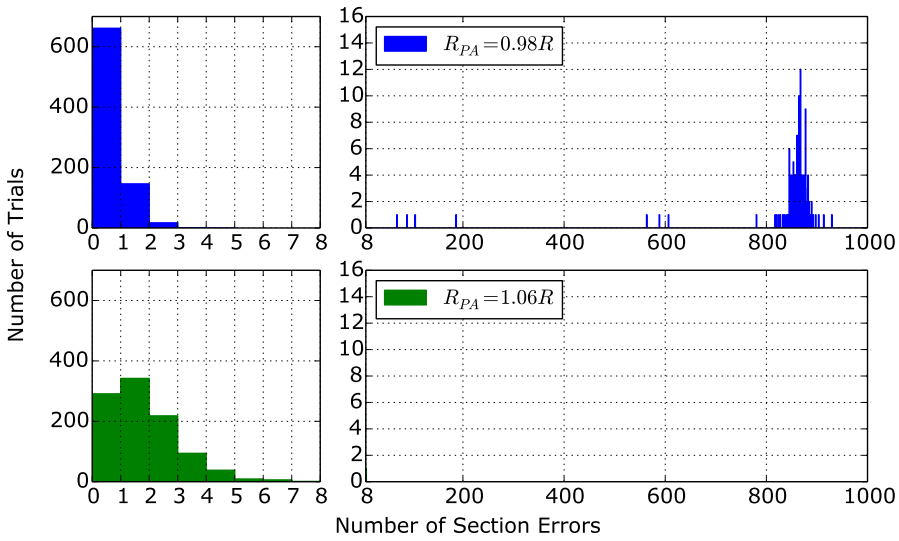


Figure 3.14: Histogram of AMP section errors over 1000 trials for $R_{\text{PA}} = 0.98R$ (top) and $R_{\text{PA}} = 1.06R$ (bottom). The SPARC parameters are $L = 1024$, $M = 512$, $R = 1.6$ and snr $= 15$. The left panels highlight distribution of trials with low section error counts (up to 8); the right panels indicate the distribution of infrequent but high-error-count trials. At lower R_{PA}, many more trials have no section errors, but those that do often have hundreds. At higher R_{PA}, at most 7 section errors were seen, but many fewer trials had zero section errors.

gives the lowest average section error rate, by starting at $R_{\text{PA}} = R$ and searching the neighborhood in steps of $0.02R$. Exceptionally, at low rates (for $R \leq 1$), the optimal R_{PA} is found to be 0, leading to a completely flat power allocation with $P_\ell = \frac{P}{L}$ for all ℓ. We note from (3.7) that for $1 \geq R > \frac{P}{2\tau_0^2 \ln 2}$, the large system limit theory does not predict that we can decode *any* of the L sections — this is because no section is above the threshold in the first iteration of decoding. However, in practice, we observe that some sections will decode initially (due to the correct column being aligned favorably with the noise vector), and this reduces the threshold enough to allow subsequent decoding to continue in most cases. For $R \leq 1$, when R_{PA} closer to R is used, the lower power in later sections hinders the finite length decoding performance.

We found that the value of $\frac{R_{PA}}{R}$ that minimises the average section error rate increases with R. In particular, the optimal $\frac{R_{PA}}{R}$ was 0 for $R \leq 1$; the optimal $\frac{R_{PA}}{R}$ for $R = 1.5$ was close to 1, and for $R = 2$, the optimal $\frac{R_{PA}}{R}$ was between 1.05 and 1.1. Though this provides a useful design guideline, a deeper theoretical analysis of the role of R_{PA} in optimizing the finite length performance is an open question.

Finally, a word of caution when empirically optimizing R_{PA} to minimise the average section error rate. Due to the loss of concentration as R_{PA} is decreased below R, care must be taken to run sufficient trials to ensure that a rare unseen trial with many section errors will not catastrophically impact the overall average section error rate. For example, in one scenario with $L = 1024, M = 512, \mathsf{snr} = 15, R = 1.4, R_{\text{PA}} = 1.316$, we observed 192 trials with errors out of 407756 trials, but only 4 of these trials had more than one error, with between 400 to 600 section errors in those 4 cases. The average section error rate was 5.6×10^{-6}. With fewer trials, it is possible that no trials with a large number of section errors would be observed, leading to an estimated error rate an order of magnitude better, at around 4.6×10^{-7}.

3.6 Online Computation of τ_t^2 and Early Termination

Recall that the update step (2.10) of the AMP decoder requires the SE coefficients τ_t^2, for $t \in [T]$. In the standard implementation [1], these coefficients are computed in advance using the SE equations (2.7)–(2.9). The total number of iterations T is also determined in advance by computing the number of iterations required by the SE to converge to its fixed point (to within a specified tolerance). This technique produced effective results, but advance computation is slow as each of the L expectations in (2.9) needs to be computed numerically via Monte-Carlo simulation, for each t. A faster approach is to compute the τ_t^2 coefficients using the asymptotic expression for $x(\tau)$ given in (3.7). This gives error performance nearly identical to the earlier approach with significant time savings, but still requires advance computation. Both these methods are referred to as "offline" as the τ_t^2 values are computed a priori.

A simple way to estimate τ_t^2 online during the decoding process is as follows. In each step t, after producing z^t as in (2.5), we estimate

$$\widehat{\tau}_t^2 = \frac{\|z^t\|^2}{n} = \frac{1}{n}\sum_{i=1}^{n} z_i^2. \tag{3.15}$$

The justification for this estimate comes from the analysis of the AMP decoder in [1,24], which shows that for large n, $\widehat{\tau}_t^2$ is close to τ_t^2 in (2.7) with high probability. In particular, [24] provides a concentration inequality for $\widehat{\tau}_t^2$ similar to (3.14). A similar online estimate has been used previously in various AMP and GAMP algorithms [31, 39, 46, 47]. The online estimator $\widehat{\tau}_t^2$ provides multiple advantages beyond avoiding advance computation.

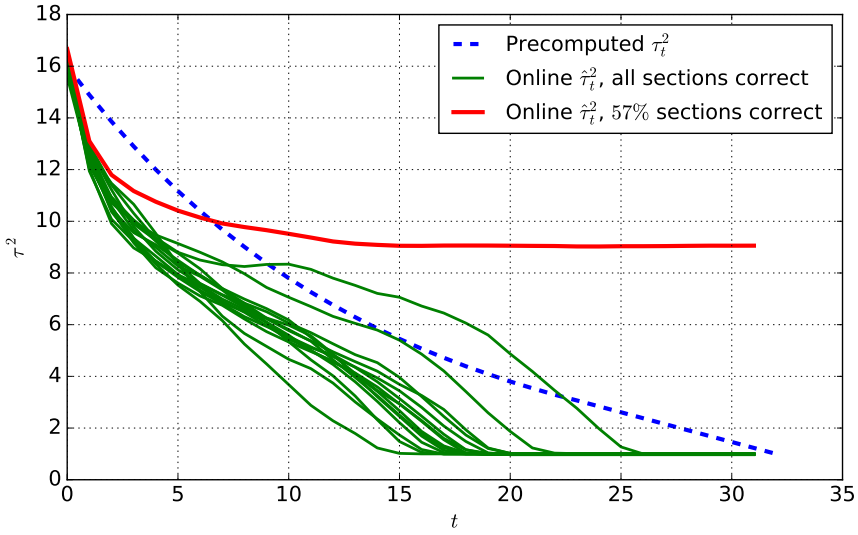


Figure 3.15: Comparison between offline and online trajectories of the effective noise variance, at $L = 1024, M = 512, P = 15, \sigma^2 = 1, R = 1.6$. The dashed line represents the pre-computed SE trajectory of τ_t^2. The plot shows 15 successful runs, and one uncommon run with many section errors. The true value of $\mathrm{Var}[s^t - \beta]$ during decoding tracks $\widehat{\tau}_t^2$ too precisely to distinguish on this plot.

First, the error performance with the online estimator was observed to consistently be the same or better than the offline methods. Another attractive feature of online estimation is that $\widehat{\tau}_t^2$ can be used to track decoding progress. Recall from the discussion at the beginning of Section 3.4 that in each step, we have

$$s^t := \beta^t + A^* z^t \approx \beta + \tau_t Z, \tag{3.16}$$

where Z is a standard normal random vector independent of β. Starting from $\tau_0^2 = \sigma^2 + P$, a judicious choice of power allocation ensures that the SE parameter τ_t^2 decreases with t, until it converges at $\tau_T^2 = \sigma^2$ in a finite number of iterations T.

However, at finite lengths there are deviations from this trajectory of τ_t^2 predicted by SE, i.e., the variance of the effective noise vector $(s^t - \beta)$ may deviate from τ_t^2. The online estimator $\widehat{\tau}_t^2$ is found to track $\mathrm{Var}(s^t - \beta) = \|s^t - \beta\|^2/n$ very accurately. This effect can be seen in Figure 3.15, where 16 independent decoder runs are plotted and compared with the SE trajectory for τ_t^2 (dashed line). For the 15 successful runs, the empirical variance $\mathrm{Var}(s^t - \beta)$ approaches $\sigma^2 = 1$ along different trajectories depending on how the decoding is progressing.

In the unsuccessful run, $\text{Var}(s^t - \beta)$ converges to a value much larger than σ^2.

In all the runs, $\widehat{\tau}_t^2$ is indistinguishable from $\text{Var}(s^t - \beta)$. This indicates that we can use the final value $\widehat{\tau}_T^2$ to accurately estimate the power of the undecoded sections — and thus the number of sections decoded correctly — at runtime. Indeed, $(\widehat{\tau}_T^2 - \sigma^2)$ is an accurate estimate of the total power in the incorrectly decoded sections. This, combined with the fact that the power allocation is non-increasing, allows the decoder to estimate the number of incorrectly decoded sections.

Furthermore, we can use the change in $\widehat{\tau}_t^2$ between iterations to terminate the decoder early. If the value $\widehat{\tau}_t^2$ has not changed between successive iterations, or the change is within some small threshold, then the decoder has stalled and no further iterations are worthwhile. Empirically we find that a stopping criterion with a small threshold (e.g., stop when $|\widehat{\tau}_t^2 - \widehat{\tau}_{t-1}^2| < P_L$) leads to no additional errors compared to running the decoder for the full iteration count, while giving a significant speedup in most trials. Allowing a larger threshold for the stopping criterion gives even better running time improvements. This early termination criterion based on $\widehat{\tau}_t^2$ gives us flexibility in choosing the number of blocks B in the iterative power allocation algorithm of Section 3.4.2. This is because the number of AMP iterations is no longer tied to B, hence B can be chosen as large as desired.

To summarise, the online estimator $\widehat{\tau}_t^2$ provides an estimate of the noise variance in each AMP iteration that accurately reflects how the decoding is progressing in that trial. It thereby enables the decoder to effectively adapt to deviations from the τ_t^2 values predicted by SE. This explains the improved performance compared to the offline methods of computing τ_t^2. More importantly, it provides an early termination criterion for the AMP decoder as well as a way to track decoding progress and predict the number of section errors at runtime.

3.7 Predicting $\mathcal{E}_{\text{sec}}$, $\mathcal{E}_{\text{ber}}$, and $\mathcal{E}_{\text{cw}}$

For a given power allocation $\{P_\ell\}_{\ell \in [L]}$ and reasonably large SPARC parameters (n, M, L), it is desirable to have a quick way to estimate the section error rate and codeword error rate, without resorting to simulations. Without loss of generality, we assume that the power allocation is asymptotically good, i.e., the large system limit SE parameters (computed using (3.7)) predict reliable decoding, i.e., the SE converges to $x_T = 1$ and $\tau_T^2 = \sigma^2$ in the large system limit. The goal is to estimate the finite length section error rate $\mathcal{E}_{\text{sec}}$.

One way to estimate $\mathcal{E}_{\text{sec}}$ is via the state evolution prediction (3.13), using $\tau_T = \sigma$. How-

ever, $\mathcal{E}_{\text{sec}}^{\text{SE}}$ in computing (3.13) requires computing L expectations, each involving a function of M independent standard normal random variables. The following result provides estimates of $\mathcal{E}_{\text{sec}}$ and $\mathcal{E}_{\text{cw}}$ that are as accurate as the SE-based estimates, but much simpler to compute.

Proposition 3.1. *Let the power allocation $\{P_\ell\}$ be such that the state evolution iteration using the asymptotic approximation* (3.7) *converges to $\tau_T^2 = \sigma^2$. Then, under the idealised assumption that $\beta^T + A^* z^T = \beta + \tau_T Z$ (where Z is a standard normal random vector independent of β), we have the following. The probability of a section (chosen uniformly at random) being incorrectly decoded is*

$$\bar{\mathcal{E}}_{\text{sec}} = 1 - \frac{1}{L}\sum_{\ell=1}^{L} \mathbb{E}_U \left[\Phi\left(\frac{\sqrt{nP_\ell}}{\sigma} + U \right) \right]^{M-1}. \tag{3.17}$$

The probability of the codeword being incorrectly decoded is

$$\bar{\mathcal{E}}_{cw} = 1 - \prod_{\ell=1}^{L} \mathbb{E}_U \left[\Phi\left(\frac{\sqrt{nP_\ell}}{\sigma} + U \right) \right]^{M-1}. \tag{3.18}$$

In both expressions above, U is a standard normal random variable, and $\Phi(.)$ is the standard normal cumulative distribution function.

Proof. As $\tau_T^2 = \sigma^2$, the effective observation in the final iteration has the representation $\beta + \sigma Z$. The denoising function η^T generates a final estimate based on this effective observation, and the index of the largest entry in each section is chosen to form the decoded message vector $\widehat{\beta}$. Consider the decoding of section ℓ of β. Without loss of generality, we can assume that the first entry of the section is the non-zero one. Using the notation $\beta_{\ell,j}$ to denote the jth entry of the section β_ℓ, we therefore have $\beta_{\ell,1} = \sqrt{nP_\ell}$, and $\beta_{\ell,j} = 0$ for $2 \le j \le M$. As the effective observation for section ℓ has the representation $(\beta^T + A^* z^T)_\ell = \beta_\ell + \sigma Z_\ell$, the section will be incorrectly decoded if and only if the following event occurs:

$$\left\{ \sqrt{nP_\ell} + \sigma Z_{\ell,1} \le \sigma Z_{\ell,2} \right\} \cup \ldots \cup \left\{ \sqrt{nP_\ell} + \sigma Z_{\ell,1} \le \sigma Z_{\ell,M} \right\}.$$

Therefore, the probability that the ℓth section is decoded in error can be computed as

$$\begin{aligned} P_{\text{err},\ell} &= 1 - \mathbb{P}\left(\sqrt{nP_\ell} + \sigma Z_{\ell,1} > \sigma Z_{\ell,j},\ 2 \le j \le M\right) \\ &= 1 - \int_{\mathbb{R}} \prod_{j=2}^{M} \mathbb{P}\left(Z_{\ell,j} < \frac{\sqrt{nP_\ell}}{\sigma} + u \,\middle|\, Z_{\ell,1} = u\right) \phi(u) du \\ &= 1 - \mathbb{E}_U\left[\Phi\left(\frac{\sqrt{nP_\ell}}{\sigma} + U\right)\right]^{M-1}, \end{aligned} \tag{3.19}$$

where ϕ and Φ denote the density and the cumulative distribution function of the standard normal distribution, respectively. In the second line of (3.19), we condition on $Z_{\ell,1}$ and then use the fact that $Z_{\ell,1}, \ldots, Z_{\ell,M}$ are i.i.d. $\sim \mathcal{N}(0,1)$.

The probability of a section chosen uniformly at random being incorrectly decoded is

$$\frac{1}{L}\sum_{\ell=1}^{L} P_{\text{err},\ell}.$$

The probability of codeword error is one minus the probability that no section is in error, which is given by

$$1 - \prod_{\ell=1}^{L}(1 - P_{\text{err},\ell}).$$

Substituting for $P_{\text{err},\ell}$ from (3.19) yields the expressions in (3.17) and (3.18). □

The section error rate and codeword error rate can be estimated using the idealised expressions in (3.17) and (3.18). This still requires computing L expectations, but each expectation is now a function of a single Gaussian random variable, rather than the M independent ones in the SE estimate. Thus we reduce the complexity by a factor of M over the SE approach; evaluations of $\bar{\mathcal{E}}_{\text{sec}}$ and $\bar{\mathcal{E}}_{\text{cw}}$ typically complete within a second.

Figure 3.12 shows $\bar{\mathcal{E}}_{\text{sec}}$ alongside the SE estimate $\mathcal{E}_{\text{sec}}^{\text{SE}}$ for $L = 1024$, and various values of M. We see that both these estimates match the simulation results closely up to a certain value of M. Beyond this point, the simulation results diverge from theoretical estimates due to lack of concentration in section error rates across trials, as described in Sec. 3.5.1. Figure 3.16 compares the idealised codeword error probability in (3.18) with that obtained from simulations. Here, there is a good match between the estimate and the simulation results as the concentration of section error rates across trials plays no role — any trial with one or more section errors

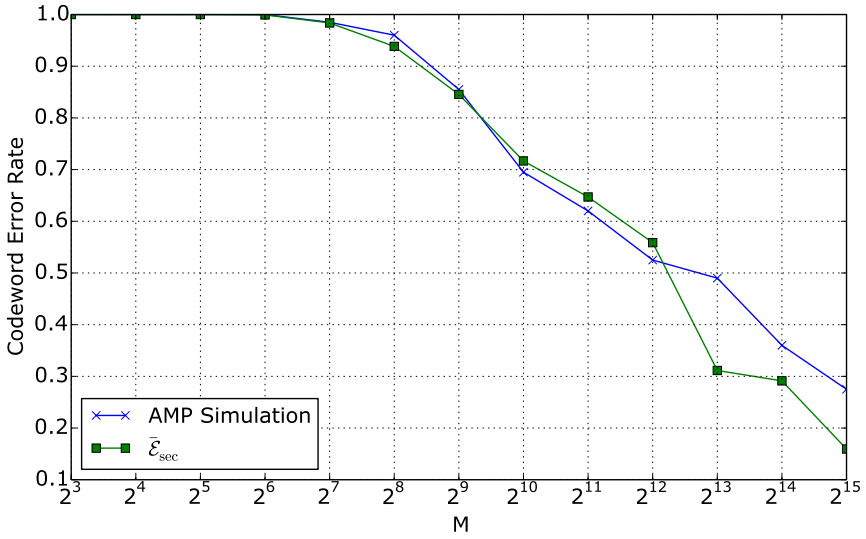


Figure 3.16: Comparison of codeword error rate between simulation results and P_{err}-based analysis, for $\mathcal{E}_{\text{cw}}$ with varying M. $L = 1024$, $R = 1.5$, $E_b/N_0 = 5.7$dB. Results are well matched even when concentration is poor.

corresponds to one codeword error.

It is further possible to find an upper bound for $\bar{\mathcal{E}}_{\text{sec}}$, eliminating computation of the expectation entirely. For brevity, let $a = \frac{\sqrt{nP_\ell}}{\sigma}$. To find an upper bound for $\bar{\mathcal{E}}_{\text{sec}}$ we first find a lower bound on the inner expectation $\mathbb{E}_U\left[\Phi(a+U)\right]$. Recall that $U \sim \mathcal{N}(0,1)$ and using the Chernoff inequality

$$1 - \Phi(x) = Q(x) \leq e^{\frac{-x^2}{2}}$$

for $x > 0$, we obtain

$$\Phi(x) \geq 1 - e^{\frac{-x^2}{2}}. \tag{3.20}$$

Split the expectation into two parts conditioned on the event $U > -a$ to ensure the argument to $\Phi(.)$ remains positive. We can bound this quantity by discarding the second term; since a is large (of order $\log M$), the event $U \leq -a$ is unlikely and so the second term does not significantly affect the bound. Use the inequality of (3.20) twice to obtain

$$\mathbb{E}_U\left[\Phi(a+U)\right] = \mathbb{E}\left[\Phi(a+U) \mid U > -a\right] P(U > -a) \tag{3.21}$$

$$+ \mathbb{E}\left[\Phi(a+U) \mid U \leq -a\right] P(U \leq -a) \tag{3.22}$$

$$\geq \mathbb{E}\left[1 - e^{-(a+U)^2/2} \mid U > -a\right] (1 - \Phi(-a)) \tag{3.23}$$

$$= \left(1 - \mathbb{E}\left[e^{-\frac{a^2}{2} - aU - \frac{U^2}{2}} \mid U > -a\right]\right) \Phi(a) \tag{3.24}$$

$$\geq \left(1 - \mathbb{E}\left[e^{-\frac{a^2}{2} - aU - \frac{U^2}{2}} \mid U > -a\right]\right) \left(1 - e^{-\frac{a^2}{2}}\right) \tag{3.25}$$

Computing the expectation by integration,

$$\mathbb{E}\left[e^{-\frac{a^2}{2} - aU - \frac{U^2}{2}} \mid U > -a\right] = \int_{-a}^{\infty} e^{-\frac{a^2}{2} - au - \frac{u^2}{2}} \frac{1}{P(u > -a)} \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du \tag{3.26}$$

$$= \frac{1}{\Phi(a)\sqrt{2\pi}} e^{-\frac{a^2}{2}} \int_{-a}^{\infty} e^{\frac{a^2}{4} - (u + \frac{a}{2})^2} du \tag{3.27}$$

$$= \frac{1}{\Phi(a)\sqrt{2\pi}} e^{-\frac{a^2}{4}} \int_{-a}^{\infty} e^{-(u + \frac{a}{2})^2} du \tag{3.28}$$

Use the substitution $v(u) = u + \frac{a}{2}, dv = du$,

$$\mathbb{E}\left[e^{-\frac{a^2}{2} - aU - \frac{U^2}{2}} \mid U > -a\right] = \frac{1}{\Phi(a)\sqrt{2\pi}} e^{-\frac{a^2}{4}} \int_{-a/2}^{\infty} e^{-v^2} dv \tag{3.29}$$

$$= \frac{1}{\Phi(a)\sqrt{2\pi}} e^{-\frac{a^2}{4}} \left(\int_{0}^{\infty} e^{-v^2} dv + \int_{-\frac{a}{2}}^{0} e^{-v^2} dv\right) \tag{3.30}$$

$$= \frac{1}{\Phi(a)\sqrt{2\pi}} e^{-\frac{a^2}{4}} \left(\frac{\sqrt{\pi}}{2} \operatorname{erf}(\infty) + \frac{\sqrt{\pi}}{2} \operatorname{erf}\left(\frac{a}{2}\right)\right) \tag{3.31}$$

$$\geq \frac{1}{1 - e^{\frac{-a^2}{2}}} \frac{1}{2\sqrt{2}} e^{-\frac{a^2}{4}}, \tag{3.32}$$

noting in the final step that $a \geq 0$ and $\operatorname{erf}(x) \geq 0$ when $x \geq 0$, and $\operatorname{erf}(\infty) = 1$ by definition.

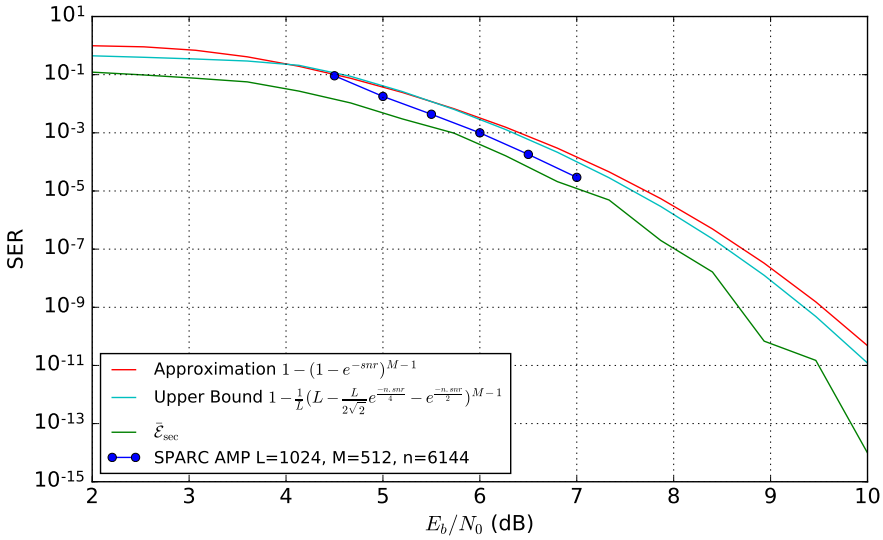


Figure 3.17: Comparison of predicted error rate $\bar{\mathcal{E}}_{\text{sec}}$ with upper bound, approximation based on snr, and SPARC AMP simulation results for $L = 1024$, $M = 512$, $n = 6144$, $R = 1.5$, $\sigma^2 = 1.0$. The SPARC AMP data is repeated in Figure 4.2.

