Practical Distributed Source Coding and Its Application to the Compression of Encrypted Data

Daniel Hillel Schonberg

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-93
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-93.html

July 25, 2007

Acknowledgement

I am very grateful to my advisor, Professor Kannan Ramchandran, for his guidance, support, and encouragement throughout the course of my graduate study. This research would not have been possible without his vision and motivation.

I would like to thank my dissertation committee, Kannan Ramchandran, Martin Wainwright, and Peter Bickel, for their assistance. Their comments and guidance have proved invaluable.

This research was supported in part by the NSF under grant CCR-0325311. Any opinions, findings, and conclusions or recommendations

expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Practical Distributed Source Coding and Its Application to the Compression of Encrypted Data

by

Daniel Hillel Schonberg

B.S.E. (University of Michigan, Ann Arbor) 2001
M.S.E. (University of Michigan, Ann Arbor) 2001

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Engineering-Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Kannan Ramchandran, Chair
Professor Martin Wainwright
Professor Peter Bickel

Fall 2007

The dissertation of Daniel Hillel Schonberg is approved.

Chair Date

Date

Date

University of California, Berkeley

Fall 2007

Practical Distributed Source Coding and Its Application to the Compression of Encrypted Data

by

Daniel Hillel Schonberg

Abstract

Practical Distributed Source Coding and Its Application to the Compression of Encrypted Data

by

Daniel Hillel Schonberg

Doctor of Philosophy in Engineering-Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Kannan Ramchandran, Chair

Distributed source codes compress multiple data sources by leveraging the correlation between those sources, even without access to the realization of each source, through the use of a joint decoder. Though distributed source coding has been a topic of research since the 1970s (Slepian and Wolf, 1973), there has been little practical work. In this thesis, we present a practical and flexible framework for distributed source coding and its applications.

Only recently, as applications have arisen, did practical work begin in earnest. The applications range from dense sensor networks to robust video coding for wireless transmission to the compression of encrypted data. Unfortunately, most published work offering practical codes consider only a limited scenario. They focus on simple idealized data models, and they assume *a priori* knowledge of the underlying joint statistics. These assumptions are made to ease the description and analysis of their solutions. We argue though that a full solution requires a construction flexible enough to be adapted to real-world distributed source coding scenarios. This solution must be able to accommodate any source and unknown statistics. In this thesis, we develop practical distributed source codes, applicable to idealized sources and real world sources such as images and videos, that are rate adaptable to all scenarios.

This thesis is broken into two halves. In the first half we discuss analytic considerations for generating practical distributed source codes. We begin by assuming *a priori* knowledge of the underlying source statistics, and then consider source coding with side information, a special case of distributed source coding. As a basis for our solution, we develop codes

for independent and identically distributed (i.i.d.) sources and then adapt the codes to parallel sources. We then generalize this framework to a complete distributed source coding construction, flexible enough for arbitrary scenarios and adaptable to important special cases. Finally we conclude this half by eliminating our assumption of *a priori* knowledge of the statistics. We discuss a protocol for rate adaptation, ensuring that our codes can operate blind of the source statistics. Our protocol converges rapidly to the entropy-rate of a stationary source.

In the second half of this thesis, we consider distributed source coding of real world sources. As a motivating application, we consider the compression of encrypted images and video. Since encryption masks the source, traditional compression algorithms are ineffective. However, through the use of distributed source-coding techniques, the compression of encrypted data is possible. It is possible to reduce data size without requiring data be compressed prior to encryption. We develop algorithms for the practical lossless compression of encrypted data. Our methods leverage the statistical correlations of the source, even without direct access to their instantiations. For video, we compare our performance to a state-of-the-art motion-compensated lossless video encoder that requires unencrypted video as input. It compresses each unencrypted frame of the "Foreman" test video sequence by 59% on average. In comparison, our proof-of-concept implementation, working on *encrypted* data, compresses the same sequence by 33%.

Professor Kannan Ramchandran
Dissertation Committee Chair

I dedicate my thesis to my parents, Ila and Leslie. Mom and Dad, thank you so much. Without your encouragement and support, I would have never gotten so far. Thank you so much for instilling in me a love of learning, and for so much more.

I further dedicate this thesis to my girlfriend, Helaine. Thank you so much for all the love and support you have given me.

Contents

Contents ii

List of Figures vi

List of Tables xiv

Acknowledgements xv

1 Introduction 1

2 Background 6

2.1 Traditional Source Coding 6

2.2 Channel Coding 8

2.3 Low-Density Parity-Check Codes and Graphical Models 9

2.3.1 The Sum-Product Algorithm and LDPC codes 11

2.3.2 LDPC Code Design 13

2.4 Distributed Source Coding 16

3 A Practical Construction for Source Coding With Side Information 20

3.1 Introduction 20

3.2 Linear Block Codes for Source Codes With Side Information 22

3.2.1 Illustrative Example 23

3.2.2 Block Codes For Source Coding With Side Information 24

3.3 Low-Density Parity-Check Codes 26

3.4 Practical Algorithm 27

3.4.1 Application of Sum-Product Algorithm 28

3.5 Simulations and Results 30

3.5.1 Compression With Side Information Results . . . 31
3.5.2 Compression of a Single Source . . . 32
3.6 Conclusions . . . 33

4 Slepian-Wolf Coding for Parallel Sources: Design and Error-Exponent Analysis 35
4.1 Introduction . . . 36
4.2 Error Exponent Analysis . . . 37
4.3 LDPC Code Design . . . 39
4.3.1 EXIT Chart Based Code Design . . . 41
4.4 Simulation Results . . . 44
4.5 Conclusions & Open Problems . . . 48

5 Symmetric Distributed Source Coding 52
5.1 Introduction . . . 53
5.1.1 Related Work . . . 54
5.2 Distributed Source Coding . . . 56
5.2.1 Illustrative Example . . . 56
5.2.2 Code Generation . . . 57
5.2.3 Decoder . . . 59
5.2.4 Multiple Sources . . . 61
5.3 Low-Density Parity-Check Codes . . . 62
5.4 Adaptation of Coding Solution to Parity Check Matrices . . . 63
5.5 Integration of LDPC Codes Into Distributed Coding Construction . . . 64
5.5.1 Application of Sum-Product Algorithm . . . 67
5.6 Simulations and Results . . . 67
5.7 Conclusions & Open Problems . . . 69

6 A Protocol For Blind Distributed Source Coding 71
6.1 Introduction . . . 71
6.2 Protocol . . . 72
6.3 Analysis . . . 74
6.3.1 Probability of Decoding Error . . . 74
6.3.2 Bounding the Expected Slepian-Wolf Rate . . . 77
6.3.3 Optimizing the Rate Margin . . . 78

6.4 Practical Considerations . . . 79
6.5 Results . . . 80
6.6 Conclusions & Open Problems . . . 82

7 The Compression of Encrypted Data: A Framework for Encrypted Simple Sources to Encrypted Images 84

7.1 Introduction . . . 85
7.2 General Framework . . . 87
7.2.1 Encryption Function . . . 89
7.2.2 Code Constraint . . . 89
7.2.3 Source and Source Model . . . 90
7.2.4 Decoding Algorithm . . . 91
7.3 From Compression of Encrypted Simple Sources to Encrypted Images . . . 92
7.3.1 1-D Markov Model . . . 93
7.3.2 2-D Markov Model . . . 93
7.3.3 Message Passing Rules . . . 94
7.3.4 Doping . . . 95
7.3.5 Experimental Results . . . 96
7.3.6 Gray Scale Images . . . 98
7.4 Conclusions & Open Problems . . . 99

8 Compression of Encrypted Video 101

8.1 Introduction . . . 101
8.1.1 High-Level System Description and Prior Work . . . 102
8.1.2 Main Results & Outline . . . 104
8.2 Source Model . . . 105
8.3 Message Passing Rules . . . 109
8.4 Experimental Results . . . 109
8.5 Blind Video Compression . . . 113
8.6 Conclusions & Open Problems . . . 114