Use this bound in (3.25) and expand a to obtain

$$\mathbb{E}_U\left[\Phi(a+U)\right] \geq \left(1 - \frac{1}{1-e^{\frac{-a^2}{2}}}\frac{1}{2\sqrt{2}}e^{-\frac{a^2}{4}}\right)\left(1-e^{-\frac{a^2}{2}}\right) \tag{3.33}$$

$$= 1 - \frac{1}{2\sqrt{2}}e^{\frac{-nP_\ell}{4\sigma^2}} - e^{\frac{-nP_\ell}{2\sigma^2}}. \tag{3.34}$$

Finally use (3.34) in (3.17) to obtain

$$\bar{\mathcal{E}}_{\text{sec}} \leq 1 - \frac{1}{L}\sum_{\ell=1}^{L}\left(1 - \frac{1}{2\sqrt{2}}e^{\frac{-nP_\ell}{4\sigma^2}} - e^{\frac{-nP_\ell}{2\sigma^2}}\right)^{M-1}. \tag{3.35}$$

To investigate the effect of snr on error rate, we can approximate (3.35). Note that for a given power allocation algorithm, $P_\ell \propto P$, and so to a rough approximation,

$$\bar{\mathcal{E}}_{\text{sec}} \approx 1 - (1 - e^{-\text{snr}})^{M-1}. \tag{3.36}$$

Figure 3.17 compares the predicted error rates, the upper bound derived above, and the

approximation based on snr to error rates obtained using the AMP decoder. There is a good correspondence in the region where AMP results are available, and we can further see expected behaviour in higher snr regimes which are impractical to simulate due to extremely low error rates. The apparently straight-line (on a log-log plot) behaviour of the SPARC AMP error rates is instead revealed to be a curve which falls off more quickly at higher snr.

Chapter 4

Outer Codes for SPARCs

In Chapter 3 we considered various techniques to improve the section error rate $\mathcal{E}_{\text{sec}}$ of the AMP decoder. While substantial improvements in $\mathcal{E}_{\text{sec}}$ were possible, the rate at which $\mathcal{E}_{\text{sec}}$ decreases as the rate backs off from capacity does not exhibit the steep "waterfall" effect common to other channel codes. In this chapter we first compare the performance of the AMP decoder to a coded modulation scheme with a state of the art LDPC code. The coded modulation scheme obtains a steep reduction in error rate at some threshold rate away from capacity, while the SPARC does not. We then describe how LDPC outer codes can be used in conjunction with the AMP decoder to obtain such a reduction. Importantly, we are able to use the soft information present in the AMP estimate β^T prior to conversion to $\hat{\beta}$ to provide a better input to the LDPC decoder. Additionally we can use our knowledge of the power allocation to only use the LDPC outer code over sections that are likely to suffer errors, reducing the rate overhead associated with the outer code.

To combine SPARCs with an outer code, user data is first encoded with the outer code, forming a new codeword that is typically only a little longer than the original user data. This new codeword is then encoded using the inner code (in our case, the SPARC), transmitted, received, and decoded, all as usual. The decoder output is now an estimated codeword for the outer code. We then run a second decoder, for the outer code, to attempt to correct any remaining errors. If the inner code can be designed to reliably leave only a very small number of sections in error, then a high rate outer code can reliably correct these errors, leading to an excellent codeword error rate $\mathcal{E}_{\text{cw}}$ in addition to the improved $\mathcal{E}_{\text{sec}}$.

4.1 Comparison with Coded Modulation

LDPC codes may be used in a bit-interleaved coded modulation (BICM) system, with a modulation technique such as binary phase-shift keying (BPSK) or quadrature amplitude modulation (QAM). In such a setup, their performance can be directly compared to SPARCs with AMP decoding, operating at the same rate and over the same channel. This is helpful both to evaluate the performance of the AMP decoder relative to a state of the art BICM system, and to inform the design of subsequent outer codes for the SPARC.

In this section we compare the performance of AMP-decoded SPARCs against coded modulation with LDPC codes. Specifically, we compare with two instances of coded modulation with LDPC codes from the WiMax standard IEEE 802.16e: 1) A 16-QAM constellation with a rate $\frac{1}{2}$ LDPC code for an overall rate $R = 1$ bit/channel use/real dimension, and 2) A 64-QAM constellation with a rate $\frac{1}{2}$ LDPC code for an overall rate $R = 1.5$ bits/channel use/real dimension. (The spectral efficiency is $2R$ bits/s/Hz.) The coded modulation results, shown in dashed lines in Figures 4.1 and 4.2, are obtained using the CML toolkit [48] with LDPC code lengths $n = 576$ and $n = 2304$.

Each figure compares the BER of the coded modulation schemes with various SPARCs of the same rate, including a SPARC with a matching code length of $n = 2304$. We plot bit error rate (BER) against E_b/N_0 as it is a more common figure of merit for BICM systems. BER is measured directly for the SPARCs by counting the number of bits of output which are different to the input bits. Using $P = E_b R$ and $\sigma^2 = \frac{N_0}{2}$, the signal-to-noise ratio of the SPARC can be expressed as $\frac{P}{\sigma^2} = \frac{2RE_b}{N_0}$. The SPARCs are implemented using Hadamard-based design matrices, power allocation designed using the iterative algorithm in Sec. 3.4.2 with $B = L$, and online $\widehat{\tau}_t^2$ parameters with the early termination criterion (Sec. 3.6).

Figure 4.1 shows that for $L = 1024$, the best value of M among those considered increases from $M = 2^9$ at lower snr values to $M = 2^{13}$ at higher snr values. This is due to the effect discussed in Section 3.5.1, where larger snr values can support larger values of M, before performance starts degrading due to loss of concentration.

At both $R = 1$ and $R = 1.5$, the SPARCs outperform the LDPC coded modulation at E_b/N_0 values close to the Shannon limit, but the error rate does not drop off as quickly at higher values of E_b/N_0. In the next section, we consider using outer codes to improve the SPARC error rate at higher E_b/N_0.

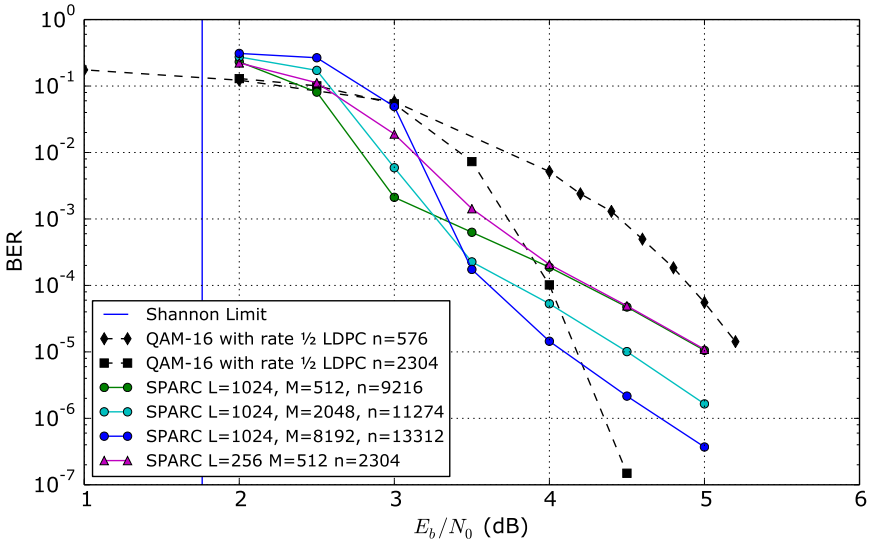


Figure 4.1: Comparison with LDPC coded modulation at $R = 1$

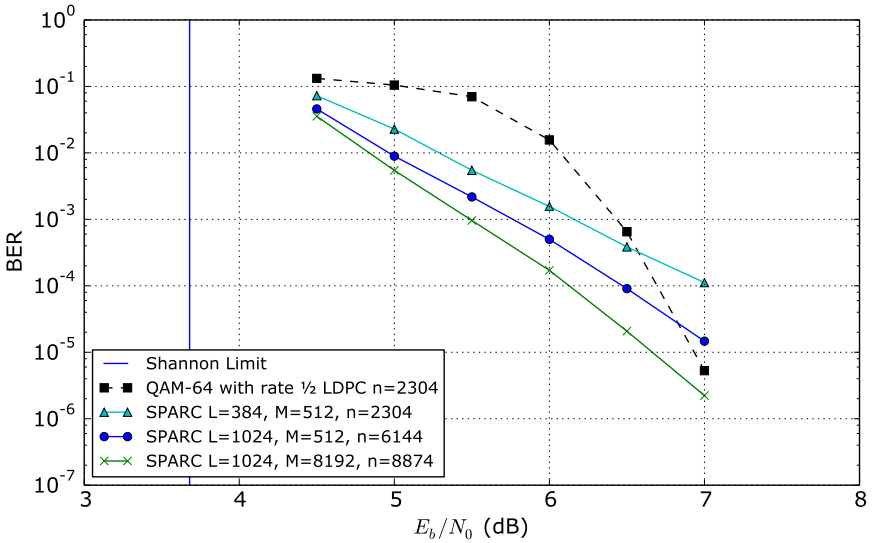


Figure 4.2: Comparison with LDPC coded modulation at $R = 1.5$

4.2 AMP with Partial Outer Codes

Figures 4.1 and 4.2 show that for block lengths of the order of a few thousands, AMP-decoded SPARCs do not exhibit a steep waterfall in section error rate. Even at high E_b/N_0 values, it is still common to observe a small number of section errors. If these could be corrected, we could hope to obtain a sharp waterfall behaviour similar to the LDPC codes.

In the simulations of the AMP decoder described in Section 3.5, when M and R_{PA} are chosen such that the average error rates are well-concentrated around the state evolution prediction, the number of section errors observed is similar across trials. Furthermore, we observe that the majority of sections decoded incorrectly are those in the flat region of the power allocation, i.e., those with the lowest allocated power. This suggests we could use a high-rate outer code to protect just these sections, sacrificing some rate, but less than if we naïvely protected all sections. We call the sections covered by the outer code *protected* sections, and conversely the earlier sections which are not covered by the outer code are *unprotected*. In [17], it was shown that a Reed-Solomon outer code (that covered all the sections) could be used to obtain a bound on the probability of codeword error from a bound on the probability of excess section error rate.

Encoding with an outer code (e.g., LDPC or Reed-Solomon code) is straightforward: just replace the message bits corresponding to the protected sections with coded bits generated using the usual encoder for the chosen outer code. To decode, we would like to obtain bit-wise posterior probabilities for each codeword bit of the outer code, and use them as inputs to a soft-information decoder, such as a sum-product or min-sum decoder for LDPC codes. The output of the AMP decoding algorithm permits this: it yields β^T, which contains weighted *column-wise* posterior probabilities; we can directly transform these into *bit-wise* posterior probabilities. See Algorithm 4.1 for details.

Moreover, in addition to correcting AMP decoding errors in the protected sections, successfully decoding the outer code also provides a way to correct remaining errors in the unprotected sections of the SPARC codeword. After decoding the outer code we can subtract the contribution of the protected sections from the channel output sequence y, and re-run the AMP decoder on just the unprotected sections. The key point is that subtracting the contribution of the later (protected) sections eliminates the interference due to these sections; then running the AMP decoder on the unprotected sections is akin to operating at a much lower rate, and so it is able to correctly decode unprotected sections which were not previously de-

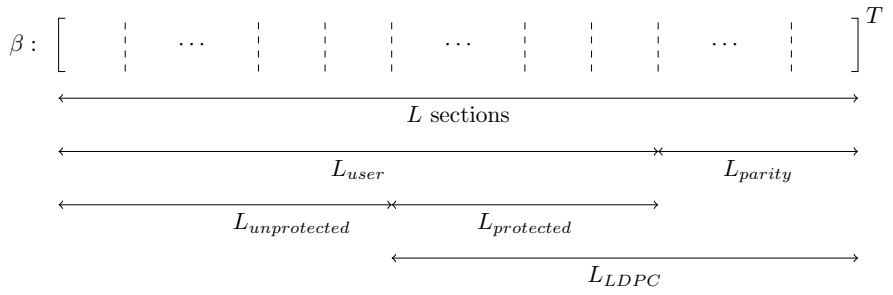


Figure 4.3: Division of the L sections of β for an outer LDPC code

coded correctly.

Thus the decoding procedure has three stages: i) first round of AMP decoding, ii) decoding the outer code using soft outputs from the AMP, and iii) subtracting the contribution of the sections protected by the outer code, and running the AMP decoder again for the unprotected sections. We find that the final stage, i.e., running the AMP decoder again after the outer code recovers errors in the protected sections of the SPARC, provides a significant advantage over a standard application of an outer code, i.e., decoding the final codeword after the second stage.

We describe this combination of SPARCs with outer codes below, using an LDPC outer code. The resulting error rate curves exhibit sharp waterfalls in final error rates, even when the LDPC code only covers a minority of the SPARC sections.

We use a binary LDPC outer code with rate R_{LDPC}, block length n_{LDPC} and code dimension k_{LDPC}, so that $k_{LDPC}/n_{LDPC} = R_{LDPC}$. For clarity of exposition we assume that both n_{LDPC} and k_{LDPC} are multiples of $\log M$ (and consequently that M is a power of two). As each section of the SPARC corresponds to $\log M$ bits, if $\log M$ is an integer, then n_{LDPC} and k_{LDPC} bits represent an integer number of SPARC sections, denoted by

$$L_{LDPC} = \frac{n_{LDPC}}{\log M} \quad \text{and} \quad L_{protected} = \frac{k_{LDPC}}{\log M},$$

respectively. The assumption that k_{LDPC} and n_{LDPC} are multiples of $\log M$ is not necessary in practice; the general case is discussed at the end of the next subsection.

We partition the L sections of the SPARC codeword as shown in Fig 4.3. There are L_{user} sections corresponding to the user (information) bits; these sections are divided into *unprotected* and *protected* sections, with only the latter being covered by the outer LDPC code. The parity bits of the LDPC codeword index the last L_{parity} sections of the SPARC. For conve-

nience, the *protected* sections and the *parity* sections together are referred to as the *LDPC* sections.

For a numerical example, consider the case where $L = 1024$, $M = 256$. There are $\log M = 8$ bits per SPARC section. For a $(5120, 4096)$ LDPC code ($R_{LDPC} = 4/5$) we obtain the following relationships between the number of the sections of each kind:

$$\begin{aligned}
L_{parity} &= \frac{n_{LDPC} - k_{LDPC}}{\log M} = \frac{(5120 - 4096)}{8} = 128, \\
L_{user} &= L - L_{parity} = 1024 - 128 = 896, \\
L_{protected} &= \frac{k_{LDPC}}{\log M} = \frac{4096}{8} = 512, \\
L_{LDPC} &= L_{protected} + L_{parity} = 512 + 128 = 640, \\
L_{unprotected} &= L_{user} - L_{protected} = L - L_{LDPC} = 384.
\end{aligned}$$

There are $L_{user} \log M = 7168$ user bits, of which the final $k_{LDPC} = 4096$ are encoded to a systematic $n_{LDPC} = 5120$-bit LDPC codeword. The resulting $L \log M = 8192$ bits (including both the user bits and the LDPC parity bits) are encoded to a SPARC codeword using the SPARC encoder and power allocation described in previous sections.

We continue to use R to denote the overall user rate, and n to denote the SPARC code length so that $nR = L_{user} \log M$. The underlying SPARC rate (including the overhead due to the outer code) is denoted by R_{SPARC}. We note that $nR_{SPARC} = L \log M$, hence $R_{SPARC} > R$. For example, with $R = 1$ and L, M and the outer code parameters as chosen above, $n = L_{user}(\log M)/R = 7168$, so $R_{SPARC} = 1.143$.

Algorithm 4.1 Weighted position posteriors β_ℓ to bit posteriors $p_0, \ldots, p_{\log M - 1}$ for section $\ell \in [L]$

Require: $\beta_\ell = [\beta_{\ell,1}, \ldots, \beta_{\ell,M}]$, for M a power of 2
  Initialise bit posteriors $p_0, \ldots, p_{\log M - 1} \leftarrow 0$
  Initialise normalization constant $c \leftarrow \sum_{i=1}^{M} \beta_{\ell,i}$
  for $\log i = 0, 1, \ldots, \log M - 1$ **do**
    $b \leftarrow \log M - \log i - 1$
    $k \leftarrow i$
    while $k < M$ **do**
      for $j = k+1, k+2, \ldots, k+i$ **do**
        $p_b \leftarrow p_b + \beta_{\ell,j}/c$
      end for
      $k \leftarrow k + 2i$
    end while
  end for
  return $p_0, \ldots, p_{\log M - 1}$

4.2.1 Decoding SPARCs with LDPC outer codes

At the receiver, we decode as follows:

1. Run the AMP decoder to obtain β^T. Recall that entry j within section ℓ of β^T is proportional to the posterior probability of the column j being the transmitted one for section ℓ. Thus the AMP decoder gives section-wise posterior probabilities for each section $\ell \in [L]$.

2. Convert the section-wise posterior probabilities to bit-wise posterior probabilities using Algorithm 4.1, for each of the L_{LDPC} sections. This requires $O(L_{LDPC} M \log M)$ time complexity, of the same order as one iteration of AMP.

3. Run the LDPC decoder using the bit-wise posterior probabilities obtained in Step 2 as inputs.

4. If the LDPC decoder fails to produce a valid LDPC codeword, terminate decoding here, using β^T to produce $\hat{\beta}$ by selecting the maximum value in each section (as per usual AMP decoding).

5. If the LDPC decoder succeeds in finding a valid codeword, we use it to re-run AMP

decoding on the unprotected sections. For this, first convert the LDPC codeword bits to a partial $\hat{\beta}_{LDPC}$ as follows, using a method similar to the original SPARC encoding:

(a) Set the first $L_{unprotected}M$ entries of $\hat{\beta}_{LDPC}$ to zero,

(b) The remaining L_{LDPC} sections (with M entries per section) of $\hat{\beta}_{LDPC}$ will have exactly one-non zero entry per section, with the LDPC codeword determining the location of the non-zero in each section. Noting that $n_{LDPC} = L_{LDPC} \log M$, we consider the LDPC codeword as a concatenation of L_{LDPC} blocks of $\log M$ bits each, so that each block of bits indexes the location of the non-zero entry in one section of $\hat{\beta}_{LDPC}$. The value of the non-zero in section ℓ is set to $\sqrt{nP_\ell}$, according to the power allocation.

Now subtract the codeword corresponding to $\hat{\beta}_{LDPC}$ from the original channel output y, to obtain $y' = y - A\hat{\beta}_{LDPC}$.

6. Run the AMP decoder again, with input y', and operating only over the first $L_{unprotected}$ sections. As this operation is effectively at a much lower rate than the first decoder (since the interference contribution from all the protected sections is removed), it is more likely that the unprotected bits are decoded correctly than in the first AMP decoder.

   We note that instead of generating y', one could run the AMP decoder directly on y, but enforcing that in each AMP iteration, each of the L_{LDPC} sections has all its non-zero mass on the entry determined by $\hat{\beta}_{LDPC}$, i.e., consistent with Step 5.b).

7. Finish decoding, using the output of the final AMP decoder to find the first $L_{unprotected}M$ elements of $\hat{\beta}$, and using $\hat{\beta}_{LDPC}$ for the remaining $L_{LDPC}M$ elements.

In the case where n_{LDPC} and k_{LDPC} are not multiples of $\log M$, the values $L_{LDPC} = n_{LDPC}/\log M$ and $L_{protected} = k_{LDPC}/\log M$ will not be integers. Therefore one section at the boundary of $L_{unprotected}$ and $L_{protected}$ will consist of some unprotected bits and some protected bits. Encoding is not affected in this situation, as the LDPC encoding happens prior to SPARC codeword encoding. When decoding, conversion to bit-wise posterior probabilities is performed for all sections containing LDPC bits (including the intermediate section at the boundary) and only the n_{LDPC} bit posteriors corresponding to the LDPC codeword are given to the LDPC decoder. When forming $\hat{\beta}_{LDPC}$, the simplest option is to treat the intermediate section as though it were unprotected and set it to zero. It is also possible to compute column

posterior probabilities which correspond to the fixed LDPC bits and probabilities arising from y, though doing so is not detailed here.

4.2.2 Simulation Results

The combined AMP and outer LDPC setup described above was simulated using the (5120, 4096) LDPC code ($R_{LDPC} = 4/5$) specified in [49] with a min-sum decoder. Bit error rates were measured only over the user bits, ignoring any bit errors in the LDPC parity bits.

Figure 4.4 plots results at overall rate $R = \frac{4}{5}$, where the underlying LDPC code (modulated with BPSK) can be compared to the SPARC with LDPC outer code, and to a plain SPARC with rate $\frac{4}{5}$. In this case $R_{PA} = 0$, giving a flat power allocation. Figure 4.5 plots results at overall rate $R = 1.5$, where we can compare to the QAM-64 WiMAX LDPC code, and to the plain SPARC with rate 1.5 of Figure 4.2.

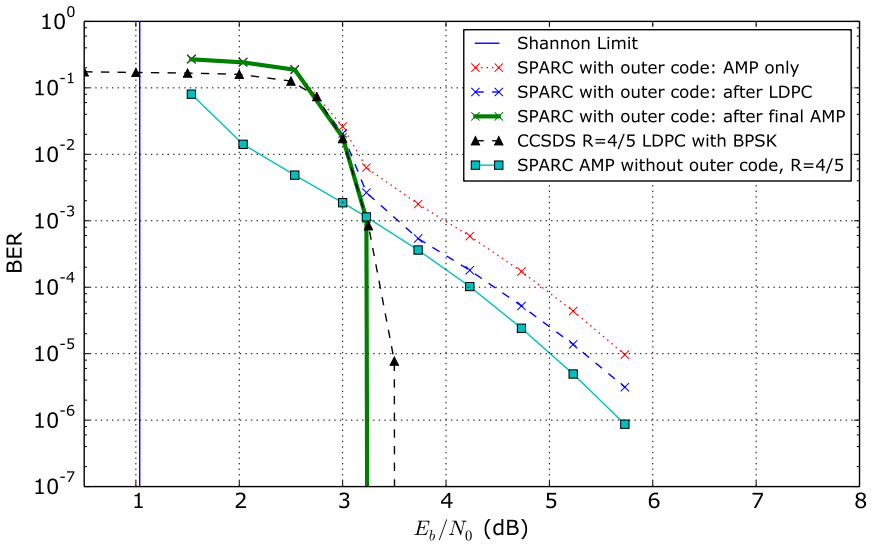


Figure 4.4: Comparison to plain AMP and to BPSK-modulated LDPC at overall rate $R = 0.8$. The SPARCs are both $L = 768$, $M = 512$. The underlying SPARC rate when the outer code is included is $R_{SPARC} = 0.94$. The BPSK-modulated LDPC is the same CCSDS LDPC code [49] used for the outer code. For this configuration, $L_{user} = 654.2$, $L_{parity} = 113.8$, $L_{unprotected} = 199.1$, $L_{protected} = 455.1$, and $L_{LDPC} = 568.9$. The SPARC with outer code has $n = 7360$ while the plain SPARC has $n = 8640$ and the raw LDPC code has $n = 5120$.

The plots show that protecting a fraction of sections with an outer code does provide a

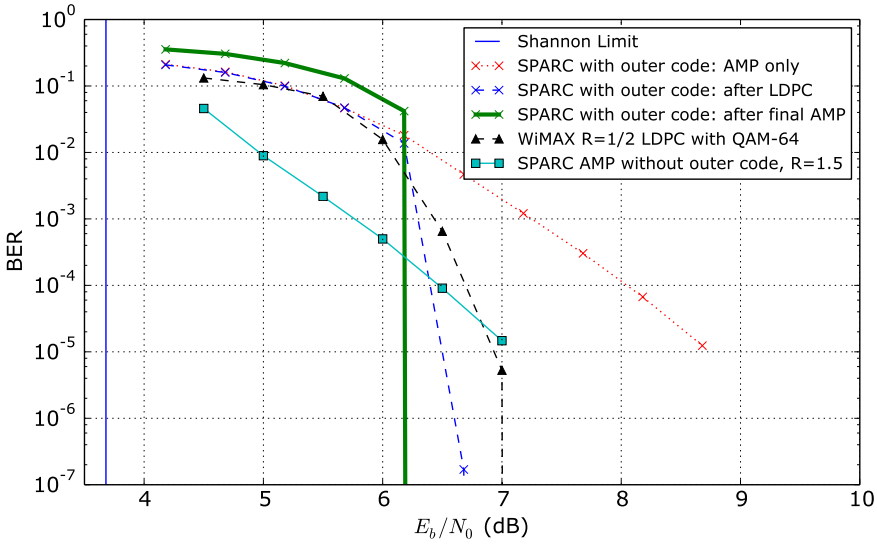


Figure 4.5: Comparison to plain AMP and to the QAM-64 WiMAX LDPC of Section 4.1 at overall rate $R = 1.5$ The SPARCs are both $L = 1024$, $M = 512$. The underlying SPARC rate including the outer code is $R_{SPARC} = 1.69$. For this configuration, $L_{user} = 910.2$, $L_{parity} = 113.8$, $L_{unprotected} = 455.1$, $L_{protected} = 455.1$, and $L_{LDPC} = 455.1$. The SPARC with outer code has $n = 5461$ while the plain SPARC has $n = 6144$ and the raw LDPC code has $n = 2304$.

steep waterfall above a threshold value of $\frac{E_b}{N_0}$. Below this threshold, the combined SPARC + outer code has worse performance than the plain rate R SPARC without the outer code. This can be explained as follows. The combined code has a higher SPARC rate $R_{SPARC} > R$, which leads to a larger section error rate for the first AMP decoder, and consequently, to worse bit-wise posteriors at the input of the LDPC decoder. For $\frac{E_b}{N_0}$ below the threshold, the noise level at the input of the LDPC decoder is beyond than the error-correcting capability of the LDPC code, so the LDPC code effectively does not correct any section errors. Therefore the overall performance is worse than the performance without the outer code.

Above the threshold, we observe that the second AMP decoder (after subtracting the contribution of the LDPC-protected sections) is successful at decoding the unprotected sections that were initially decoded incorrectly. This is especially apparent in the $R = \frac{4}{5}$ case (Figure 4.4), where the section errors are uniformly distributed over all sections due to the flat power allocation; errors are just as likely in the unprotected sections as in the protected sections.

4.2.3 Outer Code Design Choices

In addition to the various SPARC parameters discussed in the previous chapter, performance with an outer code is sensitive to what fraction of sections are protected by the outer code. When more sections are protected by the outer code, the overhead of using the outer code is also higher, driving R_{SPARC} higher for the same overall user rate R. This leads to worse performance in the initial AMP decoder, which has to operate at the higher rate R_{SPARC}. As discussed above, if R_{SPARC} is increased too much, the bit-wise posteriors input to the LDPC decoder are degraded beyond its ability to successfully decode, giving poor overall performance.

Since the number of sections covered by the outer code depends on both $\log M$ and n_{LDPC}, various trade-offs are possible. For example, given n_{LDPC}, choosing a larger value of $\log M$ corresponds to fewer sections being covered by the outer code. This results in smaller rate overhead, but increasing $\log M$ may also affect concentration of the error rates around the SE predictions, as discussed in Section 3.5.1. We conclude this chapter with two remarks about the choice of parameters for the SPARC and the outer code:

1. When using an outer code, it is highly beneficial to have good concentration of the section error rates for the initial AMP decoder. This is because a small number of errors in a single trial can usually be fully corrected by the outer code, while occasional trials with a very large number of errors cannot.

2. Due to the second AMP decoder operation, it is not necessary for all sections with low power to be protected by the outer code. For example, in Figure 4.4, all sections have equal power, and around 30% are not protected by the outer code. These sections are consequently often not decoded correctly by the first decoder. Only once the protected sections are removed is the second decoder able to correctly decode these unprotected sections. In general the aim should be to cover all or most of the sections in the flat region of the power allocation, but experimentation is necessary to determine the best trade-off.

Chapter 5

Modulated SPARCs

When encoding with the standard SPARCs considered thus far, recall that one entry out of M in each section $\ell \in [L]$ of β is selected, and is set to a fixed value $\sqrt{nP_\ell}$. This choice of column communicates $\log M$ bits of information per section, thereby communicating a total of $L \log M = nR$ bits in n uses of the channel. The decoder always knows what the fixed value will be, and need only decide which column was selected.

To increase the information communicated per section, we could instead choose from a set of values to assign to the selected column. For example, instead of always assigning $\sqrt{nP_\ell}$, we might communicate one additional bit of information by choosing either $+\sqrt{nP_\ell}$ or $-\sqrt{nP_\ell}$. In general, if the non-zero entry in each section is chosen from a K-ary constellation, we obtain $\log(KM)$ bits per section. The decoder must now determine which of the possible values was transmitted, in addition to which column was selected. The new rate is found as $nR = L \log KM$.

We call this technique *Modulated SPARCs*, referring to modulation of the non-zero entries in β, similar to pulse amplitude modulation (PAM) in baseband communications. In this chapter we derive an AMP decoder for this new structure, and for the simple case $K = 2$ (binary modulation), find its corresponding state evolution and show it remains possible to achieve the channel capacity in the large system limit as L, M, and $n \to \infty$. This new decoder is implemented and empirical results obtained.

5.1 Encoding Modulated SPARCs

To encode unmodulated SPARCs, the $L \log M$ input bits are split into L groups of $\log M$ bits, and each group converted to an integer in $[M]$. These integers map to a choice of entry in each section ℓ in β, and that corresponding entry is set to $\sqrt{nP_\ell}$, with all other entries in that section set to zero. $\{P_\ell\}_{\ell \in [L]}$ are *power allocation* coefficients, fixed a priori, such that $\sum_{\ell \in [L]} P_\ell = P$.

To illustrate, consider a very small scenario where $L = 4$ and $M = 4$. There will be $L \log M = 8$ input bits. Consider the input bits 01 10 00 10. The resulting β is:

$$\beta = \left[0, \sqrt{nP_1}, 0, 0 \mid 0, 0, \sqrt{nP_2}, 0 \mid \sqrt{nP_3}, 0, 0, 0 \mid 0, 0, \sqrt{nP_3}, 0\right]$$

where | is used to clearly delimit sections visually.

For modulated SPARCs, we instead have $L \log KM$ input bits, which are again split into L groups of $\log KM$ bits each. This time each group is further split into $\log M$ bits which will determine a choice of column as before, and $\log K$ bits which will determine the choice of modulating value.

For example, with $K = 2$, the modulating values are $\pm\sqrt{nP_\ell}$. Note that the power allocation coefficients P_ℓ are the same as in the unmodulated case; the only difference is that the non-zero entry in β may be negative as well as positive. In our illustrative example, we will still have 8 input bits, but can now set $M = 2$ and use the first bit from each group for the sign, and the second bit for the position. With the same input bits 01 10 00 10 we now obtain:

$$\beta = \left[0, -\sqrt{nP_1} \mid \sqrt{nP_2}, 0 \mid -\sqrt{nP_3}, 0 \mid \sqrt{nP_3}, 0\right]$$

For $K > 2$, the set of possible values is slightly more complicated. We require that they are evenly spaced and symmetric around 0, and furthermore that the average power in section ℓ remains nP_ℓ. We also require that K is even, as otherwise one constellation point would be at 0, leading to sections that selected the 0 point being allocated no power.

First we can find K evenly-spaced points between -1 and 1 as:

$$\left\{\pm\frac{2r-1}{K-1}\right\}_{1 \le r \le \frac{K}{2}}. \tag{5.1}$$

These have an average power of

$$\begin{aligned}\frac{2}{K}\sum_{r=0}^{\frac{K}{2}-1}\left(\frac{2r+1}{K-1}\right)^2 &= \frac{2}{K(K-1)^2}\left(1^2+3^2+\cdots+(K-1)^2\right)\\ &= \frac{K+1}{3(K-1)}.\end{aligned} \tag{5.2}$$

To obtain evenly spaced symbols $\pm a_r$ for $r = 1, \ldots, \frac{K}{2}$, with average power $\sqrt{nP_\ell}$, we use:

$$\begin{aligned}a_r &= \sqrt{nP_\ell}\sqrt{\frac{3(K-1)}{K+1}}\frac{2r-1}{K-1}\\ &= \sqrt{\frac{3nP_\ell}{K^2-1}}(2r-1).\end{aligned} \tag{5.3}$$

Note that when $K = 2$, $a_1 = \sqrt{nP_\ell}$, as expected for binary modulation.

5.2 AMP for Modulated SPARCs

The update rules for an AMP with modulated SPARCs are derived in Section 5.4. For the binary case where $K = 2$, they are:

$$\beta_i^{t+1} = \eta_i^t(A^*z^t + \beta^t = s) = \frac{\sqrt{nP_\ell}\sinh\left(\frac{s_i\sqrt{nP_\ell}}{\tau_t^2}\right)}{\sum_{k\in \text{sec}(i)}\cosh\left(\frac{s_k\sqrt{nP_\ell}}{\tau_t^2}\right)}, \quad i \in [ML] \tag{5.4}$$

$$z^t = y - A\beta^t + \frac{z^{t-1}}{\tau_{t-1}^2}\left(P - \frac{\|\beta^t\|^2}{n}\right). \tag{5.5}$$

In the general K-ary case, with the symbol set described in (5.3), the update rules are:

$$\beta_i^{t+1} = \eta_i^t(A^* z^t + \beta^t = s) = \frac{\sum_{r=1}^{K/2} a_r e^{\frac{-a_r^2}{2\tau_t^2}} \sinh\left(\frac{s_i a_r}{\tau_t^2}\right)}{\sum_{k\in \text{sec}(i)} \sum_{r=1}^{K/2} e^{\frac{-a_r^2}{2\tau_t^2}} \cosh\left(\frac{s_k a_r}{\tau_t^2}\right)} \tag{5.6}$$

$$z^t = y - A\beta^t + \frac{z^{t-1}}{\tau_{t-1}^2 n}\left(\sum_{\ell\in[L]} \frac{\sum_{r=1}^{K/2} a_r^2 e^{\frac{-a_r^2}{2\tau_t^2}} \sum_{k\in \text{sec}_\ell} \cosh\left(\frac{s_k a_r}{\tau_t^2}\right)}{\sum_{j\in \text{sec}(k)} \sum_{q=1}^{K/2} e^{\frac{-a_q^2}{2\tau_t^2}} \cosh\left(\frac{s_j a_q}{\tau_t^2}\right)} - \|\beta^t\|^2\right). \tag{5.7}$$

Note that for $K = 2$, these reduce to (5.4) and (5.5).

The required τ_t coefficients may be generated using the state evolution described in (5.8), or generated online as described in Section 3.6, using $\hat{\tau}_t^2 = \frac{\|z^t\|}{n}$. Initialise with $\beta^0 = 0$ and $z^0 = y$.

After T iterations, the decoder yields β^T. The decoded codeword, $\hat{\beta}$, is obtained by finding the entry with the maximum absolute value in each section ℓ, and quantising its value to the nearest constellation point $\pm a_r$. In the binary modulation case, we set the magnitude of the entry with the maximum absolute value to $\sqrt{nP_\ell}$, while preserving its sign. All other entries in the section are set to 0.

5.2.1 State Evolution for AMP Decoded Binary Modulated SPARCs

We can use the decoder in (5.4) and (5.5) to further derive a new state evolution for the binary modulated SPARCs, which will in turn permit a proof that the AMP decoder achieves the channel capacity in the large system limit.

The state evolution equations are:

$$\tau_0^2 = \sigma^2 + P, \quad \tau_t^2 = \sigma^2 + P(1 - x_t), \tag{5.8}$$

where

$$x_{t+1} = \sum_{\ell=1}^{L} \frac{P_\ell}{P} \mathbb{E}\left[\frac{\sinh\left(\frac{nP_\ell}{\tau_t^2} + \frac{\sqrt{nP_\ell}}{\tau_t} U_1^\ell\right)}{\cosh\left(\frac{nP_\ell}{\tau_t^2} + \frac{\sqrt{nP_\ell}}{\tau_t} U_1^\ell\right) + \sum_{j=2}^{M} \cosh\left(\frac{\sqrt{nP_\ell}}{\tau_t} U_j^\ell\right)} \right], \tag{5.9}$$

and $\{U_j^\ell\}_{j\in[M],\ell\in[L]}$ are i.i.d. $\mathcal{N}(0,1)$ random variables.

x_t tracks the expected power-weighted fraction of sections which will be decoded correctly at iteration t, while τ_t^2 represents the combined power of the channel noise and the undecoded sections. This interpretation is justified by the following result:

Proposition 5.1. *Under the assumption that $s^t = \beta + \tau_t Z$, where $Z \in \mathbb{R}^{ML}$ is i.i.d. $\sim \mathcal{N}(0,1)$ and independent of β, the quantity x^{t+1} defined in* (5.9) *satisfies:*

$$x_{t+1} = \frac{1}{nP}\mathbb{E}[\beta\beta^{t+1}], \quad 1 - x_{t+1} = \frac{1}{nP}\mathbb{E}[\|\beta - \beta^{t+1}\|^2], \tag{5.10}$$

and consequently, $\tau_{t+1}^2 = \sigma^2 + P(1 - x_t) = \sigma^2 + \frac{1}{n}\mathbb{E}[\|\beta - \beta^{t+1}\|^2]$.

Proof. As in the unmodulated case, we relabel $\{Z_k\}_{k\in[ML]}$ as $\{U_j^\ell\}_{j\in[M],\ell\in[L]}$, where $U_j^\ell = Z_{(\ell-1)M+j}$, so that U^ℓ represents the length M vector $\{U_j^\ell\}_{j\in[M]}$ and U is the length ML vector $\{U^\ell\}_{\ell\in[L]} = Z$.

We have

$$
\begin{aligned}
&\frac{1}{nP}\mathbb{E}[\beta\beta^{t+1}] = \frac{1}{nP}\mathbb{E}[\beta\eta^t(\beta+\tau_t U)] \\
&\overset{\text{(a)}}{=} \frac{1}{nP}\sum_{\ell=1}^{L}\sum_{i=1}^{M}\frac{1}{M}\Bigg(\frac{1}{2}\mathbb{E}\left[\sqrt{nP_\ell}\,\eta_i^t(\beta_\ell+\tau_t U^\ell)\Big|\beta_{\ell i}=\sqrt{nP_\ell}\right] \\
&\qquad + \frac{1}{2}\mathbb{E}\left[-\sqrt{nP_\ell}\,\eta_i^t(\beta_\ell+\tau_t U^\ell)\Big|\beta_{\ell i}=-\sqrt{nP_\ell}\right]\Bigg) \\
&\overset{\text{(b)}}{=} \frac{1}{nP}\sum_{\ell=1}^{L}\sum_{i=1}^{M}\frac{1}{2M}\Bigg(\mathbb{E}\left[\frac{\sqrt{nP_\ell}\cdot\sqrt{nP_\ell}\sinh\left(\frac{\sqrt{nP_\ell}}{\tau_t}\left(\sqrt{nP_\ell}+\tau_t U_i^\ell\right)\right)}{\cosh\left(\frac{\sqrt{nP_\ell}}{\tau_t}\left(\sqrt{nP_\ell}+\tau_t U_i^\ell\right)\right)+\sum_{j=2}^{M}\cosh\left(\frac{\sqrt{nP_\ell}}{\tau_t}U_j^\ell\right)}\right] \\
&\qquad + \mathbb{E}\left[\frac{-\sqrt{nP_\ell}\cdot\sqrt{nP_\ell}\sinh\left(\frac{\sqrt{nP_\ell}}{\tau_t}\left(-\sqrt{nP_\ell}+\tau_t U_i^\ell\right)\right)}{\cosh\left(\frac{\sqrt{nP_\ell}}{\tau_t}\left(-\sqrt{nP_\ell}+\tau_t U_i^\ell\right)\right)+\sum_{j\in\text{sec}_\ell\setminus i}\cosh\left(\frac{\sqrt{nP_\ell}}{\tau_t}U_j^\ell\right)}\right]\Bigg) \\
&\overset{\text{(c)}}{=} \frac{1}{nP}\sum_{\ell=1}^{L}\frac{1}{2}\Bigg(\mathbb{E}\left[\frac{nP_\ell\sinh\left(\frac{nP_\ell}{\tau_t^2}+\frac{\sqrt{nP_\ell}}{\tau_t}U_1^\ell\right)}{\cosh\left(\frac{nP_\ell}{\tau_t^2}+\frac{\sqrt{nP_\ell}}{\tau_t}U_1^\ell\right)+\sum_{j=2}^{M}\cosh\left(\frac{\sqrt{nP_\ell}}{\tau_t}U_j^\ell\right)}\right] \\
&\qquad + \mathbb{E}\left[\frac{-nP_\ell\sinh\left(-\frac{nP_\ell}{\tau_t^2}-\frac{\sqrt{nP_\ell}}{\tau_t}U_1^\ell\right)}{\cosh\left(-\frac{nP_\ell}{\tau_t^2}-\frac{\sqrt{nP_\ell}}{\tau_t}U_1^\ell\right)+\sum_{j=2}^{M}\cosh\left(\frac{\sqrt{nP_\ell}}{\tau_t}U_j^\ell\right)}\right]\Bigg) \\
&\overset{\text{(d)}}{=} \sum_{\ell=1}^{L}\frac{P_\ell}{P}\mathbb{E}\left[\frac{\sinh\left(\frac{nP_\ell}{\tau_t^2}+\frac{\sqrt{nP_\ell}}{\tau_t}U_1^\ell\right)}{\cosh\left(\frac{nP_\ell}{\tau_t^2}+\frac{\sqrt{nP_\ell}}{\tau_t}U_1^\ell\right)+\sum_{j=2}^{M}\cosh\left(\frac{\sqrt{nP_\ell}}{\tau_t}U_j^\ell\right)}\right] = x_{t+1}. \qquad (5.11)
\end{aligned}
$$

In the above, (a) is found by splitting the expectation into a sum of equally probable locations for the nonzero column i and into both possibilities of nonzero value $+\sqrt{nP_\ell}$ and $-\sqrt{nP_\ell}$, then noting that $\beta_\ell\beta^{t+1}$ will be zero for all entries except the ith, where it takes the value shown. (b) is found by then substituting in the expression for η_i in (5.4). (c) is found by eliminating i; since U is i.i.d., we can fix $i = 1$, and as the Gaussian density is symmetric around 0, we can replace U with $-U$ in the second expectation. (d) is found by noting that $\sinh(-x) = -\sinh(x)$ and $\cosh(-x) = \cosh(x)$, and therefore combining the two expectations into one.

For $1 - x_{t+1}$, the proof is identical to the unmodulated case given in [1]:

$$\frac{1}{nP}\mathbb{E}[\|\beta - \beta^{t+1}\|^2] = 1 + \frac{\mathbb{E}[\|\beta^{t+1}\|^2] - 2\mathbb{E}[\beta\beta^{t+1}]}{nP} \tag{5.12}$$

Under the assumption $s^t = \beta + \tau_t Z$, and with $\beta^{t+1} = \mathbb{E}[\beta \mid s^t]$, we have:

$$\begin{aligned}\mathbb{E}[\|\beta^{t+1}\|^2] = \mathbb{E}[\|\mathbb{E}[\beta \mid s^t]\|^2] = \mathbb{E}[(\mathbb{E}[\beta \mid s^t] - \beta + \beta) \cdot \mathbb{E}[\beta \mid s^t]] \\ \overset{\text{(a)}}{=} \mathbb{E}[\beta\mathbb{E}[\beta \mid s^t]] = \mathbb{E}[\beta\beta^{t+1}],\end{aligned} \tag{5.13}$$

where (a) follows due to orthogonality. This gives

$$\frac{1}{nP}\mathbb{E}[\|\beta - \beta^{t+1}\|^2] = 1 - \frac{\mathbb{E}[\beta\beta^{t+1}]}{nP} = 1 - x_{t+1}. \tag{5.14}$$

□

Having established the state evolution of the AMP decoder with binary modulated SPARCs, we can use these expressions to prove that the decoder achieves the channel capacity in the large system limit.

5.2.2 Proof of Achieving Capacity with AMP Decoder

To prove that the AMP decoder achieves the channel capacity with binary modulated SPARCs, we will show that for any rate $R < \mathcal{C}$, the expected fraction of sections which are correctly decoded, $x(\tau)$, tends to 1 in a finite number of iterations. This leads to Theorem 5.1, which states that these expected values track the performance of the AMP decoder with high probability, so that the section error rate in the large system limit will converge to zero almost surely.

We will require the same exponential power allocation used in the original unmodulated proof, described in [1], and many of the steps will be similar. The main technical difference is in the proof of Lemma 5.1, due to the new form of $x(\tau)$. Otherwise the statement of both Lemmas is the same as in [1].

Lemma 5.1. *For any power allocation $\{P_\ell\}_{\ell\in[L]}$ that is non-increasing with ℓ, and with the state evolution equations given by* (5.8) *and* (5.9)*, we have*

$$\bar{x}(\tau) := \lim x(\tau) = \lim \sum_{\ell=1}^{\lfloor \xi^*(\tau)L \rfloor} \frac{P_\ell}{P}, \tag{5.15}$$

where $\xi^*(\tau)$ is the supremum of all $\xi \in (0, 1]$ that satisfy

$$\lim LP_{\lfloor \xi L \rfloor} > 2R\tau^2 \ln 2. \tag{5.16}$$

Proof. In Section 5.2.3. □

Furthermore, when using the exponential power allocation

$$P_\ell = P \cdot \frac{2^{2\mathcal{C}/L} - 1}{1 - 2^{-2\mathcal{C}}} \cdot 2^{-2\mathcal{C}\ell/L}, \quad \ell \in [L], \tag{5.17}$$

we will obtain Lemma 5.2:

Lemma 5.2. *For the power allocation* (5.17)*, we have for $t = 0, 1, \ldots$:*

$$\bar{x}_t := \lim x_t = \frac{(1 + \mathit{snr}) - (1 + \mathit{snr})^{1-\xi_{t-1}}}{\mathit{snr}}, \tag{5.18}$$

$$\bar{\tau}_t^2 := \lim \tau_t^2 = \sigma^2 + P(1 - \bar{x}_t) = \sigma^2(1 + \mathit{snr})^{1-\xi_{t-1}}, \tag{5.19}$$

where $\xi_{-1} = 0$, and for $t > 0$,

$$\xi_t = \min\left\{\left(\frac{1}{2\mathcal{C}} \log\left(\frac{\mathcal{C}}{R}\right) + \xi_{t-1}\right), 1\right\}. \tag{5.20}$$

Proof. As the asymptotic state evolution equations in Lemma 5.1 are the same as those for the unmodulated SPARC, the original proof for the unmodulated SPARC [1, Appendix C] applies unchanged. □

From Lemma 5.2 we see that ξ_t increases monotonically to 1, in steps of size $\frac{1}{2\mathcal{C}} \log\left(\frac{\mathcal{C}}{R}\right)$. Therefore after

$$T^* = \left\lceil \frac{2\mathcal{C}}{\log(\mathcal{C}/R)} \right\rceil, \tag{5.21}$$

iterations, the decoder converges, with every section expected to decode correctly.

Finally we use the proof from [1, Theorem 1] to show that the AMP decoder tracks this expected behaviour in the large system limit, such that when the expected section error rate

converges to zero, shown by Lemma 5.2, then so too will the actual section error rate of the AMP decoder. This gives Theorem 5.1, completing the proof.