9 Conclusions & Future Directions 116

Bibliography 120

A Codes and Degree Distributions Used to Generate LDPC Codes 127

B Deriving the Slepian-Wolf Error Exponent From the Channel Coding Error Exponent **136**

List of Figures

1.1 A block diagram for the distributed source coding problem. The encoders act separately on each source for joint recovery at a common receiver. There is theoretically no loss of lossless sum rate compression performance due to the distributed nature of the problem, as shown in Slepian and Wolf [78]. . 2

2.1 A block diagram for single source and single destination source coding. The encoder maps source symbols x^n to a bitstream, which is then fed to the decoder. The decoder reconstructs $\hat{x}^n$. The rate R is the number of bits transmitted per input symbol. 7

2.2 A block diagram for channel coding. The source word M is mapped by the encoder to x^n. x^n is then transmitted through the channel and received at the decoder as y^n. To ensure efficient and reliable communication, the goal is to design the mapping of M to x^n such that $\hat{M}$ can be recovered to M from y^n with high probability. 8

2.3 A sample factor graph. The five circles represent bits while the four squares represent the parity constraints of the code. Connections between the two types of nodes correspond to the parity check matrix given in Table 2.1. A labeling of the two types of messages passed between nodes when running the sum-product algorithm is given. 10

2.4 A graphical representation of a random irregular LDPC code. The degree of each node is determined according to a degree distribution. Each edge is then randomly connected to a variable node and a check node according to some random permutation. 13

2.5 An example of a good code degree distribution pair for channel coding rate $R = 0.55$, designed using EXIT charts. The tight fit of the two curves in the plot shows that the code operates near to capacity. The dashed curve corresponds to the variable nodes while the solid curve corresponds to the check nodes. Decoding operates by moving from the left-hand side to the right-hand side of the graph, going through the gap between the two curves. Since the gap between the two curves is positive, decoding will follow the path between the curves successfully. 16

2.6 An extension of the traditional single source and single destination source coding problem to two sources, x^n and y^n. Here, the minimum rate according to information theory is the joint entropy of the two sources, i.e. $R \geq H(x, y)$. 17

2.7 The block diagram for the distributed source coding problem. Here, without communication between the two, one encoder maps the source symbols x^n to a bitstream while the other maps y^n to a separate bitstream. The decoder receives both bit streams and reconstructs $\hat{x}^n$ and $\hat{y}^n$. The minimum rate region for R_x and R_y is given in Equation (2.2). 17

2.8 A graphical representation of the Slepian-Wolf rate region as specified in Equation (2.2). 18

2.9 A special case of distributed source coding referred to as source coding with side information. Here, the correlated source y^n is available at the decoder, but is unavailable at the encoder. Using the Slepian-Wolf rate region of Equation (2.2), we see that the achievable rate is bounded as $R_x \geq H(x|y)$. 19

3.1 The source coding with side information problem. The encoder seeks to compress x given that y is available at the decoder but unavailable to itself. 22

3.2 A sample factor graph. The circles represent bits while squares represent the parity constraints of the code. A labeling of the two types of messages passed from node to node in the decoding algorithm is given. 26

3.3 The factor graph for source coding with side information. Circles labeled x_i or y_i represent source bits. Circles labeled s_{x_j} represent the compressed bits. Squares labeled f_{x_j} represent the LDPC code applied to the source. Squares labeled f_i represent the correlation between bits x_i and y_i. 28

3.4 The factor graph for entropy coding. Circles labeled x_i represent source bits. Circles labeled s_{x_j} represent the compressed bits. Squares labeled f_{x_j} represent the LDPC code applied to the source. Squares labeled f_i represent the source constraint on x_i. This model develops as a graceful degradation of the source model of Figure 3.3 in the absence of correlated side information. 28

3.5 Source coding with side information simulation results for several block lengths and varying the correlation parameter $\alpha(= \beta)$. Source x is compressed to rate 1/2 while source y is available at the decoder. Each data point is the result of the simulation of at least 10^6 bits. The results in this figure are plotted in terms of their bit error rate (BER). In addition, we plot one standard deviation error bars. Similar error bars result for all BER results in this chapter, but are omitted for clarity. 32