Theorem 5.1. *Fix any rate $R < C$. Consider a sequence of rate R binary modulated SPARCs $\{S_n\}$, indexed by block length n, with design matrix parameters L and $M = L^a$ for some $a > 0$ such that $nR = L \log 2M$, and using the exponential power allocation given in* (5.17)*. Consider the AMP decoder defined by* (5.4) *and* (5.5)*, with τ_t given by $\bar{\tau}_t$ from Lemma 5.2. After T^* iterations, the decoder yields β^{T^*}, which is decoded to an estimated codeword $\hat{\beta}$.*

Defining the section error rate as

$$\mathcal{E}_{sec} := \frac{1}{L} \sum_{\ell=1}^{L} \mathbf{1} \left\{ \hat{\beta}_\ell \neq \beta_\ell \right\}, \tag{5.22}$$

then $\mathcal{E}_{sec}$ converges to zero almost surely, i.e., for any $\epsilon > 0$,

$$\lim_{n_0 \to \infty} P(\mathcal{E}_{sec}(S_n) < \epsilon, \forall n \geq n_0) = 1. \tag{5.23}$$

Proof. As the asymptotic state evolution equations in Lemma 5.2, for the exponential power allocation in (5.17), are the same as those for the unmodulated SPARC, the original proof for the unmodulated SPARC [1, Section V] applies unchanged. □

5.2.3 Proof of Lemma 5.1

We consider $x_{t+1} = x(\tau_t)$, where the function $x(\tau)$ is:

$$x(\tau) := \sum_{\ell=1}^{L} \frac{P_\ell}{P} \mathbb{E} \left[\frac{\sinh\left(\frac{\sqrt{nP_\ell}}{\tau_t} \left(\frac{\sqrt{nP_\ell}}{\tau_t} + U_1^\ell \right) \right)}{\cosh\left(\frac{\sqrt{nP_\ell}}{\tau_t} \left(\frac{\sqrt{nP_\ell}}{\tau_t} + U_1^\ell \right) \right) + \sum_{j=2}^{M} \cosh\left(\frac{\sqrt{nP_\ell}}{\tau_t} U_j^\ell \right)} \right]. \tag{5.24}$$

We will show that asymptotically this expression can be written as

$$\bar{x}(\tau) := \lim x(\tau) = \lim \sum_{\ell=1}^{\lfloor \xi^*(\tau) L \rfloor} \frac{P_\ell}{P}, \tag{5.25}$$

where $\xi^*(\tau)$ is the supremum of all $\xi \in (0, 1]$ which satisfy

$$\lim LP_{\lfloor \xi L \rfloor} > 2R\tau^2 \ln 2.$$

This can be viewed as summing over all sections where the power allocated to that section exceeds some threshold which leads to correct decoding asymptotically. As the power allocation is non-increasing, once some P_ℓ value does not exceed the threshold, no subsequent P_ℓ for some larger ℓ will either.

First, write $x(\tau)$ as:

$$x(\tau) = \sum_{\ell=1}^{L} \frac{P_\ell}{P} \mathcal{E}_\ell(\tau), \tag{5.26}$$

where

$$\mathcal{E}_\ell(\tau) := \mathbb{E}\left[\frac{\sinh\left(\frac{\sqrt{nP_\ell}}{\tau}\left(U_1 + \frac{\sqrt{nP_\ell}}{\tau}\right)\right)}{\cosh\left(\frac{\sqrt{nP_\ell}}{\tau}\left(U_1 + \frac{\sqrt{nP_\ell}}{\tau}\right)\right) + \sum_{j=2}^{M} \cosh\left(\frac{\sqrt{nP_\ell}}{\tau} U_j\right)} \right]. \tag{5.27}$$

We will show that for $\ell = \lfloor \xi L \rfloor$ such that $\lim LP_{\lfloor \xi L \rfloor} > 2R\tau^2 \ln 2$, $\mathcal{E}_\ell(\tau)$ will tend to 1, ensuring that the respective section decodes in the large system limit. We will then upper bound the absolute value of $\mathcal{E}_\ell(\tau)$ for $\lim LP_{\lfloor \xi L \rfloor} < 2R\tau^2 \ln 2$ and show that it tends to 0, as it could otherwise become large and negative. Together this gives a lower bound on decoder performance.

Case 1

We now show that $\lim \mathcal{E}_\ell = 1$ when $\lim LP_\ell > 2\ln(2)R\tau^2$.

Let $\nu_\ell = \frac{LP_\ell}{R\tau^2 \ln 2}$; then, using $n = \frac{L \log(2M)}{R}$, we obtain $\frac{nP_\ell}{\tau^2} = \nu_\ell \ln(2M)$, then for brevity write:

$$\Delta = \sqrt{\nu_\ell \ln(2M)} \tag{5.28}$$

$$V = \exp\left(\Delta\left(U_1 + \Delta\right)\right) \tag{5.29}$$

$$X = \sum_{j=2}^{M} \left(\exp\left(\Delta U_j\right) + \exp\left(-\Delta U_j\right)\right). \tag{5.30}$$

We can now write (5.27) in a simpler form:

$$\mathcal{E}_\ell(\tau) = \mathbb{E}\left[\frac{V - V^{-1}}{V + V^{-1} + X}\right]. \tag{5.31}$$

We can rewrite the expectation as follows by first conditioning on U_1:

$$\mathcal{E}_\ell(\tau) = \mathbb{E}_{U_1}\mathbb{E}_X\left[\frac{V - V^{-1}}{V + V^{-1} + X}\middle| U_1\right]. \tag{5.32}$$

Noting that V is a function of U_1, if $V < 1$ then $\mathcal{E}_\ell(\tau)$ will be negative, which occurs if $U_1 < -\Delta$. We therefore split the expectation into one integral over the range $U_1 \leq -\sqrt{\Delta}$, where $V \leq 1$, and a second integral where $U_1 > -\sqrt{\Delta}$, giving some slack in the inequality. Note that since ν_ℓ is an order 1 quantity, $\Delta >> 1$ for large M. We have

$$\begin{aligned}
\mathcal{E}_\ell(\tau) = &\int_{U_1 \leq -\sqrt{\Delta}} p(u_1)\mathbb{E}_X\left[\frac{V - V^{-1}}{V + V^{-1} + X}\middle| U_1 = u_1\right] du_1 \\
&+ \int_{U_1 > -\sqrt{\Delta}} p(u_1)\mathbb{E}_X\left[\frac{V - V^{-1}}{V + V^{-1} + X}\middle| U_1 = u_1\right] du_1 \\
&= I_1 + I_2.
\end{aligned}$$

We first find a lower bound on I_1. Note from the definition of $\mathcal{E}_\ell$ in (5.27) that

$$\mathcal{E}_\ell = \frac{\sinh(.)}{\cosh(.) + \sum \cosh(.)} \in [-1, 1], \tag{5.33}$$

and therefore

$$\min_{U_1 \leq -\sqrt{\Delta}} \mathcal{E}_\ell = -1, \tag{5.34}$$

giving

$$I_1 \geq \int_{U_1 \leq -\sqrt{\Delta}} -p(u_1)du_1 \tag{5.35}$$

$$= -Q(\sqrt{\Delta}), \tag{5.36}$$

where $Q(.)$ is the tail probability of the standard normal distribution, $Q(x) = \frac{1}{2\pi}\int_x^\infty e^{-x^2/2}dx$. Applying the Chernoff bound of $Q(x) \leq e^{-\frac{x^2}{2}}$ for $x > 0$ gives

$$I_1 \geq e^{-\frac{\Delta}{2}}. \tag{5.37}$$

For I_2, where $U_1 > -\sqrt{\Delta}$, $(V - V^{-1})$ is always positive, and so

$$\frac{V - V^{-1}}{V + V^{-1} + X}$$

is a convex function of X. We may therefore use Jensen's inequality to write

$$I_2 \geq \int_{U_1 > -\sqrt{\Delta}} p(u_1)\frac{V - V^{-1}}{V + V^{-1} + \mathbb{E}X}du_1, \tag{5.38}$$

where

$$\mathbb{E}X = \sum_{j=2}^{M} \left(\mathbb{E}e^{\Delta U_j} + \mathbb{E}e^{-\Delta U_j}\right). \tag{5.39}$$

Using the Gaussian moment generating function $\mathbb{E}e^{tX} = e^{\frac{t^2}{2}}$, and recalling $\Delta = \sqrt{\nu_\ell \ln(2M)}$,

$$\mathbb{E}X = (M-1)2e^{\nu_\ell(\ln 2M)/2} \leq (2M)^{1+\nu_\ell/2}, \tag{5.40}$$

giving

$$I_2 \geq \int_{U_1 > -\sqrt{\Delta}} p(u_1)\frac{V - V^{-1}}{V + V^{-1} + (2M)^{1+\nu_\ell/2}}du_1. \tag{5.41}$$

By writing

$$\begin{aligned}\frac{V - V^{-1}}{V + V^{-1} + (2M)^{1+\nu_\ell/2}} &= \frac{\sinh(\Delta(U_1 + \Delta))}{\cosh(\Delta(U_1 + \Delta)) + (2M)^{1+\nu_\ell/2}} \\ &= \frac{1}{\coth(\Delta(U_1 + \Delta)) + \operatorname{csch}(\Delta(U_1 + \Delta))(2M)^{1+\nu_\ell/2}},\end{aligned} \tag{5.42}$$

we see that the RHS of (5.42) increases with U_1 for $U_1 > -\sqrt{\Delta}$, and therefore

$$\operatorname*{arg\,min}_{U_1 \geq -\sqrt{\Delta}} \frac{V - V^{-1}}{V + V^{-1} + (2M)^{1+\nu_\ell/2}} = -\sqrt{\Delta}, \tag{5.43}$$

and therefore we bound the integral of (5.41):

$$\begin{aligned}I_2 &\geq \int_{U_1 > -\sqrt{\Delta}} p(u_1) \frac{\sinh(\Delta(-\sqrt{\Delta} + \Delta))}{\cosh(\Delta(-\sqrt{\Delta} + \Delta)) + (2M)^{1+\nu_\ell/2}} du_1 \\ &= \frac{Q(-\sqrt{\Delta}) \sinh(\Delta^2 - \Delta^{3/2})}{\cosh(\Delta^2 - \Delta^{3/2}) + (2M)^{1+\nu_\ell/2}}.\end{aligned} \tag{5.44}$$

Finally we recombine I_1 and I_2 to obtain

$$\mathcal{E}_\ell(\tau) \geq e^{-\Delta/2} + \frac{Q(-\sqrt{\Delta}) \sinh(\Delta^2 - \Delta^{3/2})}{\cosh(\Delta^2 - \Delta^{3/2}) + (2M)^{1+\nu_\ell/2}}. \tag{5.45}$$

Taking the limit as $M \to \infty$, recalling that $\Delta = \sqrt{\nu_\ell \ln(2M)}$, and noting that

$$Q(-\sqrt{\Delta}) = 1 - Q(\sqrt{\Delta}) \geq 1 - e^{-\Delta/2} \to 1 \text{ as } M \to \infty, \tag{5.46}$$

we obtain

$$\lim \mathcal{E}_\ell(\tau) \geq 0 + \lim \frac{1 \sinh(\Delta^2)}{\cosh(\Delta^2) + (2M)^{1+\nu_\ell/2}} \tag{5.47}$$

$$= \lim \frac{e^{\nu_\ell \ln 2M} - e^{-\nu_\ell \ln 2M}}{e^{\nu_\ell \ln 2M} + e^{-\nu_\ell \ln 2M} + (2M)^{1+\nu_\ell/2}} \tag{5.48}$$

$$= \lim \frac{(2M)^{\nu_\ell}}{(2M)^{\nu_\ell} + (2M)^{1+\nu_\ell/2}}, \tag{5.49}$$

$$= 1 \text{ for } \nu_\ell > 2. \tag{5.50}$$

We therefore have that $\mathcal{E}_\ell \to 1$ when $\nu_\ell > 2$.

Case 2

Next, we show that $\mathcal{E}_\ell \to 0$ when $\nu_\ell < 2$. Using $\Delta = \sqrt{\nu_\ell \ln 2M}$, we note that $V = e^{\Delta(U_1+\Delta)} = (2M)^{\nu_\ell} e^{\Delta U_1}$. We have that

$$\mathcal{E}_\ell(\tau) = \mathbb{E}\left[\frac{V - V^{-1}}{V + V^{-1} + X}\right]$$

therefore,

$$\begin{aligned}
|\mathcal{E}_\ell(\tau)| &\leq \mathbb{E}\left[\frac{V + V^{-1}}{V + V^{-1} + X}\right] \\
&= \mathbb{E}\left[\left(1 + \frac{X}{V + V^{-1}}\right)^{-1}\right] \\
&= \mathbb{E}\left[\left(1 + \frac{\sum_{j=2}^{M} e^{\Delta U_j} + e^{-\Delta U_j}}{(2M)^{\nu_\ell} e^{\Delta U_1} + (2M)^{-\nu_\ell} e^{-\Delta U_1}}\right)^{-1}\right] \\
&\leq \mathbb{E}\left[\left(1 + \frac{\max\limits_{2\leq j\leq M} e^{\Delta U_j} + e^{-\Delta U_j}}{(2M)^{\nu_\ell} e^{\Delta U_1} + (2M)^{-\nu_\ell} e^{-\Delta U_1}}\right)^{-1}\right]
\end{aligned} \tag{5.51}$$

Let $\mathcal{F}$ be the event where:

1. For a small $\epsilon > 0$, $\max\limits_{2\leq j\leq M} |U_j| \geq \sqrt{2 \ln M}(1 - \epsilon)$, and
2. $-\sqrt{\Delta} \leq U_1 \leq \sqrt{\Delta}$.

When $\mathcal{F}$ holds, we have:

$$\frac{\max\limits_{2\le j\le M} e^{\Delta U_j}+e^{-\Delta U_j}}{(2M)^{\nu_\ell}e^{\Delta U_1}+(2M)^{-\nu_\ell}e^{-\Delta U_1}} \ge \frac{(2M)^{-\nu_\ell}e^{\Delta\sqrt{2\ln M}(1-\epsilon)}}{e^{\Delta\sqrt{\Delta}}+(2M)^{-2\nu_\ell}e^{\Delta\sqrt{\Delta}}} \tag{5.52}$$

$$\ge \frac{2^{-\nu_\ell}M^{\sqrt{2\nu_\ell}(1-\epsilon)-\nu_\ell}}{(1+(2M)^{-2\nu_\ell})\exp\left((\nu_\ell\ln 2M)^{\frac{3}{4}}\right)} \tag{5.53}$$

For small ϵ and $\nu_\ell < 2$, there exists some positive constant δ such that

$$\sqrt{2\nu_\ell}(1-\epsilon)-\nu_\ell > \delta, \tag{5.54}$$

and so continuing from (5.53),

$$\frac{2^{-\nu_\ell}M^{\sqrt{2\nu_\ell}(1-\epsilon)-\nu_\ell}}{(1+(2M)^{-2\nu_\ell})e^{(\nu_\ell\ln 2M)^{\frac{3}{4}}}} \ge \frac{2^{-\nu_\ell}M^{\delta}}{2e^{(\nu_\ell\ln 2M)^{\frac{3}{4}}}} = \frac{2^{-\nu_\ell}e^{\delta\ln M}}{2e^{(\nu_\ell\ln 2M)^{\frac{3}{4}}}} \to \infty \quad \text{as} \quad M\to\infty. \tag{5.55}$$

By using (5.55) in (5.51), and noting $|\mathcal{E}_\ell(\tau)| < 1$, we have:

$$|\mathcal{E}_\ell(\tau)| \le \mathbb{P}(\mathcal{F}^c)\cdot 1 + \mathbb{P}(\mathcal{F})\cdot\frac{2e^{(\nu_\ell\ln 2M)^{\frac{3}{4}}}}{2^{-\nu_\ell}M^{\delta}}, \tag{5.56}$$

and using standard bounds for the standard normal distribution,

$$\begin{aligned} P(\mathcal{F}^c) &\le \mathbb{P}\left(\max_{2\le j\le M}|U_j| < \sqrt{2\ln M}(1-\epsilon)\right) + P\left(|U_1| > (\nu_\ell\ln 2M)^{\frac{1}{4}}\right) \\ &\to 0 \quad \text{as} \quad M\to\infty, \end{aligned} \tag{5.57}$$

therefore for $\nu_\ell < 2$, $|\mathcal{E}_\ell(\tau)| \to 0$ as $M\to\infty$. This completes the proof for Lemma 5.1.

An interesting direction for future work is to derive the state evolution equations for general K-ary modulated SPARCs, which may lead to a similar proof of achieving capacity in the K-ary case.

5.3 Implementation and Simulation Results

Implementing the binary modulated SPARCs requires only minor modifications compared to the unmodulated SPARCs. The most significant numerical detail comes in regularising the exponent terms in η, which otherwise quickly exceed the available numerical range and lead to overflow.

In the unmodulated case, detailed in Chapter 3, we find $\mathsf{S} = \max_{j \in \text{sec}(i)} \frac{s_j \sqrt{nP_\ell}}{\tau_t^2}$ and factor out a common e^{S} from each term. The common factors cancel, and the resulting terms in the exponents are all less than or equal to zero, preventing any overflow. For the new η in (5.4), the s_j terms may now be negative, so we instead constrain the maximum absolute value, redefining $\mathsf{S} = \max_{j \in \text{sec}(i)} |\frac{s_j \sqrt{nP_\ell}}{\tau_t^2}|$. By subtracting S from each exponent before computing the exponential function, we are likewise able to prevent overflow.

The simulation must also now encode one message bit into the sign for each section, in addition to $\log M$ bits of column choice, and this must also be reflected in the decoder and error counter. Besides these two modifications, the rest of the AMP routine, including the z update, the early termination, and the online estimation of τ, all remain unchanged.

Figure 5.1 shows the results of simulations run to compare the binary modulated SPARC performance to the standard SPARCs considered for comparison with LDPC codes, at rates $R = 0.8, 1.0, 1.5$. The reference unmodulated SPARC simulations are the same results shown in Figures 4.1, 4.2, and 4.4, all with $M = 512$. The binary modulated SPARCs have the same L, but have $M = 512$ and $M = 256$ for comparison. All the SPARCs have the same power allocation.

We observe almost indistinguishable results between the unmodulated SPARCs with $M = 512$ and the binary modulated SPARCs with $M = 256$. Since $nR = L \log(2M)$ for the binary modulated SPARCs, when $M_{\text{modulated}} = M_{\text{unmodulated}}/2$ then n will be identical between the two. Achieving the same error performance at the same block length and rate but for a factor of two reduction in M allows computational complexity benefits, and demonstrates the feasibility of the modulated SPARC design presented in this chapter.

For the binary modulated SPARCs with $M = 512$, we see improved performance in the $R \leq 1.0$ experiments, while at $R = 1.5$ the larger M has a detrimental effect of approximately the same magnitude. This is likely due to the concentration effects discussed in Chapter 3, but more refined non-asymptotic analysis would be required to estimate the scaling with M for modulated SPARCs. In practice, numerical optimisation would be required to find the best

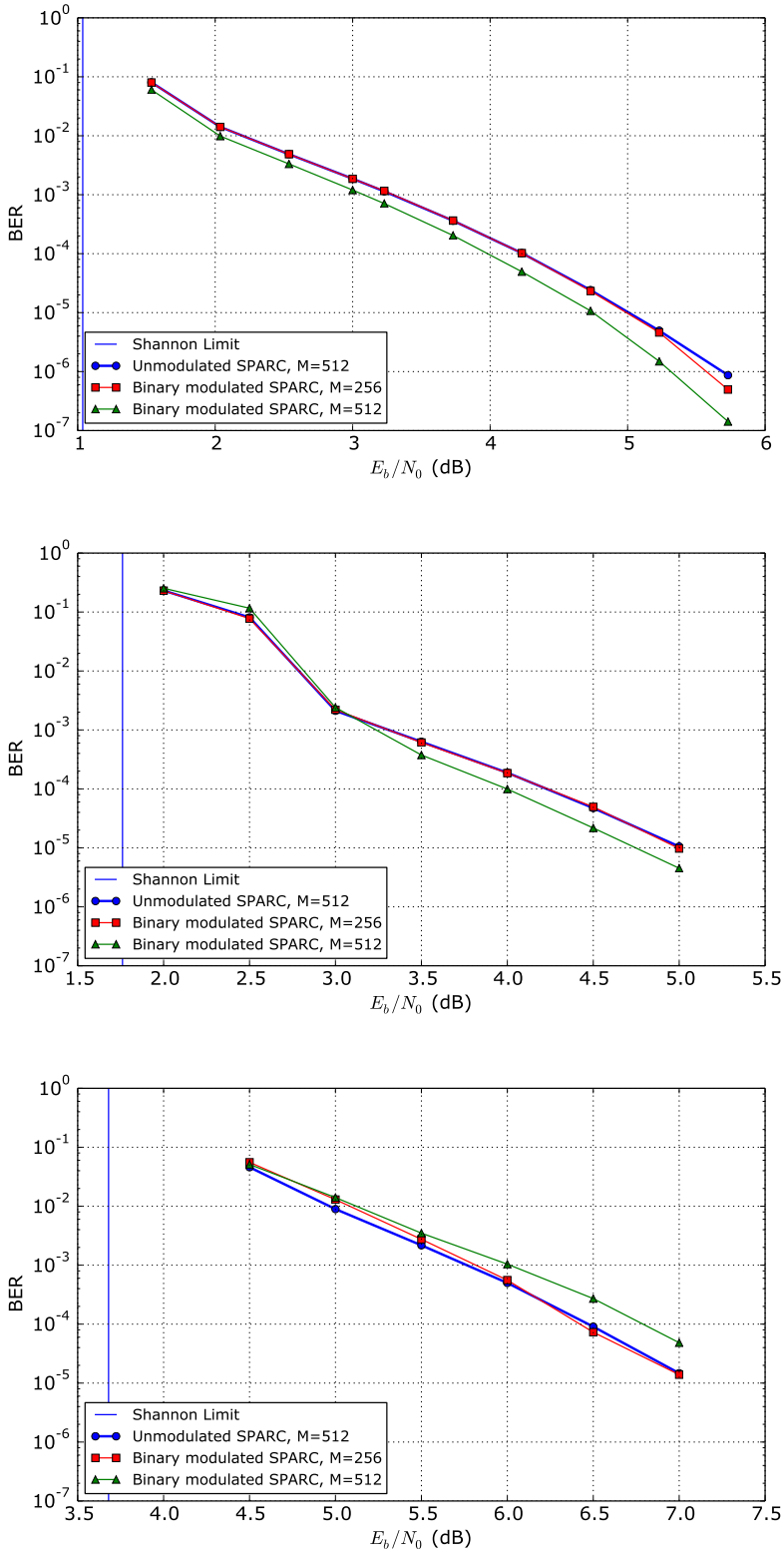


Figure 5.1: Bit error rates for binary modulated and unmodulated SPARCs. Top: $R = 0.8$, $L = 768$, $R_{\text{PA}} = 0$, centre: $R = 1.0$, $L = 1024$, $R_{\text{PA}} = 0$, bottom: $R = 1.5$, $L = 1024$, $R_{\text{PA}} = 1.53$.

value of M in a particular channel, as with the unmodulated SPARCs.

An interesting direction for future work is to investigate the performance of the general K-ary modulated SPARCs, and the tradeoff between performance and decoder complexity with increasing K.

5.4 Derivation of AMP for Modulated SPARCs

In this section we derive the update rules for z_a^t and β_i^{t+1} required for an AMP decoder using modulated SPARCs. We consider the general case of a K-ary modulated SPARC, where for each section $\ell \in [L]$ of β, there are M entries, one of which is non-zero and takes one of a set of K possible values $\{\pm a_r\}_{r\in[K/2]}$, with K even.

The values for a_r were found in (5.3) to be:

$$a_r = \sqrt{\frac{3nP_\ell}{K^2-1}}(2r-1), \quad 1 \le r \le \frac{K}{2}.$$

This choice of a_r gives an average power of nP_ℓ. When $K = 2$ we recover the simpler binary modulation case, where the non-zero values are either $+\sqrt{nP_\ell}$ or $-\sqrt{nP_\ell}$. Therefore this derivation will be used for both the simpler binary case which has been analysed in more detail, and for the general PAM case.

First we will obtain a new form for η_i^t which is used to determine β_i^{t+1}, and then we will use this new η_i^t to derive the AMP and thus an update rule for z_a^t.

5.4.1 Derivation of η_i^t

As with the unmodulated AMP in [1], we start by using the property that the test statistic $s^t := \beta^t + A^* z^t$ is asymptotically (as $n \to \infty$) distributed as $\beta + \tau_t Z$, where Z is an i.i.d. $\mathcal{N}(0,1)$ random vector independent of β.

Given this property, we would like to find $\beta^{t+1}(s^t)$ as a Bayes optimal estimate of β given $s = s^t$, a noisy observation of β:

$$\begin{aligned}\beta_i^{t+1} &= \eta_i^t(s) = \mathbb{E}\left[\beta_i \mid \beta + \tau_t Z = s\right] \\ &= \mathbb{E}\left[\beta_i \mid \{\beta_j + \tau_t Z_j = s_j\}_{j\in \text{sec}(i)}\right] \\ &= \sum_{r=1}^{K/2} a_r \mathbb{P}(\beta_i = a_r \mid \{s_j\}_{j\in\text{sec}(i)}) - a_r \mathbb{P}(\beta_i = -a_r \mid \{s_j\}_{j\in\text{sec}(i)}) \end{aligned} \tag{5.58}$$

Use Bayes' rule to expand $\mathbb{P}(\beta_i = a \mid \{s_j\}_{j\in\text{sec}(i)})$, we have

$$\mathbb{P}(\beta_i = a \mid \{s_j\}_{j\in\text{sec}(i)}) = \frac{\mathbb{P}(\{s_j\}_{j\in\text{sec}(i)} \mid \beta_i = a)\mathbb{P}(\beta_i = a)}{\mathbb{P}(\{s_j\}_{j\in\text{sec}(i)})}, \tag{5.59}$$

where

$$\begin{aligned}\mathbb{P}(\{s_j\}_{j\in\text{sec}(i)} \mid \beta_i = a) &\propto e^{\frac{-(s_i-a)^2}{2\tau_t^2}} \prod_{j\in\text{sec}(i)\setminus i} e^{\frac{-s_j^2}{2\tau_t^2}} \\ &= e^{\frac{2s_i a - a^2}{2\tau_t^2}} \prod_{j\in\text{sec}(i)} e^{\frac{-s_j^2}{2\tau_t^2}}, \end{aligned} \tag{5.60}$$

$$\mathbb{P}(\beta_i = a) = \frac{1}{KM}, \tag{5.61}$$

$$\begin{aligned}\mathbb{P}(\{s_j\}_{j\in\text{sec}(i)}) &= \sum_{k\in\text{sec}(i)} \sum_{r=1}^{K/2} \big[\mathbb{P}(\{s_j\}_{j\in\text{sec}(i)} \mid \beta_k = a_r)\mathbb{P}(\beta_k = a_r) \\ &\qquad + \mathbb{P}(\{s_j\}_{j\in\text{sec}(i)} \mid \beta_k = -a_r)\mathbb{P}(\beta_k = -a_r)\big] \\ &= \frac{1}{KM} \sum_{k\in\text{sec}(i)} \sum_{r=1}^{K/2} \left(e^{\frac{2s_k a_r - a_r^2}{2\tau_t^2}} + e^{\frac{-2s_k a_r - a_r^2}{2\tau_t^2}} \right) \prod_{j\in\text{sec}(i)} e^{\frac{-s_j^2}{2\tau_t^2}} \end{aligned} \tag{5.62}$$

Substituting these expressions into (5.58) and cancelling the common terms yields:

$$\eta_i^t(s) = \frac{\sum_{r=1}^{K/2} a_r e^{\frac{-a_r^2}{2\tau_t^2}} \left[e^{\frac{s_i a_r}{\tau_t^2}} - e^{\frac{-s_i a_r}{\tau_t^2}} \right]}{\sum_{k \in \text{sec}(i)} \sum_{r=1}^{K/2} e^{\frac{-a_r^2}{2\tau_t^2}} \left[e^{\frac{s_k a_r}{\tau_t^2}} + e^{\frac{-s_k a_r}{\tau_t^2}} \right]}$$
$$= \frac{\sum_{r=1}^{K/2} a_r e^{\frac{-a_r^2}{2\tau_t^2}} \sinh\left(\frac{s_i a_r}{\tau_t^2}\right)}{\sum_{k \in \text{sec}(i)} \sum_{r=1}^{K/2} e^{\frac{-a_r^2}{2\tau_t^2}} \cosh\left(\frac{s_k a_r}{\tau_t^2}\right)} \tag{5.63}$$

For the special case where $K = 2$, there is only a single term in each summation and the exponential terms cancel:

$$\eta_i^t(s) = \frac{\sqrt{nP_\ell} \sinh\left(\frac{s_i \sqrt{nP_\ell}}{\tau_t^2}\right)}{\sum_{k \in \text{sec}(i)} \cosh\left(\frac{s_k \sqrt{nP_\ell}}{\tau_t^2}\right)} \tag{5.64}$$

Having found these expressions for η_i^t we now proceed to use them to derive the full AMP update rules for the modulated SPARCs.

5.4.2 Derivation of update rules

We start from the same message-passing algorithm as in the unmodulated case [1, (22) and (23)]:

$$z_{a \to i}^t = y_a - \sum_{j \in [ML] \setminus i} A_{aj} \beta_{j \to a}^t \tag{5.65}$$
$$\beta_{i \to a}^{t+1} = \eta_i^t(s_{i \to a}), \tag{5.66}$$

where $\eta_i^t(.)$ is the new estimator defined in (5.63), and the test statistic $s_{i \to a}^t$ is the same as in [1], namely that for $i \in \text{sec}_\ell$, $s_{i \to a}^t \in \mathbb{R}^M$ is defined as

$$(s^t_{i\to a})_i = \sum_{b\in[n]\setminus a} A_{bi} z^t_{b\to i} \tag{5.67}$$

$$(s^t_{i\to a})_j = \sum_{b\in[n]} A_{bj} z^t_{b\to j}, \quad j \in \mathrm{sec}_\ell \setminus i. \tag{5.68}$$

We set $z^t_{a\to i} = z^t_a + \delta z^t_{a\to i}$ and $\beta^{t+1}_{i\to a} = \beta^{t+1}_i + \delta\beta^{t+1}_{i\to a}$ and see that

$$z^t_a = y_a - \sum_{j\in[ML]} A_{aj}\beta^t_{j\to a} \tag{5.69}$$

$$\delta z^t_{a\to i} = A_{ai}\beta^t_{i\to a}. \tag{5.70}$$

To determine $\delta\beta^t_{i\to a}$ we expand η^t_i in a first order Taylor series around the argument $\left\{\sum_{b\in[n]} A_{bj} z^t_{b\to j}\right\}_{j\in\mathrm{sec}(i)}$.

We obtain

$$\begin{aligned}\beta^{t+1}_{i\to a} \approx & \eta^t_i\left(\left\{\sum_{b\in[n]} A_{bj} z^t_{b\to j}\right\}_{j\in\mathrm{sec}(i)}\right) \\ & - A_{ai} z^t_{a\to i}\partial\eta^t_i\left(\left\{\sum_{b\in[n]} A_{bj} z^t_{b\to j}\right\}_{j\in\mathrm{sec}(i)}\right),\end{aligned} \tag{5.71}$$

where $\partial_i\eta^t_i$ is the partial derivative of η^t_i from (5.63) with respect to the ith component of the argument:

$$\partial_i \eta_i = \frac{\partial}{\partial s_i} \eta_i(\{s_j\}_{j \in \text{sec}(i)}) = \eta_i \partial_i \ln \eta_i \tag{5.72}$$

$$= \eta_i \left(\partial_i \ln \left[\sum_{r=1}^{K/2} a_r e^{\frac{-a_r^2}{2\tau_t^2}} \sinh\left(\frac{s_i a_r}{\tau_t^2} \right) \right] - \partial_i \ln \left[\sum_{j \in \text{sec}(i)} \sum_{r=1}^{K/2} e^{\frac{-a_r^2}{2\tau_t^2}} \cosh\left(\frac{s_j a_r}{\tau_t^2} \right) \right] \right) \tag{5.73}$$

$$= \eta_i \left(\frac{\sum_{r=1}^{K/2} a_r e^{\frac{-a_r^2}{2\tau_t^2}} \frac{a_r}{\tau_t^2} \cosh\left(\frac{s_i a_r}{\tau_t^2} \right)}{\sum_{q=1}^{K/2} a_q e^{\frac{-a_q^2}{2\tau_t^2}} \sinh\left(\frac{s_i a_q}{\tau_t^2} \right)} - \frac{\sum_{r=1}^{K/2} e^{\frac{-a_r^2}{2\tau_t^2}} \frac{a_r}{\tau_t^2} \sinh\left(\frac{s_i a_r}{\tau_t^2} \right)}{\sum_{j \in \text{sec}(i)} \sum_{q=1}^{K/2} a_q e^{\frac{-a_q^2}{2\tau_t^2}} \cosh\left(\frac{s_i a_q}{\tau_t^2} \right)} \right) \tag{5.74}$$

$$= \frac{\eta_i}{\tau_t^2} \left(\frac{\sum_{r=1}^{K/2} a_r^2 e^{\frac{-a_r^2}{2\tau_t^2}} \cosh\left(\frac{s_i a_r}{\tau_t^2} \right)}{\sum_{q=1}^{K/2} a_q e^{\frac{-a_q^2}{2\tau_t^2}} \sinh\left(\frac{s_i a_q}{\tau_t^2} \right)} - \eta_i \right) \tag{5.75}$$

Using (5.75) in (5.71), and for brevity letting

$$s = \{s_j\}_{j \in \text{sec}(i)} = \left\{ \sum_{b \in [n]} A_{bj} z_{b \to j}^t \right\}_{j \in \text{sec}(i)}, \tag{5.76}$$

we obtain:

$$\beta_{i \to a}^{t+1} = \eta_i^t(s) - \frac{A_{ai} z_a^t}{\tau_t^2} \eta_i^t(s) \left(\frac{\sum_{r=1}^{K/2} a_r^2 e^{\frac{-a_r^2}{2\tau_t^2}} \cosh\left(\frac{s_i a_r}{\tau_t^2} \right)}{\sum_{q=1}^{K/2} a_q e^{\frac{-a_q^2}{2\tau_t^2}} \sinh\left(\frac{s_i a_q}{\tau_t^2} \right)} - \eta_i(s) \right). \tag{5.77}$$

We can write (5.77) as $\beta_i^{t+1} + \delta\beta_{i\to a}^{t+1}$, giving the update rule for β^t:

$$\beta_i^{t+1} = \eta_i^t(s), \tag{5.78}$$

and letting $\delta\beta_{i\to a}^{t+1}$ be the second term. Furthermore, from (5.69) and (5.70), we can split the $z_{b\to j}^t$ in v_i into

$$\begin{aligned} z_{b\to j}^t &= z_b^t + \delta z_{b\to j}^t \\ &= z_b^t + A_{bj}\beta_{j\to b}^t \\ &= z_b^t + A_{bj}\beta_j^t + A_{bj}\delta\beta_{j\to b}^t. \end{aligned} \tag{5.79}$$

and as $A_{bj}\delta\beta_{j\to b}^t = O(\log n/n)$, we neglect that term and write $\delta z_{b\to j}^t = A_{bj}\beta_j^t$, giving a new s:

$$\begin{aligned} s &= \left\{ \sum_{b\in[n]} A_{bj} z_b^t + A_{bj}^2 \beta_j^t \right\}_{j\in \text{sec}(i)} \\ &= \left\{ \left(A^* z^t + \beta^t\right)_j \right\}_{j\in \text{sec}(i)}, \end{aligned} \tag{5.80}$$

where we have used $\sum_{b\in[n]} A_{bj}^2 \to 1$ as $n \to \infty$. Combining (5.77) and (5.80) with (5.69) we obtain:

$$z_a^t = y_a - \sum_{k\in[ML]} A_{ak}\eta_k^{t-1}(s) - \frac{A_{ak}^2 z_a^{t-1}}{\tau_t^2}\eta_k^{t-1}(s) \left(\frac{\sum_{r=1}^{K/2} a_r^2 e^{\frac{-a_r^2}{2\tau_t^2}} \cosh\left(\frac{s_k a_r}{\tau_t^2}\right)}{\sum_{q=1}^{K/2} a_q e^{\frac{-a_q^2}{2\tau_t^2}} \sinh\left(\frac{s_k a_q}{\tau_t^2}\right)} - \eta_k(s) \right). \tag{5.81}$$

By expanding $\eta_k^{t-1}(s_k)$ we can cancel its numerator:

$$
\begin{aligned}
&\eta_k^{t-1}(s)\frac{\sum_{r=1}^{K/2} a_r^2 e^{\frac{-a_r^2}{2\tau_t^2}} \cosh\left(\frac{s_k a_r}{\tau_t^2}\right)}{\sum_{q=1}^{K/2} a_q e^{\frac{-a_q^2}{2\tau_t^2}} \sinh\left(\frac{s_k a_q}{\tau_t^2}\right)} \\
&= \frac{\sum_{r=1}^{K/2} a_r e^{\frac{-a_r^2}{2\tau_t^2}} \sinh\left(\frac{s_k a_r}{\tau_t^2}\right)}{\sum_{j\in \text{sec}(k)} \sum_{q=1}^{K/2} e^{\frac{-a_q^2}{2\tau_t^2}} \cosh\left(\frac{s_j a_q}{\tau_t^2}\right)} \cdot \frac{\sum_{r=1}^{K/2} a_r^2 e^{\frac{-a_r^2}{2\tau_t^2}} \cosh\left(\frac{s_k a_r}{\tau_t^2}\right)}{\sum_{q=1}^{K/2} a_q e^{\frac{-a_q^2}{2\tau_t^2}} \sinh\left(\frac{s_k a_q}{\tau_t^2}\right)} \\
&= \frac{\sum_{r=1}^{K/2} a_r^2 e^{\frac{-a_r^2}{2\tau_t^2}} \cosh\left(\frac{s_k a_r}{\tau_t^2}\right)}{\sum_{j\in \text{sec}(k)} \sum_{q=1}^{K/2} e^{\frac{-a_q^2}{2\tau_t^2}} \cosh\left(\frac{s_j a_q}{\tau_t^2}\right)}.
\end{aligned} \tag{5.82}
$$

Using (5.82) in (5.81):

$$
z_a^t = y_a - \sum_{k\in[ML]} A_{ak}\eta_k^{t-1}(s_k) - \frac{A_{ak}^2 z_a^{t-1}}{\tau_t^2}\left(\frac{\sum_{r=1}^{K/2} a_r^2 e^{\frac{-a_r^2}{2\tau_t^2}} \cosh\left(\frac{s_k a_r}{\tau_t^2}\right)}{\sum_{j\in \text{sec}(k)} \sum_{q=1}^{K/2} e^{\frac{-a_q^2}{2\tau_t^2}} \cosh\left(\frac{s_j a_q}{\tau_t^2}\right)} - \eta_k(s)^2\right). \tag{5.83}
$$

Finally, by noting that

$$
A_{ak}^2 \approx \frac{1}{n}, \text{and} \tag{5.84}
$$

$$
\sum_{k\in[ML]} \eta_k^t(s)^2 = \sum (\beta_k^t)^2 = \|\beta^t\|^2, \tag{5.85}
$$

and splitting the sum of $k \in [ML]$ into L sums of $k \in [M]$, we obtain:

$$
z_a^t = y_a - (A\beta^t)_a + \frac{z_a^{t-1}}{\tau_{t-1}^2 n}\left(\sum_{\ell\in[L]} \frac{\sum_{r=1}^{K/2} a_r^2 e^{\frac{-a_r^2}{2\tau_t^2}} \sum_{k\in \text{sec}_\ell} \cosh\left(\frac{s_k a_r}{\tau_t^2}\right)}{\sum_{q=1}^{K/2} e^{\frac{-a_q^2}{2\tau_t^2}} \sum_{j\in \text{sec}_\ell} \cosh\left(\frac{s_j a_q}{\tau_t^2}\right)} - \|\beta^t\|^2\right). \tag{5.86}
$$

For the special case of $K = 2$, where $a_1 = \sqrt{nP_\ell}$, the sums over r and q collapse to a

single term, permitting cancellation:

$$\begin{aligned} z_a^t &= y_a - (A\beta^t)_a + \frac{z_a^{t-1}}{\tau_{t-1}^2 n}\left(\sum_{\ell\in[L]} \frac{nP_\ell \sum_{k\in \text{sec}_\ell} \cosh\left(\frac{s_k\sqrt{nP_\ell}}{\tau_t^2}\right)}{\sum_{j\in \text{sec}_\ell} \cosh\left(\frac{s_j\sqrt{nP_\ell}}{\tau_t^2}\right)} - \|\beta^t\|^2\right) \\ &= y_a - (A\beta^t)_a + \frac{z_a^{t-1}}{\tau_{t-1}^2}\left(P - \frac{\|\beta^t\|^2}{n}\right) \end{aligned} \tag{5.87}$$

This is the z^t update rule for the $K = 2$ modulated case, and is also the same as the update rule for the unmodulated case. For larger values of K, computing z^t update requires additional calculations each iteration, but note that the cosh terms are already computed for finding η, so may be re-used in this calculation as well.

Chapter 6

SPARCs for Multiuser Channel and Source Coding Models

The previous chapters have considered SPARCs with AMP decoding for the point-to-point channel, where a single transmitter communicates to a single receiver. The same ideas can be extended to a range of more complicated channels, permitting practical implementations of capacity-achieving codes. Using SPARCs for multi-user models was first considered in [50], where it is shown that SPARCs under minimum distance decoding can achieve the capacity region for various multi-user AWGN models. We focus on practical implementations for SPARCs, using the various techniques discussed earlier in this thesis.

In this chapter, two such multi-user channels are studied first and in the most detail: the broadcast channel (BC) in Section 6.1, where a single transmitter communicates to multiple receivers; and the multiple access channel (MAC) in Section 6.2, where many transmitters communicate to a single receiver. In both cases extensions to the point-to-point encoding and decoding schemes permit operation on these multiuser channels. We consider implementation challenges and evaluate empirical performance.

SPARCs have also been considered for use as rate-distortion codes for source coding [32]. In this role the same code structure is used with different encoding algorithms to compress a source at some information rate, with a goal of achieving a certain distortion as a function of that rate. In addition to encoding a source into a single message, SPARCs can be used in a multiple description (MD) [51] role, whereby multiple messages are created from the same source. A receiver with any one message can reconstruct the source, but with more messages, a

higher quality reconstruction is possible. This has roles in situations where some messages may not reach the receiver, for example packet networks where packets may be lost. In Section 6.3 we implement such an encoder and evaluate its performance.

Source encoders using SPARCs can also be used in combination with the channel decoders discussed previously to design systems for channels such as dirty paper coding (DPC) [52, 53] and for source coding problems such as Wyner-Ziv distributed source coding (WZ) [54]. In DPC a transmitter with some knowledge of the channel state can adapt its encoding to best take advantage of this knowledge, improving the effective snr at the receiver. In WZ, we consider a source present at the encoder which is also present corrupted by noise at the receiver. The encoder must communicate some information to the receiver to help it obtain a lower-noise reconstruction of the source. In Section 6.4, we implement a Wyner-Ziv encoder and decoder using a combination of the source encoder and the channel decoder, and evaluate its performance.

A common challenge for all of the multiuser models discussed in this chapter is the choice of a suitable power allocation, as each channel has unique constraints on, and tensions between, the various possible choices. Typically a power allocation that is suitable for one user will be poor for the other, so finding a suitable optimum between these two can be challenging. This point is considered in detail for each channel.

6.1 Gaussian Broadcast Channel

The K-user Gaussian broadcast channel [26] has a single input sequence $X = \{x_j\}_{j\in[n]}$ and K output sequences, one for each user. The output sequence of user i is denoted by $Y_i = \{y_{ij}\}_{j\in[n]}$, where each output is an observation of the input corrupted by independent Gaussian white noise $Z_i = \{z_{ij}\}_{j\in[n]}$, $z_{ij} \sim \mathcal{N}(0, \sigma_i^2)$. This setup is illustrated in Figure 6.1 for two users. One practical example of this channel is a shared radio medium with a single transmitter communicating to multiple receivers. In the general case, the transmitter wishes to communicate distinct messages to each receiver — the medium is common, but the messages to be communicated are not.

We will consider the two-output scenario for simplicity, although the techniques presented here could be extended to an arbitrary number of receivers. Without loss of generality, we assume $\sigma_1^2 \leq \sigma_2^2$, i.e., the noise affecting the first user has a lower variance than the noise affecting the second user.

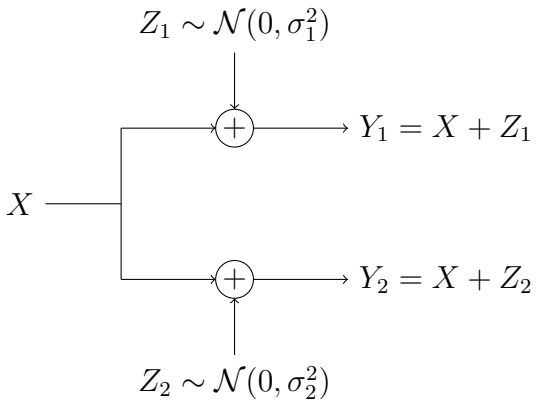


Figure 6.1: The Gaussian broadcast channel, shown with two receivers.

The channel outputs for an input X are:

$$Y_1 = X + Z_1, \tag{6.1}$$

$$Y_2 = X + Z_2. \tag{6.2}$$

The corresponding capacity region [26] is the union of all rate pairs (R_1, R_2) over $\alpha \in [0, 1]$ which satisfy

$$R_1 \leq \frac{1}{2} \log \left(1 + \frac{\alpha P}{\sigma_1^2} \right), \tag{6.3}$$

$$R_2 \leq \frac{1}{2} \log \left(1 + \frac{(1-\alpha)P}{\alpha P + \sigma_2^2} \right). \tag{6.4}$$

The transmitter sends X with average power constraint $\sum_{j=0}^{n} x_j^2 \leq P$, allocating αP of that power to the first receiver and $(1-\alpha)P$ to the second receiver, with $0 \leq \alpha \leq 1$ and $P_1 = \alpha P$, $P_2 = (1-\alpha)P$. The choice of α is arbitrary and permits a trade-off between the rates for each user, R_1 and R_2.

One scheme to communicate over this channel is time sharing, where the users are allocated time periods proportional to their share of the overall rate, and each communicates exclusively inside their time period. However, this scheme does not achieve the entire capacity region. Instead, it is possible to communicate to both receivers simultaneously while achieving the capacity region by using an appropriate code.

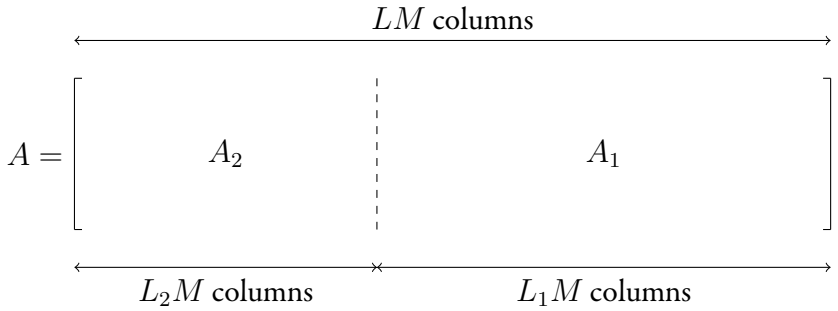


Figure 6.2: Division of the SPARC design matrix A for two users in the Gaussian broadcast channel. The second user is allocated the first L_2M columns (or L_2 sections, each of M columns), and the first user is allocated the remaining L_1M columns.

One theoretical coding scheme which will help elucidate the capacity bounds uses Shannon random codebooks, and is presented in [26]. At the transmitter, random codebooks are used to select codewords X_1 and X_2 for each receiver, and $X = X_1 + X_2$ is transmitted.

At receiver 2, the desired message X_2 is corrupted by the noise Z_2, but also suffers interference due to the other message X_1. The total adversarial power is thus $\sigma_2^2 + \alpha P$, which gives an effective snr of $\frac{(1-\alpha P)}{\sigma_2^2+\alpha P}$, leading to the constraint on R_2 seen in the capacity region. The receiver simply attempts to decode X_2 in the presence of this noise, as though it were operating on a point-to-point link with the same effective snr.

At receiver 1, since $\sigma_1^2 \leq \sigma_2^2$, the rate R_2 of X_2 will always be below this receiver's effective capacity, and so it is always possible to initially decode the second receiver's codeword. The receiver then subtracts it from Y_1 to leave just $X_1 + Z_1$, the desired codeword observed in noise of power σ_1^2, giving the capacity bound for R_1.

6.1.1 SPARCs for the Gaussian Broadcast Channel

We implement a coding scheme for the Gaussian broadcast channel using Sparse Regression Codes (SPARCs), as described in [50].