3.6 Results of the same experiments as in Figure 3.5 plotted in terms of the symbol error rates (SER). Each value is the result of the simulation of at least 10^3 symbols (blocks). In addition, we plot one standard deviation error bars. 33

3.7 Simulation results compressing source x to rate 1/2 while source y is available at the decoder. The simulations are of source coding with side information results, as in Figure 3.5. Each data point is the result of the simulation of at least 10^6 bits. The block length is fixed to 10,000 and a more general correlation structure is considered. The type of data point used indicates the marginal entropy of the source x. 33

3.8 Simulation results from entropy coding source x to rate 1/2. This plot considers single source results for several block lengths and source statistics. Each data point is the result of the simulation of at least 10^6 bits. Resulting coding performance is similar to that obtained when source coding with side information as in Figure 3.5. 34

4.1 A comparison of the bound presented in Equation (4.3) for joint coding versus the bound presented in Equation (4.4) for separate coding. The details of the scenario considered are discussed in the text. For each value of R, the values of R_1 and R_2 are chosen to give the least upper bound on the probability of error. This demonstrates the improved performance achievable with joint coding across parallel sources. 40

4.2 A sample LDPC graphical model representing the matrix at right. The circles represent bits and correspond to matrix columns while the squares represent parity constraints and matrix rows. Elements of the matrix indicate edges between nodes. Whenever the element in the ith row and jth column is 1, an edge connects constraint node i with variable node j. A 0 in the corresponding position implies no edge connects the nodes. 41

4.3 One example of a code designed using EXIT charts for a rate R=0.45 source code. The tight fit of the two curves shows that the degree distributions are good. The dashed curve corresponds to the variable nodes while the solid curve corresponds to the check nodes. Decoding operates by moving from the left-hand side to the right-hand side of the graph, going through the gap between the two curves. Since the gap between the two curves is positive, decoding will follow the path between the curves successfully. 42

4.4 A second example of a code using EXIT charts, as in Figure 4.3. This code is a rate R=0.55 source code. Since the gap between the two curves is positive here as well, decoding will follow the path between the curves successfully. . 43

4.5 A comparison of the EXIT charts for separate coding of two sources. These curves are taken from Figures 4.3 and 4.4. Note the intersection of the curves for $\lambda_2(x)$ and $\rho_1(x)$ prevents the use of these curves for a joint code. Since the gap between these two curves is not maintained, decoding will fail. . . . 44

4.6 A plot of the EXIT chart for the best possible degree distributions found for coding over parallel sources. For these curves, we have considered one check node curve (solid line) and matched both variable node curves (dashed lines) to it. Since the gap between the variable node curves and the check node curve is always positive, decoding will succeed. 45

4.7 $\log_{10}(P(Error))$ results of the simulations for separate coding. The first code is designed with rate $R_1 = 0.45$ (Figure 4.3) while the second code is designed with rate $R_2 = 0.45$ (Figure 4.4). The binary entropy ranges of Q_1 and Q_2 are shown on the horizontal and vertical axes. Dark regions of the plot indicate low probability of error while light regions indicate high probability of error. In these plot, the probability of error is high whenever $H_2(Q_1)$ *or* $H_2(Q_2)$ are large 47

4.8 $\log_{10}(P(Error))$ results of the simulations using a code designed for rate $R = 0.5$. The binary entropy ranges of Q_1 and Q_2 are shown on the horizontal and vertical axes respectively. Dark regions of the plot indicate low probability of error while light regions indicate high probability of error. Unlike the separate coding results shown in Figure 4.7, the probability of error only gets large when $H_2(Q_1)$ *and* $H_2(Q_2)$ are large. 47