We view the combined message to be transmitted, X, as a single SPARC codeword, with the first L_2 sections allocated to the second receiver, and the remaining L_1 sections allocated to the first receiver, such that $L_1 + L_2 = L$. In effect this means each receiver has a separate SPARC design matrix A_1 and A_2, with $A = [A_2 A_1]$. This division is illustrated in Figure 6.2.

Instead of a single power allocation over all L sections, we use two separate power alloca-

tions, $\{P_{1,\ell}\}$ and $\{P_{2,\ell}\}$, for $\ell \in [L]$. Each receiver's power allocation is determined using that receiver's L_i, σ_i^2, R_i, and P_i, for $i \in \{1, 2\}$, in the same way as for a point-to-point SPARC, for example using the iterative allocation described in Chapter 3.

With this division of sections, we obtain the following rate equalities for each user:

$$nR_1 = L_1 \log M, \tag{6.5}$$

$$nR_2 = L_2 \log M. \tag{6.6}$$

Encoding

To encode, we take the message m_i for receiver i, consisting of L_i integers each between 1 and M, and encode it to a L_i-long β_i by setting the entries indexed by m_i in each section ℓ of β to $\sqrt{nP_{i,\ell}}$. We then form the SPARC codewords for each receiver as $X_i = A_i\beta_i$ and finally add them to obtain the combined codeword, $X = X_1 + X_2$.

In effect, the message for each user is encoded using their split of the design matrix A in exactly the same way as for the point-to-point setup.

X is transmitted and each receiver observes $Y_i = X + Z_i$.

Decoding

Receiver 2, with the higher noise variance, decodes by running a normal SPARC AMP decoding routine, using A_2 (the first L_2 columns in A) as the design matrix, attempting to recover X_2 and thus m_2. This is identical to the standard point-to-point setup.

Receiver 1 wishes to decode X_2 first, and then subsequently obtain its own message X_1. A simple approach would be to run two successive AMP decoders, first to extract X_2 exactly as per receiver 2, and then subtracting X_2 from Y and running a new AMP to decode X_1. However, this requires two separate AMP decoders, and may not perform as well as a single decoder pass, since only hard information on X_2 is exchanged between the two decoders.

Instead, we concatenate the two power allocations, forming an overall $\{P_\ell\}$ for $\ell \in [L]$, where

$$P_\ell = \begin{cases} P_{2,\ell} & \ell \leq L_2 \\ P_{1,\ell} & \ell > L_2, \end{cases} \tag{6.7}$$

and run the AMP decoder on the full design matrix A, with the combined power allocation $\{P_\ell\}$. The recovered $\hat{\beta}$ contains both messages, so after decoding completes we discard the first L_2 sections and keep only the latter L_1 sections, representing m_1. This method avoids having to actually find X_2 and subtract it from Y; instead the AMP decoder will find the entire message X.

6.1.2 Implementation and Results

The physical channel is specified in advance by P, σ_1^2, and σ_2^2. The SPARC design matrix is specified by a fixed M and n, and the power allocations for each user are designed in the same way as for a point-to-point channel. The operating point is set as follows: fix $\alpha \in [0, 1]$, which specifies the balance of power allocation between the two receivers. Then find $\mathcal{C}_1 = \frac{1}{2}\log(1 + \frac{\alpha P}{\sigma_1^2})$ and $\mathcal{C}_2 = \frac{1}{2}\log(1 + \frac{1-\alpha}{\alpha P + \sigma_2^2})$. Fix $\gamma \in [0, 1]$ and set $R_1 = \gamma\mathcal{C}_1$ and $R_2 = \gamma\mathcal{C}_2$. γ therefore determines the back-off from the boundary point $(\mathcal{C}_1, \mathcal{C}_2)$ of the capacity region. Finally set the remaining SPARC design matrix parameters $L_1 = nR_1/\log(M)$ and $L_2 = nR_2/\log(M)$.

The encoding and decoding operations are then performed as described above. Section error rate is measured separately for each receiver, recording the number of sections in error for just their own message after decoding finishes. We can then also investigate the worst-case section error rate for either receiver at any operating point. Figure 6.3 shows the performance for a setup with $P = 63$, $\sigma_1^2 = 1$, and $\sigma_2^2 = 2$. The resulting capacity region is shown on the charts. The fixed SPARC parameters were $M = 512$ and $n = 4095$, with L_1 and L_2 varying depending on the choice of α. The first and second charts show the bit error rate performance achieved when only considering the first and second receiver, while the third chart shows the worst case of those two. Additionally, a contour is plotted showing the boundaries of regions where the bit error rate is found to be below 1×10^{-3}.

When considering each receiver independently, we observe that bit error rates are reasonably good when that receiver's rate is above 0.5, which is in accordance with the results from point-to-point channels. However, because each receiver suffers badly once its rate goes below 0.5, the worst-case error is only better than 1×10^{-3} in a small number of cases, relatively far from the capacity boundary and only near equal power balance where $\alpha = 0.5$.

Poor performance at low rates has been observed in the simpler point-to-point case, but in this multiuser setup there is an additional factor contributing to the performance. For the experimental setup as described, M is fixed to the same value for both users. As we saw in

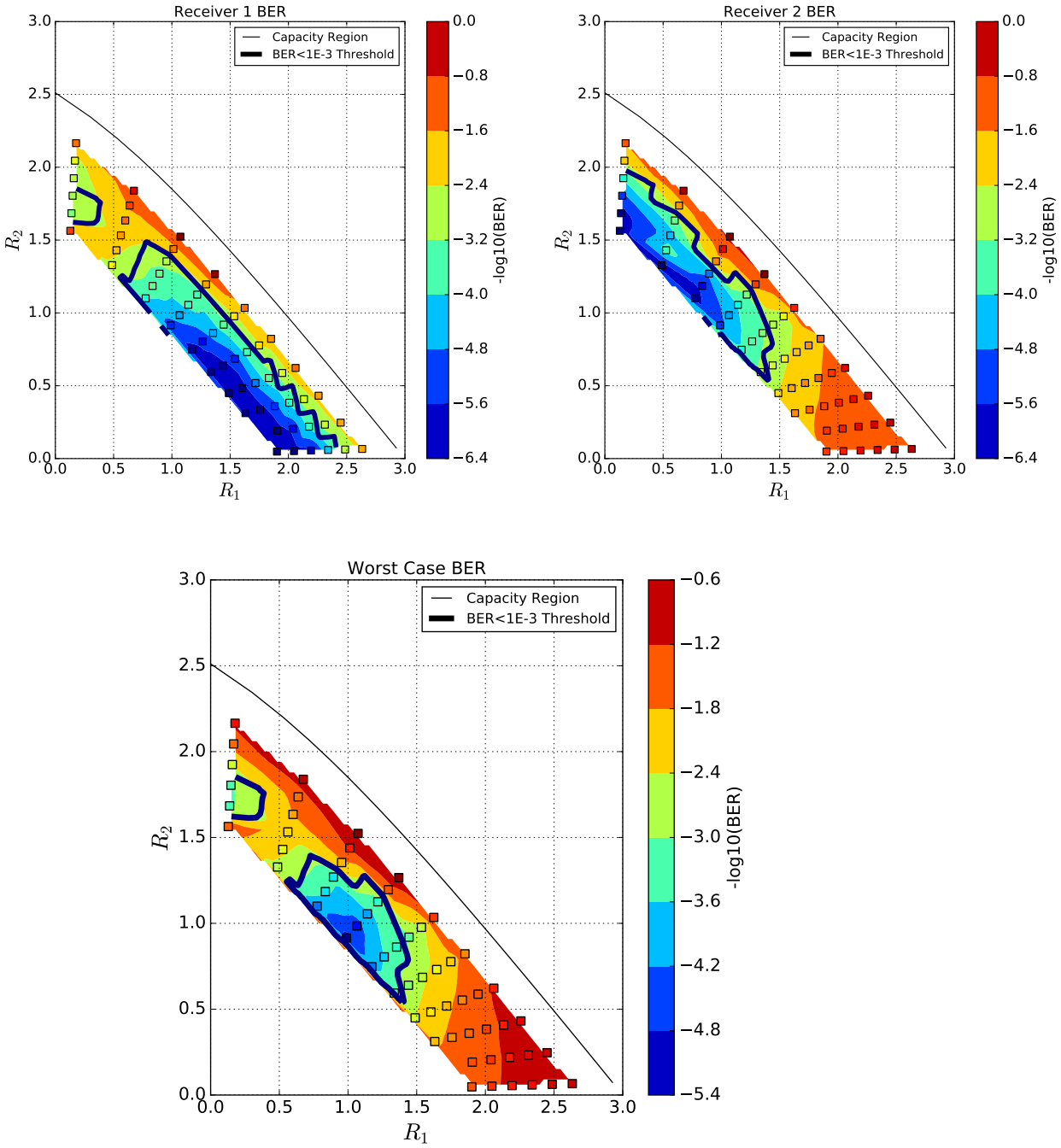


Figure 6.3: Bit error rates for broadcast channel simulations, with contour indicating the boundary of the regions where the error rate is below 1×10^{-3}. Each displayed point is the average of approximately 3000 trials. $P = 63, \sigma_1^2 = 1.0, \sigma_2^2 = 2.0, M = 512, n = 4095$.

Chapter 3, for a particular channel set-up, there is an optimal M, above and below which performance can rapidly decrease. Therefore, when the rates for the two receivers in the BC setup differ substantially, so too will the optimal M. The gap between each receiver's optimum M and the fixed value will lead to performance degradation, as observed. It is conceptually possible to run the AMP decoder with differing values for M in different sections, which would allow each receiver to operate on an optimal M, but this has not been explored.

6.2 Gaussian Multiple Access Channel

The K-user Gaussian multiple access channel [26] has K transmitters and a single receiver. The input sequence of user i is denoted $X_i = \{x_{ij}\}_{j\in[n]}$, while the output sequence is denoted $Y = \{y_j\}_{i\in[n]}$, which is an observation of the sum of all the inputs, corrupted by independent Gaussian white noise $Z = \{z_j\}_{j\in[n]}$, $z_j \sim \mathcal{N}(0, \sigma^2)$. This setup is illustrated in Figure 6.4. A practical example of this channel is a shared radio medium with multiple transmitters communicating simultaneously to a single receiver. The receiver must determine the message from each transmitter.

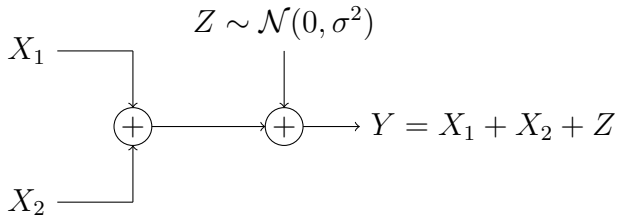


Figure 6.4: The Gaussian multiple access channel, shown with two transmitters.

We will consider the two-input scenario for simplicity, although the techniques presented here could be extended to an arbitrary number of transmitters.

The channel output for inputs X_1 and X_2 is

$$Y = X_1 + X_2 + Z \tag{6.8}$$

where

$$Z \sim \mathcal{N}(0, \sigma^2). \tag{6.9}$$

The two transmitters each send a codeword and the receiver observes the sum in noise.

Each transmitter $i \in 1, 2$ has an average power constraint $\sum_{j=0}^{n} x_{i,j} \leq P_i$. The corresponding capacity region is the set of rate pairs (R_1, R_2) which satisfy [26]

$$R_1 \leq \frac{1}{2} \log \left(1 + \frac{P_1}{\sigma^2} \right), \tag{6.10}$$

$$R_2 \leq \frac{1}{2} \log \left(1 + \frac{P_2}{\sigma^2} \right), \tag{6.11}$$

$$R = R_1 + R_2 \leq \frac{1}{2} \log \left(1 + \frac{P_1 + P_2}{\sigma^2} \right). \tag{6.12}$$

This capacity region can be seen as the combination of the two independent point-to-point capacity constraints with an additional constraint on the sum rate which is controlled by the sum power. This third constraint will tend to be the one which affects practical multi-user operating points, where no rates are close to zero.

As with the broadcast channel, one scheme to communicate over this channel is time sharing, where the users are each allocated time periods proportional to their share of the overall rate, and each user transmits exclusively within their time period. However, this scheme cannot achieve the full capacity region given above, as it is limited by each transmitter's point-to-point constraint. For example, if both transmitters had the same power $P_1 = P_2 = 15$, and shared time equally, they could each transmit $R_1 = R_2 = 0.5 \times \frac{1}{2} \log(1 + 15) = 1$ bit, but $R_1 = R_2 = 1.238$ would still be inside the capacity region and represents a higher possible throughput.

It is possible to achieve the full capacity region by using an appropriate code. Similarly to the broadcast channel, Shannon random codebooks present one method for achieving capacity, as described in [26]. Each user generates their own independent random codebook and transmits one codeword, and the receiver performs joint decoding. In effect, the receiver has a codebook containing all possible combinations of messages from the transmitters. This is viewed as a codebook with the sum rate $R = R_1 + R_2$, and since the total power from the two users is $P = P_1 + P_2$, we require that $R < \frac{1}{2} \log(1 + \frac{P}{\sigma^2})$ for successful decoding — which is precisely the sum constraint given above. However, this scheme is computational infeasible.

6.2.1 SPARCs for the Gaussian Multiple Access Channel

We implement a coding scheme for the Gaussian multiple access channel using SPARCs, as described in [50].

Each user has an independent SPARC design matrix and encodes one codeword from it in the usual way for point-to-point operation. Each user then transmits their codeword simultaneously. Since SPARC codewords are weighted sums of many columns from the design matrix, we can view the sum of two SPARC codewords as a single codeword from a wide matrix formed by combining the individual design matrices, and therefore decode it with the same AMP decoder developed for the point-to-point channel.

Given a rate R_i for transmitter i, $i \in \{1, 2\}$, and SPARC design parameters M and n, we can find the number of SPARC sections L_i required for each transmitter as $L_i = \frac{nR_i}{\log M}$. Each transmitter's design matrix A_i is of size $L_i M \times n$, together forming a larger $A = [A_1 A_2]$, with $L = L_1 + L_2$ sections in total. This is the same setup as for the broadcast channel, illustrated in Figure 6.2, with the order of the two users swapped.

Denote by $\{P_\ell\}_{\ell \in [L]}$ the overall power allocation, designed with $L = L_1 + L_2$ and $R = R_1 + R_2$. The allocation is designed using the same techniques described in Chapter 3, so will also be non-increasing, with a flat section of equal power at the end.

The power allocation over this design matrix is critical for good performance. As there is a single combined decoder, we wish to use a single power allocation, which can be designed in advance using the same iterative techniques as for point-to-point channels. However, it must also be partitioned between the two transmitters, which must each know their respective power allocations in advance.

This partitioning is challenging: transmitter i must be allocated precisely L_i sections, which must sum to no more than total power P_i. Additionally, we would like for errors to be fairly distributed between transmitters, so that each transmitter experiences approximately the same error rate. We know from Chapter 3 that the majority of errors occur in sections towards the end of the power allocation, where section power is lower and many sections share the same power (the *flat* region). Therefore we would like each user to have the same proportion of their allocation be flat. To summarise, the requirements for the power allocation are:

1. Of the $L = L_1 + L_2$ sections, we must allocate any L_1 sections to user 1, and the remaining L_2 to user 2.

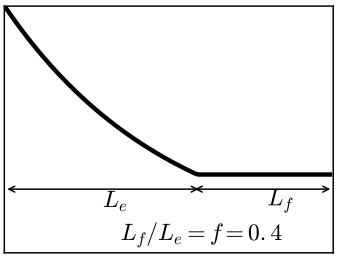
2. The sections allocated to user 1 must sum to no more than P_1, and those for user 2 to no more than P_2.

3. Once the previous conditions are met, choose the solution which divides any equal power sections equally between the two users.

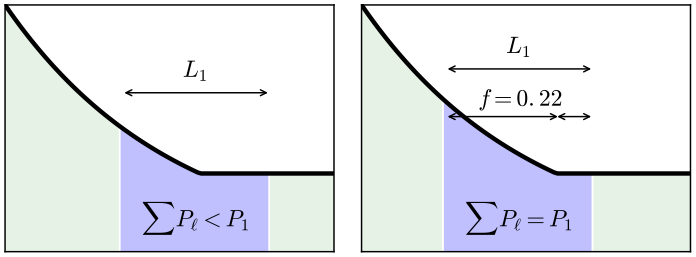
Our strategy for partitioning the power allocation is given in Algorithm 6.1. In brief, we locate a bracket of size either L_1 or L_2 sections inside $\{P_\ell\}_{\ell \in [L]}$ such that its sum is as close as possible to, without exceeding, P_1 or P_2 respectively, and allocate the coefficients within the bracket to transmitter 1 or transmitter 2 as appropriate, and the remaining sections on either side of the bracket are allocated to the other transmitter. The choice of bracket size (and thus of which transmitter is allocated the bracketed coefficients) is determined by which option gives the closest to optimal division of the coefficients from the flat section. This strategy is illustrated graphically in Figure 6.5.

Algorithm 6.1 MAC Power Allocation Partitioning Routine

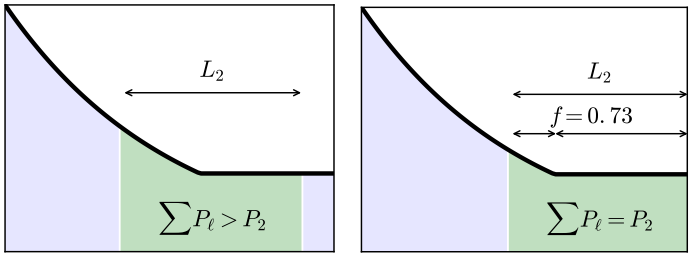
Require: A power allocation P_ℓ for $1 \leq \ell \leq L$ such that $\sum_{\ell=1}^{L} = P$
Require: A target power $P_{T_i} > 0$ for each user $i \in 1, 2$
Require: A required section count L_i for each user i where $0 < L_i < L$
  Find ℓ_f, the first ℓ where $P_\ell = P_{\ell+1}$, or set $\ell_f = L$ if no such ℓ exists
  for each user $i \in 1, 2$ **do**
    Make a bracket $B = [\ell_a, \ell_b]$ such that $\ell_f \in B$ and $\ell_b - \ell_a = L_{Ti}$
    Denote $P_B = \sum_{\ell \in B} P_\ell$ the power in the bracket
    While the gap from P_{Ti} exceeds than the change from by shifting the bracket by one:
    while $|P_B - P_{Ti}| > P_{\ell_a}$ **do**
      if $P_B > P_{Ti}$ **then**
        Power inside bracket is too large, so move bracket right:
        $B \leftarrow [\ell_a + 1, \ell_b + 1]$
      else
        Power inside bracket is too small, so move bracket left:
        $B \leftarrow [\ell_a - 1, \ell_b - 1]$
      end if
      if $\ell_a < 1$ or $\ell_b > L$ **then**
        The bracket is at the edge of the power allocation, so it is now impossible to include any more or less power while maintaining a contiguous bracket
        Abort search for this user, no bracket can be found
      end if
    end while
    End search for this user successfully.
  end for
  if only one bracket was found **then**
    Assign the user whose bracket we found the power allocation inside that bracket
    Assign the other user all remaining sections of the power allocation
  else if a bracket was found for both users **then**
    Select the bracket where the proportion of flat sections is closest to that of the original power allocation
    Assign that bracket to its corresponding user, and all remaining sections to the other user
  else
    No bracket could be found, cannot proceed for this set of parameters
  end if



(a) The overall power allocation to partition.
The ratio of exponential to flat sections is denoted f, here with $f = 0.4$.



(b) We first consider a bracket of size L_1, shown in blue. Our first attempt (on the left) contains too little power, so we move it leftwards until the contained power reaches P_1, shown on the right. At this position, $f = 0.22$.



(c) Next we consider a bracket of size L_2, shown in green. The first attempt contains too much power, so we move it rightwards until the contained power is P_2. In this position $f = 0.73$. Since the L_1 bracket has an f closer to that of the original allocation, we use that bracket for partitioning.

Figure 6.5: Example of the power allocation partitioning strategy. Sections highlighted in blue are considered for user 1, and those in green for user 2.

It is possible that no suitable partition can be found, for example where the smallest L_1 coefficients from the design $\{P_\ell\}_{\ell \in [L]}$ will sum to a power greater than P_1. In general this occurs only for cases very near the edge of the capacity region, for example where one transmitter has non-zero power but zero or close-to-zero rate. It would be possible to accommodate these scenarios by allowing M to vary for each transmitter; then the number of sections L_i required can

be varied to accommodate the power requirement (since L_i depends on M). In the extreme case, $M_i = 0$ would allow any number of sections to be allocated without consuming any rate. However, allowing M to vary per transmitter adds implementation complexity and has not been explored.

After partitioning the power allocation, encoding and decoding proceeds as follows:

1. Each transmitter has been assigned a set of power allocation coefficients, which are sorted in descending order, giving $P_{\ell,1}$ and $P_{\ell,2}$.

2. Each transmitter encodes their individual message using their power allocation and the respective A_i, in the same way as a point-to-point SPARC, obtaining x_i.

3. The channel observation is $y = x_1 + x_2 + z$, where $z \sim \mathcal{N}(0, \sigma^2)$.

4. The decoder uses the combined $A = [A_1 A_2]$ and a combined power allocation $[P_{\ell,1} P_{\ell,2}]$ and performs regular AMP decoding of y, giving $\hat{\beta}$.

5. After decoding, the first L_1 sections of $\hat{\beta}$ contain the message from the first transmitter, and the remaining L_2 sections contain the message from the second transmitter.

6.2.2 Implementation and Results

The physical channel is specified in advance by P_1, P_2, and σ^2. The SPARC design matrix is specified by a fixed M and n, with L_1 and L_2 found as follows. We find the sum-rate limit $\mathcal{C}_{\text{sum}} = \frac{1}{2}\log(1 + \frac{P_1+P_2}{\sigma^2})$. We fix $\gamma \in [0, 1]$ and then set $R_1 + R_2 = R = \gamma \mathcal{C}_{\text{sum}}$. We therefore have γ representing the backoff from the sum rate limit. Next, fix $\alpha \in [0, 1]$ which sets the share of this sum rate for each transmitter, $R_1 = \alpha R$ and $R_2 = (1 - \alpha)R$. Finally we find $L_1 = nR_1/\log(M)$ and $L_2 = nR_2/\log(M)$, and $L = L_1 + L_2$.

A power allocation is then designed using R and L, and partitioned according to the strategy described in Algorithm 6.1. Encoding and decoding is then performed as described above. After decoding, the number of sections decoded in error is counted separately for the first L_1 and then the last L_2 sections, and used to report section error rates for each transmitter. We also report the worst case section error rate between the two users. Since a design goal of the power allocation partition is that users experience similar error rates, the worst case performance should be similar to either user.

Figure 6.6 shows the resulting section error rates for one channel with $P_1 = 15, P_2 = 15, \sigma^2 = 1$. The capacity region for this channel is also shown.

As the experimental set-up is equivalent to a single point-to-point channel with the same sum power and sum rate and power allocation, we expect to obtain similar bit error performance to that scenario. The results support this, with good bit error rates even reasonably close to capacity at all points in the rate region. The worst case BER is also close to each individual user's BER, showing that the power allocation partition is generally successful at ensuring equal error rates between users.

No results are found for rate pairs where either rate is close to 0 due to the difficulties in finding a valid power allocation partition described above.

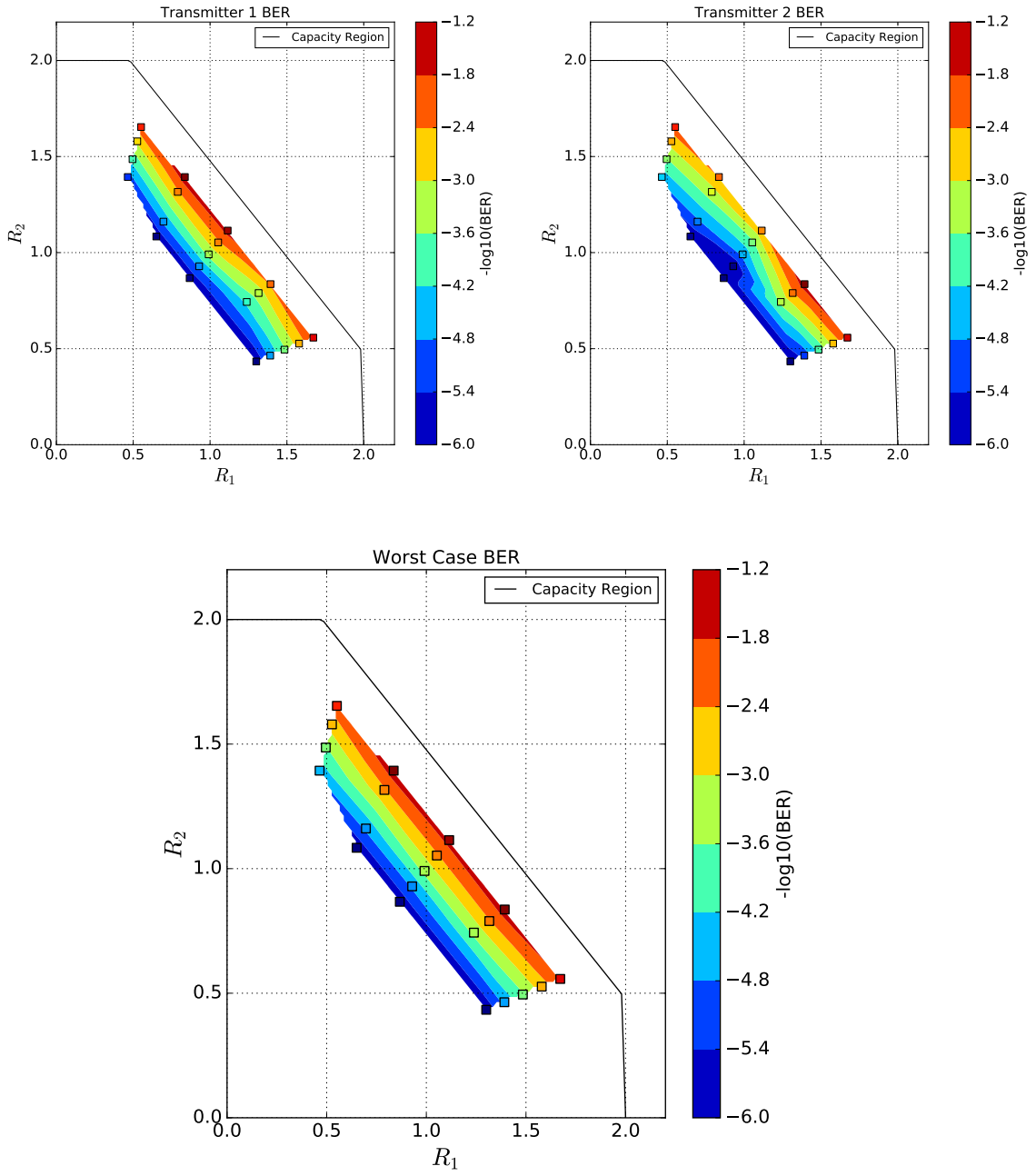


Figure 6.6: Bit error rates for multiple access channel simulations. Each displayed point is the average of approximately 30000 trials. $P_1 = 15, P_2 = 15, \sigma^2 = 1, M = 512, n = 4095$.

6.3 Multiple Description Source Coding

Source coding, introduced in Section 1.2, is the process of encoding a source vector to a codeword at a predetermined rate. In the simple point-to-point case there is a single encoder which, given a source sequence, selects a codeword. The decoder receives the codeword index and simply produces the codeword as its estimate the source sequence. In contrast to channel coding, the encoder is more complex than the decoder, since it must select the best codeword for a given source sequence from all possible codewords, while the decoder only reconstructs the source estimate using the provided codeword index. In fact, there is a formal duality relationship between point-to-point source coding and channel coding, where the source encoder is the dual of the channel decoder [55]. In addition to the point-to-point case, there are several multi-terminal scenarios which involve source coding. We will consider using SPARCs to construct efficient algorithms for two of these: multiple description source coding, and Wyner-Ziv distributed source coding.

The multiple description problem is a network information theory model where we wish to form multiple descriptions of a source such that if some subset of those descriptions is received, a target distortion for that subset is achieved at reconstruction. The problem was introduced by Witsenhausen in in [51] with subsequent results in [26, 56, 57]. For a practical example, if an audio signal was transmitted over a packet-switched communication network which may drop packets, we could imagine sending several descriptions of the audio, one per packet, and arrange that if only one description is received then a certain (high) level of distortion is realised, but for each additional packet received we recover a lower level of distortion. This provides graceful handling of missing packets, while maintaining a low distortion when all packets are received correctly. It is less wasteful than simply transmitting the same representation multiple times, where receiving multiple packets would be of no benefit.

In this model we have one encoder which generates k descriptions m_i, $i \in [k]$, of a Gaussian source $X \sim \mathcal{N}(0, \sigma_X^2)$, each at rate R_i (which may all be the same in the symmetric case). Some distinct subset of these descriptions reaches the receiver, which has effectively 2^k distinct decoders, one for each possible combination of received descriptions. The trivial case of receiving no descriptions is generally discounted (optimally we estimate all zeros and obtain a distortion equal to the source variance σ_X^2). We refer to the decoders that take a single description as the *side* decoders, and the decoder that takes every description as the *central* decoder. Decoders output $\hat{X}_j$ for $j \in [2^k - 1]$, their estimate of the source, which has an associated

distortion D_j. Figure 6.7 illustrates this for the $k = 2$ case.

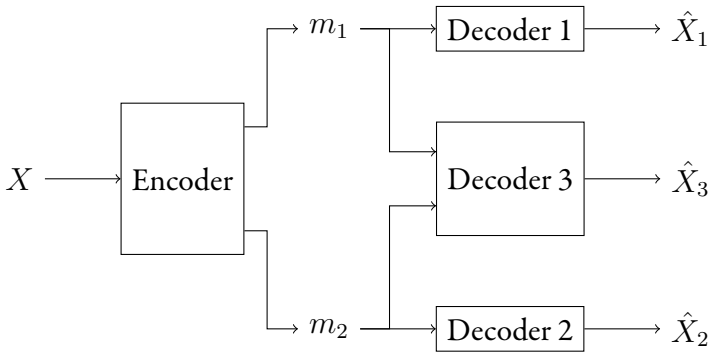


Figure 6.7: Multiple Description channel with $k = 2$. A source X is encoded into two messages m_i at corresponding rates R_i for $i \in \{1, 2\}$, and we obtain a different reconstruction $\hat{X}_j$ and corresponding distortion D_j for $j \in \{1, 2, 3\}$, depending on which messages are available at the receiver.

The achievable region is the set of rate-distortion tuples $(R_1, \ldots, R_k, D_1, \ldots, D_{2^k-1})$ which are achievable, i.e., the specified distortions can be met at the specified rates. This region is not known for the general case, but for the two-description problem with Gaussian sources and squared-error distortion metrics, the El Gamal-Cover achievable region [57] is optimal. This is called the quadratic Gaussian case, and here the rate-distortion tuple $(R_1, R_2, D_1, D_2, D_3)$ is achievable if and only if

$$\begin{aligned} R_1 &\geq \frac{1}{2} \log \frac{\sigma_X^2}{D_1}, \\ R_2 &\geq \frac{1}{2} \log \frac{\sigma_X^2}{D_2}, \\ R_s = R_1 + R_2 &\geq \frac{1}{2} \log \frac{\sigma_X^2}{D_3} + \frac{1}{2} \log \psi(D_1, D_2, D_3, \sigma_X^2), \end{aligned} \tag{6.13}$$

where

$$\psi(D_1, D_2, D_3, \sigma_X^2) = \begin{cases} 1, & D_3 < D_1 + D_2 - \sigma_X^2 \\ \frac{\sigma_X^2 D_3}{D_1 D_2}, & D_3 > \left(\frac{1}{D_1} + \frac{1}{D_2} - \frac{1}{\sigma_X^2}\right)^{-1} \\ \frac{(\sigma_X^2 - D_3)^2}{(\sigma_X^2 - D_3)^2 - \left(\sqrt{(\sigma_X^2 - D_1)(\sigma_X^2 - D_2)} - \sqrt{(D_1 - D_3)(D_2 - D_3)}\right)^2}, & \text{otherwise.} \end{cases} \tag{6.14}$$

The cases $D_3 < D_1 + D_2 - \sigma_X^2$ and $D_3 > (\frac{1}{D_1} + \frac{1}{D_2} - \frac{1}{\sigma_X^2})^{-1}$ are degenerate, as in either case there exists an achievable rate-distortion tuple with lower distortions at the same rates [58]. We therefore only consider the third case, with $(\frac{1}{D_1} + \frac{1}{D_2} - \frac{1}{\sigma_X^2})^{-1} \geq D_3 \geq D_1 + D_2 - \sigma_X^2$.

Note that compared to the rate-distortion limit for the point-to-point case, our sum rate requires some excess rate $\frac{1}{2}\log\psi(D_1, D_2, D_3, \sigma_X^2)$ beyond the rate required to achieve D_3 normally. This excess rate may be split between R_1 and R_2, or we may allocate the entire excess to either R_1 or R_2.

To encode in this two-description case we need to form two descriptions that meet a certain correlation structure. The two descriptions should be individually good, and therefore highly correlated with the original source, but must also refine each other, and therefore have low correlation with each other. These two requirements are in conflict and balancing them is key to a successful multiple description implementation.

6.3.1 Gram-Schmidt based Multiple Descriptions

In [58] it is shown that a construction based on Gram-Schmidt orthogonalisation can achieve the distortion region for the two-description quadratic Gaussian case. The key feature of this construction is that it achieves the multiple description rate-distortion region using separate *point-to-point* source encoders, without requiring a joint encoder. It is therefore amendable to implementation using the point-to-point SPARC source encoder of [32]. While the methodology discussed in [58] centres around the use of entropy coded dithered lattices for quantisation (ECDQ), the underlying technique is applicable to a range of quantisers, including the SPARC source encoder. We will focus only on applying this methodology to SPARCs.

One method to find suitable descriptions with the desired correlation properties is using Shannon-style random codebooks in a successive quantisation scheme. This is the scheme used originally in [57]. Two random codebooks are generated, the first large enough to quantise the source with the required D_1 using rate R_1, as in the normal source coding setup. The second codebook must be much larger, to encompass not only the source space but instead the product of the source and first codebook space, so that a second codeword U_2 may be found which is jointly typical with both the source and the first codeword U_1. This much larger second codebook is an impediment to practical multiple description. In [58] it is demonstrated that if some $f(X, U_1) = V$ can be found such that $(X, U_1) \to V \to U_2$ form a Markov chain, then

the second encoder need only cover the V-space, as any U_2 jointly typical with V will likely be jointly typical with (X, U_1). This observation is key to the application of the Gram-Schmidt orthogonalisation argument.

The main result of [58] is then as follows. First, with a similar construction to [51] and [26], we represent the quantised source descriptions U_1 and U_2 as

$$U_1 = X + T_0 + T_1, \tag{6.15}$$
$$U_2 = X + T_0 + T_2, \tag{6.16}$$

where T_0, T_1, T_2 are jointly Gaussian random variables representing the quantisation noise, with a specific correlation structure as follows. We require that

$$\mathbb{E}[T_1 T_2] = -\sigma_{T_1}\sigma_{T_2}, \tag{6.17}$$

and that X, T_0, (T_1, T_2) are all independent.

Given U_1 and U_2, the decoding functions are then

$$\hat{X}_1 = \mathbb{E}[X \mid U_1] = \alpha_1 U_1, \tag{6.18}$$
$$\hat{X}_2 = \mathbb{E}[X \mid U_2] = \alpha_2 U_2, \tag{6.19}$$
$$\hat{X}_3 = \mathbb{E}[X \mid U_1, U_2] = \beta_1 U_1 + \beta_2 U_2, \tag{6.20}$$

with the coefficients α_i and β_i defined as

$$\alpha_i = \frac{\sigma_X^2}{\sigma_X^2 + \sigma_{T_0}^2 + \sigma_{T_i}^2}, \quad i \in 1, 2, \tag{6.21}$$
$$\beta_1 = \frac{\sigma_X^2 \sigma_{T_2}}{(\sigma_{T_1} + \sigma_{T_2})(\sigma_X^2 + \sigma_{T_0}^2)}, \tag{6.22}$$
$$\beta_2 = \frac{\sigma_X^2 \sigma_{T_1}}{(\sigma_{T_1} + \sigma_{T_2})(\sigma_X^2 + \sigma_{T_0}^2)}, \tag{6.23}$$

where

$$\sigma_{T_0}^2 = \frac{D_3\sigma_X^2}{\sigma_X^2 - D_3}, \tag{6.24}$$

$$\sigma_{T_i}^2 = \frac{D_i\sigma_X^2}{\sigma_X^2 - D_i}, \quad i \in 1, 2. \tag{6.25}$$

The question is then how to obtain codewords jointly distributed according to the joint distribution of (U_1, U_2). In [58], they are represented using basis vectors from a Gram-Schmidt orthogonalisation process on the source.

The basis vectors are constructed as

$$B_1 = X, \tag{6.26}$$

$$B_2 = U_1 - E(U_1|X) = U_1 - X, \tag{6.27}$$

$$B_3 = U_2 - E(U_2|X, U_1) = U_2 - (a_1X + a_2U_1), \tag{6.28}$$

where

$$a_1 = \frac{\sigma_{T_1}^2 + \sigma_{T_1}\sigma_{T_2}}{\sigma_{T_0}^2 + \sigma_{T_1}^2}, \tag{6.29}$$

$$a_2 = \frac{\sigma_{T_0}^2 - \sigma_{T_1}\sigma_{T_2}}{\sigma_{T_0}^2 + \sigma_{T_1}^2}. \tag{6.30}$$

Rearranging,

$$U_1 = X + B_2, \tag{6.31}$$

$$U_2 = (a_1X + a_2U_1) + B_3, \tag{6.32}$$

suggests the interpretation that B_2 and B_3 are quantisation noises, with U_1 and U_2 formed by successively quantising X and $(a_1X + a_2U_1)$ respectively. Finally we can see that $a_1X + a_2U_1$ is a suitable V, since $U_2 = \mathbb{E}[U_2|X, U_1] + B_3$, with B_3 independent of (X, U_1), therefore $(X, U_1) \to \mathbb{E}[U_2|X, U_1] \to U_2$ form a Markov chain. It is then possible to demonstrate that this setup provides the required correlation structure between X, U_1 and U_2 [58].

Having found this relationship, the basic quantisation algorithm is straightforward:

1. The first description is obtained by encoding X to U_1 at rate R_1.
2. The second description is obtained by encoding $a_1X + a_2U_1$ to U_2 at rate R_2.
3. Decoding to $\hat{X}_i$ is per (6.20).

6.3.2 SPARCs for Multiple Description

We may use the SPARC successive cancellation source encoding scheme described in [32] to perform the two quantisations described above. We set the first side rate R_1 to the rate-distortion function result

$$R_1 = \frac{1}{2}\log\left(\frac{\sigma_X^2}{D_1}\right). \tag{6.33}$$

The sum rate is set to

$$R_s = \frac{1}{2}\log\left(\frac{\sigma_X^2}{D_3}\right) + \frac{1}{2}\log\psi(D_1, D_2, D_3, \sigma_X^2), \tag{6.34}$$

and all the required excess rate is then assigned to the second side encoder,

$$R_2 = R_s - R_1. \tag{6.35}$$

It is possible to allocate this excess rate to either side encoder or to split it between the two, though splitting is more complex. For these simulations, only allocating to a single side encoder is considered.

To determine the SPARC parameters, the section size M and the number of sections L_1 for the first encoder is specified, and from this the n parameter (which must be common to both encoders as it is the size of the source block) is determined as $n = \frac{L_1 \log M}{R_1}$. Finally $L_2 = \frac{\log M}{nR_2}$.

Figure 6.8 shows the results obtained from this implementation. Here the target single-message distortions D_1^* and D_2^* are both set to 0.25, and the target central distortion D_3^* is varied between 0.001 and 0.14. Above $D_3^* = 0.14$, the rate-distortion tuple becomes degenerate as no excess rate is required for the central reconstruction, so there is no benefit to targeting a higher D_3^*.

We see that the side distortions D_1 and D_2 are reasonably close to their target $D_1^* = D_2^* = 0.25$ and do not change significantly as D_3^* is changed, despite the excess rate being allocated

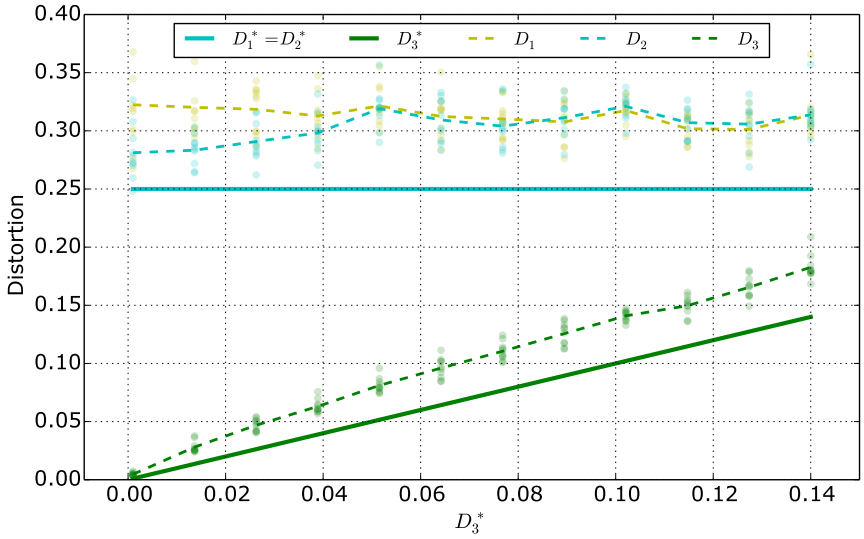


Figure 6.8: Distortions obtained from 2-description MD using the SPARC source encoder, with demanded D_1^* and D_2^* of 0.25 and demanded D_3^* being varied

to the second encoding increasing. This suggests the desired correlation structure is achieved in practice. Furthermore, the central distortion D_3 tracks the target D_3^* well. These results are encouraging: the SPARC implementation efficiently produces the two codewords, and the excess rate assigned to one of them is successfully used to give a lower central distortion which comes close to the theoretical limits.

6.4 Wyner-Ziv Distributed Source Coding

Wyner-Ziv distributed source coding, introduced by Wyner and Ziv in [54] and also known as source coding with decoder side-information, considers a model where the source $X \sim \mathcal{N}(0, \sigma^2)$ must be compressed with distortion D, and in addition to the nR bits of information from the encoder, the decoder also has available side-information in the form of a noisy observation $Y = X + Z$ where $Z \sim \mathcal{N}(0, N)$ is independent of X. If Y were available to the encoder, a very simple optimal strategy is for the encoder to simply encode the noise $Z = Y - X$, which the decoder can then subtract from Y. This would require a minimum rate of $\frac{1}{2}\log(\frac{\mathrm{Var}(X|Y)}{D})$. It is shown in [54] that this rate is achievable even when Y is only available at the decoder and not at the encoder.

The coding strategy from [54] is as follows. First define an auxiliary random variable $U = X + V$, where $V \sim \mathcal{N}(0, Q)$ is also independent of X. We view this as a test channel, with U the distribution of a codeword produced by the source encoder and V is the distribution of the corresponding quantisation noise. Using Shannon-style random codebooks, we generate at least $2^{nI(U;X)}$ codewords i.i.d. according to $P(U)$, ensuring that the encoder will be able to find a codeword with quantisation variance close to Q.

Then, instead of transmitting the index corresponding to the codeword directly to the encoder, we divide the codebook into 2^{nR} ***bins***, and send only the index of the bin that contained the codeword. This requires nR bits be communicated to the decoder, lower than the $nI(U;X)$ that would be required to transmit the index of the codeword in its entirety.

The decoder must recover the index of the codeword using the bin index from the encoder and the side-information Y. We use the bin index to construct a codebook consisting of just that bin, which must contain the codeword. This is then a channel decoding problem with $Y = X + Z = U + V + Z$, so we are observing the codeword in noise of power $N + Q$. As long as the number of codewords is exponentially less than $2^{nI(U;Y)}$ then decoding should succeed. Combining this requirement with the minimum size of the source codebook, we find that the number of bins must satisfy

$$2^{nR} > \frac{2^{nI(U;X)}}{2^{nI(U;Y)}}. \tag{6.36}$$

Finally, after recovering the codeword, the decoder then finds $\hat{X}$ as an MMSE estimate of X given both the recovered codeword and Y.

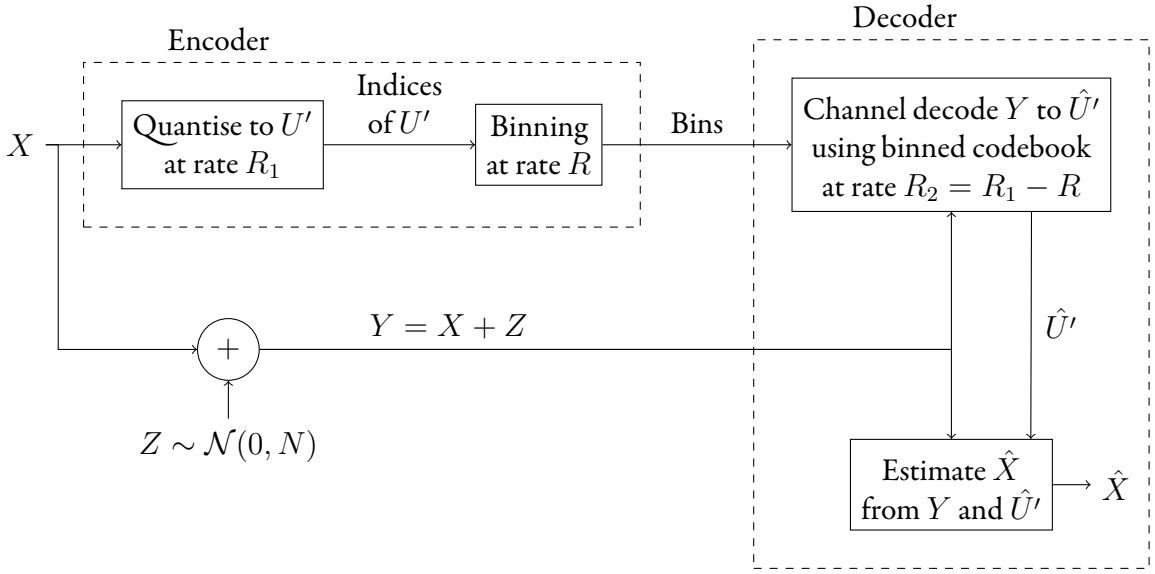


Figure 6.9: The Wyner-Ziv distributed source coding model. The source $X \sim \mathcal{N}(0, \sigma^2)$ is first quantised to a codeword, then only the bin indices for that codeword are transmitted at rate R to the decoder. The decoder observes $Y = X + Z$ and performs channel decoding using the codebook corresponding to the received bin indices. With Shannon random codebooks there is only a single bin; with SPARCs there are L bins, one per section.

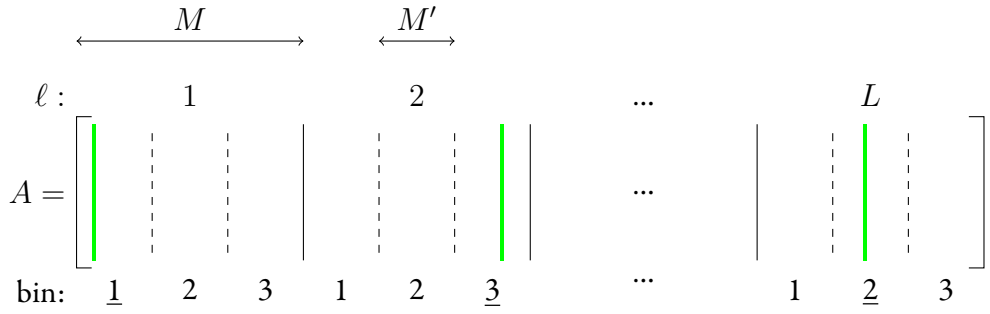


Figure 6.10: SPARC binning. The columns in A corresponding to the non-zero entries in β are highlighted. Instead of transmitting their indices directly, which requires $L \log M$ bits, we partition each section into bins of size M' columns, and transmit only the bin indices (underlined) in $L \log \frac{M}{M'}$ bits. The decoder then forms a new design matrix A' from only the selected bins, which will still contain the selected columns.