4.9 $\log_{10}(P(Error))$ results of the simulations for the EXIT chart based design method (Figure 4.6). The binary entropy ranges of Q_1 and Q_2 are shown on the horizontal and vertical axes respectively. Dark regions of the plot indicate low probability of error while light regions indicate high probability of error. Like the single code simulations shown in Figure 4.8, the probability of error only gets large when $H_2(Q_1)$ *and* $H_2(Q_2)$ are large. 48

4.10 Comparison of the results from Figures 4.7 and 4.9. The region of $P(error) > 10^{-2}$ is considered a decoding failure while everywhere else is a success. In this plot, decoding success is seen in the lower left corner and decoding failure in the upper right corner. The middle region represent where the EXIT chart designed code succeeds and the separate coding fails. This figure shows the advantage of joint compression. 49

4.11 Comparison of the results from Figures 4.8 and 4.9. The regions of $P(error) > 10^{-2}$ is considered a decoding failure while everywhere else is a success. As in Figure 4.10, in this plot decoding success is seen in the lower left corner and decoding failure in the upper right corner. The middle region represent where the EXIT chart designed code succeeds and the simple rate $R = 0.5$ code fails. This figure shows the advantage of designing the code for the parallel sources. 49

5.1 The partition and distribution of a single code's generator matrix for the codes for compressing x and y. $\mathbf{G}_c$ is shared between both resulting codes to allow for arbitrary rate point selection. 59

5.2 Iterative generator matrix definition process for achieving a corner point for the distributed compression of L sources. The process initializes with the L-th matrix on the right-hand side and iterates until the left-most matrix is defined. After defining these codes, arbitrary points can be achieved by moving rows between generator matrices. 62

5.3 A sample factor graph. The circles represent bits while squares represent the parity constraints of the code. A labeling of the two types of messages passed node to node in the decoding algorithm is given. 63

5.4 Factor graph for distributed source coding. Circles labeled x_i or y_i represent source bits. Circles labeled S_{x_j} or S_{y_j} represent the compressed bits. Squares labeled f_{x_j} or f_{y_j} represent the LDPC code applied to each source. Squares labeled f_i represent the correlation between bits x_i and y_i. 65

5.5 An abstracted version of the factor graph of Figure 5.4. Recall that we use rectangles as well as squares to represent constraints. Here the graph is abstracted into the operations on each source and the correlation interface between the two. This graph can be extended to an arbitrary number of sources by interfacing additional sources with the generalized correlation constraint. 66

5.6 Simulation results for several arbitrary rate combinations in the Slepian-Wolf region for a block length of $10,000$. Each data point is the result of the simulation of at least 10^6 bits. The BER results are plotted versus the difference between the sum rate and the Slepian-Wolf bound. As the difference grows, resulting error rates improve. 69

6.1 A block diagram for blind source coding with side information. For the analysis, we assume the simple correlation structure for x_i and y_i as shown here. During each cycle, the source is transmitted at rate R_k over the forward channel. The decoder feeds back an acknowledgment and an estimate of the parameter $\hat{Q}_k$. 72

6.2 Error exponents and quadratic approximations for $Q = 0.05$ and $Q = 0.3$. Note, for $Q = 0.3$, the approximation is so close to the error exponent that the lines are on top of each other. This demonstrates that the quadratic approximation is quite good. 77

6.3 A block diagram for adaptation of our blind distributed source coding algorithm to the general distributed source coding problem. During each cycle, the source is transmitted at rate R_k over the forward channel. The decoder feeds back an acknowledgment and a rate request, R_{k+1}, for the next cycle. 79

6.4 Averaged results (over 25 trials) of blind source coding with side information of sources as modeled in Section 6.2. The horizontal axis is the number of cipher-text bits considered, n_k, in log-scale, and the vertical axis is the system redundancy (the percent of bits used above the entropy rate). We present one standard deviation error bars to demonstrate consistency. As the number of bits transmitted grows performance improves. Note, the bound from Equation (6.18) plotted here excludes feedback costs. Ideally, we would like to transmit at 0% redundancy (the entropy rate). 82

6.5 Averaged results (over 25 trials) of blind source coding with side information where the source model of Section 6.2 is altered such that the samples of z_i exhibit Markov memory. Similar to Figure 6.4, the horizontal axis is the number of cipher-text bits considered, n_k, in log-scale, and the vertical axis is the system redundancy (the percent of bits used above the entropy rate). As the number of bits transmitted grows performance improves. 83

7.1 The source is first encrypted and *then* compressed. The compressor does not have access to the key used in the encryption step. The decoder jointly decompresses and decrypts. The key serves as the decoder side information. 85

7.2 A sample 100×100 world map image is on the left ($10,000$ bits total). To encrypt this image, the $10,000$ bit random key in the center is added to the unencrypted image on the left (using a bit-wise exclusive-OR). 86

7.3 The abstracted model for compressing encrypted data. This model consists of two parts. The first part, the generative model and encoder, describes how the raw source data is first encrypted and then compressed. This part operates on the true source bits, x_i, the key bits, k_i, the encrypted bits, y_i, and the compressed bits, s_i. The second part, the decoder, shows how the various components of the source are modeled at the decoder for recovering an estimate of the original source. This part uses the same compressed bits, s_i, and key bits k_i, but works on an estimate of the encrypted bits, $\hat{y}_i$, and the source bits, $\hat{x}_i$. 88

7.4 The graphical constraints imposed by stream-cipher encryption. The source estimate $\hat{x}_i$, the key bit k_i, and the cipher-text estimate $\hat{y}_i$ must have even parity. 89

7.5 The graphical model of the constraints imposed by using a LDPC code for compression. All cipher-text estimate $\hat{y}_i$ connected to the same parity-check g_j must sum to the syndrome value s_j. 90

7.6 The simple i.i.d. model considered in [38] and in Chapter 3. This model consists of the source estimates $\hat{x}_i$ and their priors. 91

7.7 The 1-D source model. This model consists of bits (circles), source priors (squares below the bits) and correlation constraints (squares between the bits). In addition, message labels, μ and ν, are presented. 93

7.8 A section of the 2-D source model. This model includes the source bits (circles) and both horizontal correlations, $M^h_{i,j}$, and vertical correlations, $M^v_{i,j}$. 94

7.9 A sample 100×100 world map image (also shown in Figure 7.2) is on the left ($10,000$ bits total). To encrypt this image, the $10,000$ bit random key in the center is added to the unencrypted image on the left (using a bit-wise exclusive-OR). 96

7.10 A comparison of the compressed bits and reconstructed image using the 1-D Markov model and the 2-D Markov model. The 1-D model compressed the encrypted data to $7,710$ bits. This is $3,411$ bits more than the $4,299$ bits used for the 2-D model. The i.i.d model could not compress the encrypted image at all. 97

7.11 A comparison of the intermediate estimates at the decoder. The estimates from the 1-D Markov model are on the top row, while estimates from the 2-D model are on the bottom row. The estimates from the 1-D model exhibit convergence along north-south lines. In contrast, the 2-D model decoder exhibits a localized clumping nature. 98

7.12 A sample image, "Foreman," is shown on the left (811,008 bits total). To encrypt this image, the 811,008 bit random key in the center is added to the unencrypted image (using a bitwise exclusive-OR). Many classical methods exist to compress the "Foreman" image. These methods use techniques such as transform based methods and localized predictors. Unfortunately, due to the nonlinear effects of bitwise encryption, none of the classical methods or techniques will successfully compress the encrypted image. 99

8.1 A sample frame from the standard "Foreman" video test sequence is shown on the left (811,008 bits total). To encrypt this frame, the 811,008 bit random key in the center is added to the unencrypted image (using a bitwise exclusive-OR). Although many classical methods exist to compress the "Foreman" video sequence, none will successfully compress the encrypted video. 103

8.2 The source is first encrypted and *then* compressed. The compressor does not have access to the key used in the encryption step. The decoder jointly decompresses and decrypts. The key serves as the decoder side information. 104

8.3 A reprint of Figure 7.8. A section of the 2-D source model. This model includes the source bits (circles) and both horizontal correlations, $M^h_{i,j}$, and vertical correlations, $M^v_{i,j}$. We apply this model to each bit plane of each frame of the video sequences. 106