6.4.1 SPARCs for Wyner-Ziv Distributed Coding

The scheme described above can be implemented using SPARCs, as described in [50].

At the encoder, we perform a regular SPARC source encoding at rate R_1 to find a SPARC codeword U' close to the source X. By reversing the test channel described above, we can write $X = U' + V'$, where $V' \sim \mathcal{N}(0, \frac{\sigma^2 Q}{\sigma^2+Q})$. Instead of transmitting the nR_1 bits of the positions of the selected columns in the SPARC design matrix A which represent this codeword, requiring a high overall rate, we will split each section of A into bins of M' columns each, and transmit just the indices of the bins which contain the codeword's columns. This process is called *binning*, and requires only $nR = L\log(\frac{M}{M'})$ bits to communicate the bins. This concept is illustrated in Figure 6.10.

At the decoder, we can reconstruct a smaller codebook A' from the bins selected at the encoder. We also observe $Y = X + Z$, a noisy version of X. We then run the SPARC channel decoder on the design matrix formed only from the received bins, which attempts to decode X from Y as though X were itself a SPARC codeword. We know that $U' = X + V'$ is a codeword, so we are attempting to decode U' in noise of $V' + Z$. This gives us an snr of

$$\mathsf{snr} = \frac{\sigma^4}{\sigma^2 Q + (\sigma^2 + Q)N}, \tag{6.37}$$

where $Q = \frac{\mathrm{Var}[X|Y]D}{\mathrm{Var}[X|Y]-D}$. So long as $R_2 < \frac{1}{2}(1+\mathsf{snr})$ we expect decoding to succeed, producing $\hat{U}' = U'$. Finally we estimate $\hat{X}$ as a weighted linear combination of U' and Y,

$$\hat{X} = \left(\frac{1}{Q} + \frac{1}{\sigma^2} + \frac{1}{N}\right)^{-1} \left(\frac{\hat{U}'}{Q} + \frac{Y}{N}\right). \tag{6.38}$$

We can tolerate relatively large section error rates from the channel decoder, assuming errors predominantly occur in sections with low allocated power, as those sections will have small contributions to the distortion in the final estimate $\hat{X}$.

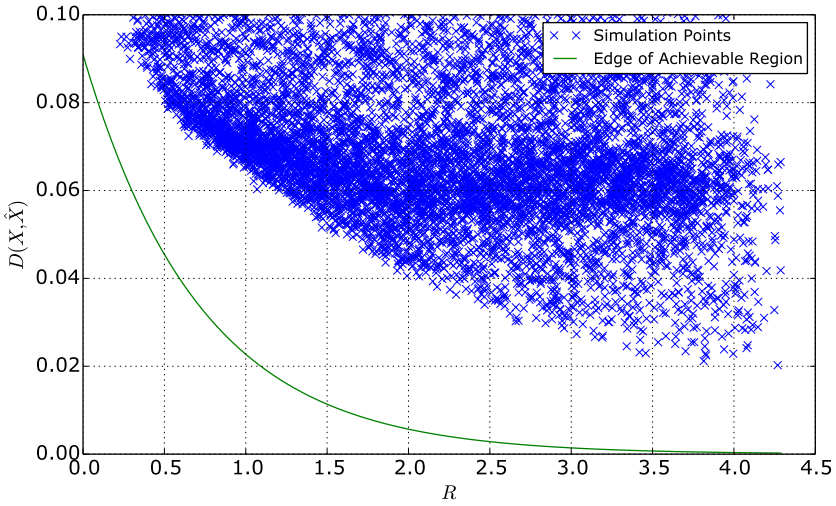


Figure 6.11: Empirical results for SPARC Wyner-Ziv coding. Here $\sigma^2 = 1$ and $N = 0.1$. The green line indicates the achievable region, while each blue mark is a single trial at different rates R and target distortions D. There is a distinct gap from the achievable region, visible between the lowest-distortion blue marks and the green line.

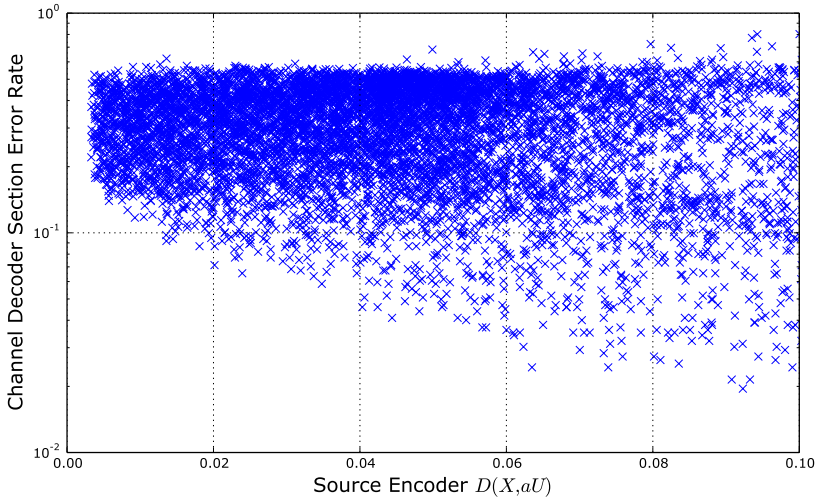


Figure 6.12: The results from Figure 6.11, plotted as the channel decoder section error rate against the source encoder distortion. Results with good source distortion tend to have poor channel error rates, and vice versa. Ideally all points would be concentrated in the lower-left, representing good section error rates *and* good source encoder distortion.

Figure 6.11 presents the results obtained from implementing the setup described above, using the SPARC source encoder from [50] and the channel decoder from Chapter 2. The green line indicates the boundary of the achievable region, and each blue mark represents the distortion obtained at a single trial. For each trial, a random target D was selected below $\mathrm{Var}[X \mid Y]$, a random R_2 was selected below $\frac{1}{2}\log(1 + \mathsf{snr})$, and the power allocation parameters $(a, f) \in (0, 1)$ were chosen randomly.

There is some gap from the achievable region. We find that for trials where the source encoder performs well, the channel decoder often has high error rates, and vice versa. This is illustrated in Figure 6.12, which plots the channel decoder section error rate against the source encoder distortion. Ideally, all points would cluster in the lower left, where we obtain good performance from both algorithms. Instead, points with good section error rate had generally poor source distortion, or vice versa.

The main reason for this conflict appears to be the choice of power allocation. Power allocations which performed well for source encoding are steeper exponentials, similar to the optimal source encoding power allocation from [32]. However, we have already seen in Chapter 3 that these allocations do not perform well for the channel decoder. Finding a good power allocation which balances the needs of both source and channel decoders is an open question.

Chapter 7

Conclusions

Contributions

The overall aim of this research was to develop novel techniques and algorithms for the use of sparse regression codes, with a focus on practical and efficient implementations in single- and multi-user settings. In Chapters 3 and 4, several improvements to the previous state of the art have been made concerning the practical implementation of the approximate message passing channel decoder for SPARCs, including

- The use of Hadamard-based design matrices to provide much lower computational complexity in the decoder, reducing complexity from $O(LMn)$ to $O(Ln \log n)$,
- Improved power allocation designs which yield error rates often orders of magnitude lower than previously,
- Online computation of τ_t^2, yielding slightly improved performance but crucially offering an early-termination criteria which can reduce decoding time, and
- A novel three-stage decoder combining an inner SPARC with an outer LDPC code to provide a sharp waterfall in bit error rates and very low overall codeword error rates.

Taken together, these improvements give the AMP decoder state of the art error correcting performance which is competitive with modern LDPC-based coded modulation schemes, while dramatically decreasing the decoder time and memory complexity, compared to previous SPARC implementations, to levels that are practical to implement.

In Chapter 5, the SPARC construction was extended by allowing the non-zero value in each section to be chosen from a fixed constellation. An AMP decoder for this new construction was developed and analysed. It was shown that with this decoder, binary modulated SPARCs provably achieve the channel capacity, with empirical performance that is near-identical while halving the size of the design matrix.

In Chapter 6, designs for SPARCs on multi-user channels which were previously only theoretical were efficiently implemented and simulated, obtaining good empirical performance on the broadcast and multiple access channels. New algorithms to find power allocations for these challenging scenarios were proposed and gave good results. We further implemented SPARCs for the multiple description and Wyner-Ziv source coding problems, obtaining good empirical performance with multiple descriptions and investigating the challenges involved in combining source and channel algorithms in the Wyner-Ziv scenario.

In addition to these advancements in SPARCs and the AMP decoder, this research has produced helpful heuristics into the behaviour of the AMP decoder under various conditions. These include the effect of the SPARC design parameters on the error concentration, the understanding of which is critical to obtaining good section error rate performance and was previously unexplored.

Limitations

There are a few design issues which remain unresolved. Given a power allocation or a choice of L, M, it is still not clear if one can theoretically predict how well concentrated the empirical section error rate will be around the state evolution prediction. Not having a theoretical prediction not only hinders predicting performance for a specific scenario, but is an impediment in designing a power allocation. The final iterative power allocation algorithm proposed in this thesis therefore still requires numerical optimisation of one parameter, via full AMP decoder simulation. Similarly, obtaining the optimal SPARC design parameters for a given setup requires extensive simulation with only heuristic guidance.

The AMP decoder does not currently perform well at low rates $R < 1$, where the best known power allocation becomes completely flat. There might be superior power allocations or other tweaks to the decoder algorithm to improve performance in this regime, which would also beneficial to the multi-user scenarios where one user has a low rate.

Time and resource limits prevented further simulation of the modulated SPARC and multi-user SPARC scenarios, and so the simulation results are limited in their applicability. While

they provide evidence of good performance in the specific setup considered for simulation, more investigation is needed to determine how well the SPARC performs over a wider range of channel parameters.

Future Work

This research leaves several open ends which would benefit from further investigation. Addressing the limitations discussed above is of obvious benefit, with the ability to accurately predict concentration-based results being especially useful for practical SPARC AMP decoders. Further directions include:

- The Hadamard based design matrices are very efficient, but could be taken one step further: instead of using $n \times n$ sized Hadamard matrices for each section of A, it would be possible to use $\lceil \frac{n}{M} \rceil$ Hadamard matrices of size $M \times M$, which would provide a further efficiency improvement.

- SPARCs have been considered for use in a spatially coupled framework [31, 39, 46], where the design matrix is modified to provide coupling between blocks, which can improve decoder performance. While this is interesting in its own right, recent unpublished work by K. Hsieh suggests that combining spatial coupling with the power allocations discussed in Chapter 3 yields results superior to either technique alone.

- Many interesting questions remain open for the modulated SPARCs, especially in regards to $K > 2$, where the number of possible modulation values increases. Does the modified AMP achieve the channel capacity for larger values of K? This could be proven by extending the existing proof where $K = 2$ along similar lines. Increasing K allows M to decrease while holding everything else constant: what is the optimal tradeoff between M and K, for either lowest computational complexity or best error performance?

- Another interesting direction for modulated SPARCs is to consider the use of a complex modulation pattern, along the lines of traditional QAM and PSK modulations. The currently real-valued non-zero values in β would be extended to allow complex values. This would require an update to the AMP decoder, but may be worthwile as it provides a natural extension of SPARCs to complex channels, which are in widespread use in modern radio systems. PSK especially would allow equal power for each section, independent of the modulation symbol chosen, while still permitting arbitrarily high K.

- There also remain open questions for SPARCs on the multiuser channels. We have covered the broadcast and multiple access channel for two users, but they extend naturally to many users, and it would be interesting to evaluate how well SPARCs perform as the number of users increases. Finding good power allocations for more than two users may be a particular challenge.

- Furthermore there may be applications of multiuser SPARCs to MIMO channels, where multiple transmit and/or receive antennas are used to improve channel capacity. Beyond treating each antenna path as a user in a multiuser channel, there may be further applications of the underlying sparse estimation inherent in SPARCs to better infer MIMO channel characteristics or perform channel coding.

- There is currently no proof that SPARCs can achieve the multiple description rate-distortion region. It is likely that a proof could be found along similar lines to the one given in [58] for ECDQ, by showing that the SPARC source encoder output follows the required distribution for both the first and second quantisations.

- For Wyner-Ziv source coding, the power allocation required tradeoffs between the optimal choice for the source encoder and channel decoder. Further work on novel power allocations for this model could yield much improved results.

- The empirical performance on both multiple descriptions and Wyner-Ziv seems to be limited by the source encoder performance: perhaps we could find a different source encoder, such as a modified AMP algorithm, which would yield better performance. Some tentative work on using the AMP for source encoding resulted in poor empirical performance. It appears that since there are many good solutions to the source coding problem, the AMP is less able to select a single good solution. There may be similarities to using low-density generator matrices for source encoding [59–61], where decimation and other techniques are used.

- The dirty paper coding problem [53] is a dual to the Wyner-Ziv channel and SPARCs may be expected to perform well there too. A design for using SPARCs on this channel was given in [50], but work would be required to design suitable power allocations and to implement the algorithms.

- All of the numerical results presented in this thesis are the outcome of computer simulations with software decoders. It would be interesting to implement many of these ideas in a practical decoder designed for operating on physical channels. There would likely be many new challenges, including operating the decoder at line rate, recovering the symbol clock from the received data (which does not have well defined transitions unlike the coded modulation setup), and handling uncertain channel gain. There would likely also be opportunities for further optimisations in the decoder when implemented in hardware, including efficient parallel hardware FWHT. Some work on hardware implementations of the original AMP decoder for SPARCs has been published in [62,63].

Bibliography

[1] C. Rush, A. Greig, and R. Venkataramanan, "Capacity-achieving sparse superposition codes via approximate message passing decoding," *IEEE Transactions on Information Theory*, vol. 63, no. 3, pp. 1476–1500, March 2017.

[2] A. Greig and R. Venkataramanan, "Techniques for improving the finite length performance of sparse superposition codes," *IEEE Transactions on Communications*, vol. 66, no. 3, pp. 905–917, March 2018.

[3] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, Jul. 1948.

[4] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley, 1991.

[5] A. Feinstein, *A new basic theorem of information theory*. Research Laboratory of Electronics, Massachusetts Institute of Technology, 1954, no. 282.

[6] R. G. Gallager, *Information theory and reliable communication*. Springer, 1968.

[7] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, January 1962.

[8] D. MacKay and R. Neal, "Good codes based on very sparse matrices," in *Cryptography and Coding*, ser. Lecture Notes in Computer Science, C. Boyd, Ed. Springer Berlin Heidelberg, 1995, vol. 1025, pp. 100–111.

[9] D. MacKay, *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003.

[10] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes," in *Communications, 1993. ICC '93 Geneva. Technical Program, Conference Record, IEEE International Conference on*, vol. 2, May 1993, pp. 1064–1070 vol.2.

[11] C. Berrou and A. Glavieux, "Turbo codes," *Encyclopedia of Telecommunications*, 2003.

[12] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, July 2009.

[13] E. Şaşoğlu *et al.*, "Polarization and polar codes," *Foundations and Trends® in Communications and Information Theory*, vol. 8, no. 4, pp. 259–381, 2012.

[14] A. Guillén i Fàbregas, A. Martinez, and G. Caire, *Bit-interleaved coded modulation*. Now Publishers Inc, 2008.

[15] G. D. Forney and G. Ungerboeck, "Modulation and coding for linear gaussian channels," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2384–2415, 1998.

[16] D. J. Costello and G. D. Forney, "Channel coding: The road to channel capacity," *Proceedings of the IEEE*, vol. 95, no. 6, pp. 1150–1177, 2007.

[17] A. Barron and A. Joseph, "Least squares superposition codes of moderate dictionary size are reliable at rates up to capacity," *IEEE Transactions on Information Theory*, vol. 58, no. 5, pp. 2541–2557, Feb 2012.

[18] A. Joseph and A. R. Barron, "Fast sparse superposition codes have near exponential error probability for $R < \mathcal{C}$," *IEEE Transactions on Information Theory*, vol. 60, no. 2, pp. 919–942, Feb. 2014.

[19] U. Erez and R. Zamir, "Achieving 1/2 log (1+snr) on the awgn channel with lattice encoding and decoding," *IEEE Transactions on Information Theory*, vol. 50, no. 10, pp. 2293–2314, Oct 2004.

[20] R. Zamir, *Lattice Coding for Signals and Networks: A Structured Coding Approach to Quantization, Modulation, and Multiuser Information Theory*. Cambridge University Press, 2014.

[21] C. E. Shannon, *Coding theorems for a discrete source with fidelity criterion*. McGraw-Hill, 1960, pp. 93–126.

[22] A. R. Barron and S. Cho, "High-rate sparse superposition codes with iteratively optimal estimates," in *Proc. IEEE Int. Symp. Inf. Theory*, 2012.

[23] S. Cho and A. Barron, "Approximate iterative Bayes optimal estimates for high-rate sparse superposition codes," in *Sixth Workshop on Information-Theoretic Methods in Science and Engineering*, 2013.

[24] C. Rush and R. Venkataramanan, "The error exponent of sparse regression codes with AMP decoding," in *Proc. IEEE Int. Symp. Inf. Theory*, 2017.

[25] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the society for industrial and applied mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[26] A. El Gamal and Y.-H. Kim, *Network Information Theory*. Cambridge, 2011.

[27] D. L. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," *Proceedings of the National Academy of Sciences*, vol. 106, no. 45, pp. 18 914–18 919, 2009.

[28] A. Montanari, "Graphical models concepts in compressed sensing," *Compressed Sensing: Theory and Applications*, pp. 394–438, 2012.

[29] M. Bayati and A. Montanari, "The dynamics of message passing on dense graphs, with applications to compressed sensing," *IEEE Transactions on Information Theory*, pp. 764–785, 2011.

[30] S. Rangan, "Generalized approximate message passing for estimation with random linear mixing," in *Proc. IEEE Int. Symp. Inf. Theory*, 2011, pp. 2168–2172.

[31] J. Barbier and F. Krzakala, "Approximate message-passing decoder and capacity-achieving sparse superposition codes," *IEEE Transactions on Information Theory*, vol. 63, no. 8, pp. 4894–4927, 2017.

[32] R. Venkataramanan, T. Sarkar, and S. Tatikonda, "Lossy compression via sparse linear regression: Computationally efficient encoding and decoding," *IEEE Transactions on Information Theory*, vol. 60, no. 6, pp. 3265–3278, 2014.

[33] R. Venkataramanan, A. Joseph, and S. Tatikonda, "Lossy compression via sparse linear regression: Performance under minimum-distance encoding," *IEEE Transactions on Information Theory*, vol. 60, no. 6, pp. 3254–3264, 2014.

[34] R. Venkataramanan and S. Tatikonda, "The rate-distortion function and excess-distortion exponent of sparse regression codes with optimal encoding," *IEEE Transactions on Information Theory*, vol. 63, no. 8, pp. 5228–5243, 2017.

[35] M. Mezard and A. Montanari, *Information, physics, and computation*. Oxford University Press, 2009.

[36] J. Boutros and G. Caire, "Iterative multiuser joint decoding: Unified framework and asymptotic analysis," *IEEE Transactions on Information Theory*, vol. 48, no. 7, pp. 1772–1793, 2002.

[37] M. Yoshida and T. Tanaka, "Analysis of sparsely-spread cdma via statistical mechanics," in *Proc. IEEE Int. Symp. Inf. Theory*. IEEE, 2006, pp. 2378–2382.

[38] D. Guo and C.-C. Wang, "Multiuser detection of sparsely spread cdma," *IEEE journal on selected areas in communications*, vol. 26, no. 3, pp. 421–431, 2008.

[39] J. Barbier, C. Schülke, and F. Krzakala, "Approximate message-passing with spatially coupled structured operators, with applications to compressed sensing and sparse superposition codes," *Journal of Statistical Mechanics: Theory and Experiment*, no. 5, 2015.

[40] M. Bayati and A. Montanari, "The Dynamics of Message Passing on Dense Graphs, with Applications to Compressed Sensing," *IEEE Transactions on Information Theory*, vol. 57, no. 2, pp. 764–785, Feb. 2011.

[41] C. Rush and R. Venkataramanan, "Finite sample analysis of approximate message passing," *Proc. IEEE Int. Symp. Inf. Theory*, 2015, full version: https://arxiv.org/abs/1606.01800.

[42] A. Joseph and A. R. Barron, "Fast sparse superposition codes have near exponential error probability for $R < C$," *IEEE Transactions on Information Theory*, vol. 60, no. 2, pp. 919–942, Feb. 2014.

[43] Y. Takeishi, M. Kawakita, and J. Takeuchi, "Least squares superposition codes with Bernoulli dictionary are still reliable at rates up to capacity," *IEEE Transactions on Information Theory*, vol. 60, pp. 2737–2750, May 2014.

[44] J. L. Shanks, "Computation of the Fast Walsh-Fourier transform," *IEEE Transactions on Computers*, vol. 18, no. 5, pp. 457–459, May 1969.

[45] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[46] J. Barbier and F. Krzakala, "Replica analysis and approximate message passing decoder for superposition codes," in *Proc. IEEE Int. Symp. Inf. Theory*, 2014, pp. 1494–1498.

[47] S. Rangan, "Generalized approximate message passing for estimation with random linear mixing," in *Proc. IEEE Int. Symp. Inf. Theory*, 2011, pp. 2168–2172.

[48] (2008) The coded modulation library. [Online]. Available: http://www.iterativesolutions.com/Matlab.htm

[49] *131.0-B-2 TM Synchonization and Channel Coding*, CCSDS, August 2011. [Online]. Available: https://public.ccsds.org/Pubs/131x0b2ec1.pdf

[50] R. Venkataramanan and S. Tatikonda, "Sparse regression codes for multi-terminal source and channel coding," in *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*. IEEE, 2012, pp. 1966–1974.

[51] H. S. Witsenhausen, "On source networks with minimal breakdown degradation," *Bell System Technical Journal*, vol. 59, no. 6, pp. 1083–1087, 1980.

[52] S. Gel'fand and M. Pinsker, "Coding for channel with random parameters," in *Problems of Control and Information Theory*, 01 1980, vol. 9.

[53] M. Costa, "Writing on dirty paper (corresp.)," *IEEE Transactions on Information Theory*, vol. 29, no. 3, pp. 439–441, 1983.

[54] A. Wyner and J. Ziv, "The rate-distortion function for source coding with side information at the decoder," *IEEE Transactions on Information Theory*, vol. 22, no. 1, pp. 1–10, 1976.

[55] S. S. Pradhan, J. Chou, and K. Ramchandran, "Duality between source coding and channel coding and its extension to the side information case," *IEEE Transactions on Information Theory*, vol. 49, no. 5, pp. 1181–1203, 2003.

[56] J. Wolf, A. Wyner, and J. Ziv, "Source coding for multiple descriptions," *Bell System Technical Journal*, vol. 59, no. 8, pp. 1417–1426, 1980.

[57] A. E. Gamal and T. M. Cover, "Achievable rates for multiple descriptions," *IEEE Transactions on Information Theory*, vol. 28, no. 6, pp. 851–857, Nov. 1982.

[58] J. Chen, C. Tian, T. Berger, and S. S. Hemami, "Multiple description quantization via gram–schmidt orthogonalization," *IEEE Transactions on Information Theory*, vol. 52, no. 12, pp. 5197–5217, 2006.

[59] M. J. Wainwright and E. Martinian, "Low-density graph codes that are optimal for binning and coding with side information," *IEEE Transactions on Information Theory*, vol. 55, no. 3, pp. 1061–1079, 2009.

[60] M. J. Wainwright and E. Maneva, "Lossy source encoding via message-passing and decimation over generalized codewords of ldgm codes," in *Proc. IEEE Int. Symp. Inf. Theory*. IEEE, 2005, pp. 1493–1497.

[61] V. Aref, N. Macris, and M. Vuffray, "Approaching the rate-distortion limit with spatial coupling, belief propagation, and decimation," *IEEE Transactions on Information Theory*, vol. 61, no. 7, pp. 3954–3979, 2015.

[62] C. Condo and W. J. Gross, "Sparse superposition codes: A practical approach," in *IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2015, pp. 1–6.

[63] ——, "Implementation of sparse superposition codes," *IEEE Transactions on Signal Processing*, vol. 65, no. 9, pp. 2421–2427, 2017.