8.4 A diagram of the way by which information is generated and fed to the decoder for each frame. Here, predicted frames are developed from the two previous frames. A measure of the predictor's accuracy is generated by measuring the accuracy of the previous frame's predictor. This gives the decoder sufficient assistance to recover the frame with the standard joint decompresser and decrypter discussed in Chapter 7. 107

8.5 Predictor generation using motion compensation and motion extrapolation. Frames $\hat{x}^{t-2}$ and $\hat{x}^{t-1}$ are available after having been previously decoded. For each block in frame $\hat{x}^{t-1}$, motion estimation is performed to find the best matching block in frame $\hat{x}^{t-2}$. In the above example, the best predictor for the block with thickened edges in $\hat{x}^{t-1}$ is the lightly thickened box in $\hat{x}^{t-2}$, shown with its corresponding motion vector (v_x, v_y). We estimate the motion vector for each block in frame $\hat{x}^t$ with that of its co-located block in frame $\hat{x}^{t-1}$. Its predictor is then indicated by the estimated motion vector. Here, the predictor for the block with thickened edges in $\hat{x}^t$ is the lightly thickened box in $\hat{x}^{t-1}$, as pointed to by the motion vector (v_x, v_y). 108

8.6 A sample compression of the fourth frame of the "Foreman" sequence. This block diagram demonstrates the generation of each of the elements available to the encoder and to the decoder. Here, the decoder is able to reconstruct frame 4 from the predicted frame 4, the probability model, and the compressed bits. The encryption key has been omitted from this figure for clarity. 112

8.7 Compression results for the 9 frames considered by our system. The average rate (in source bits per output bit) used on each frame. This plot demonstrates consistent rate savings from frame to frame. 112

8.8 Compression results for the 9 frames considered by our system. This plot shows the rate used on each bit plane, ranging from most significant (left) to least significant (right). Maximum and minimum are taken across all 9 frames. This demonstrates the consistent compressibility of each bit plane from frame to frame. It further demonstrates that successively less significant bit planes are successively less compressible. 113

List of Tables

2.1 The parity check matrix corresponding to the graph shown in Figure 2.3. Whenever the element in the ith row and jth column is 1, an edge connects constraint node i with variable node j. A 0 in the corresponding position implies no edge connects the nodes. 10

3.1 Structure of the joint distribution between x and y considered for Section 3.5. 31

5.1 Structure of the joint distribution between x and y considered for Section 5.6. 68

5.2 Simulation results for several arbitrary rate combinations in the Slepian-Wolf region for a block length of $10,000$. The BER$\times 10^4$ results are given for several rate combinations. As the sum rate grows, the resulting error rates improve. 69

8.1 Comparison of results for the compression of three encrypted sequences compared to results of traditional encoders compressing the unencrypted sequences. Despite operating on encrypted data, the proposed approach performs well. 111

8.2 Results for the blind compression of the three encrypted sequences. The proposed approach achieves significant compression gains. This demonstrates the value of the approach presented here. 114

Acknowledgements

I am very grateful to my advisor, Professor Kannan Ramchandran, for his guidance, support, and encouragement throughout the course of my graduate study. This research would not have been possible without his vision and motivation.

I would like to thank my dissertation committee, Kannan Ramchandran, Martin Wainwright, and Peter Bickel, for their assistance. Their comments and guidance have proved invaluable.

I would like to thank my colleagues and collaborates, in alphabetical order, Stark Draper, Prakash Ishwar, Mark Johnson, Vinod Prabhakaran, Sandeep Pradhan, Rohit Puri, and Chuohao Yeo for their invaluable help at various stages of this work and for the many productive discussions we had. I would like to single out Stark Draper for special acknowledgment. I cannot thank him enough for all he's done for me.

I would also like to thank all the members of the BASiCS group, past and present, for making my stay a memorable and enjoyable one. I would like to specially acknowledge those regularly in the lab space, listed in alphabetical order, Jim Chou, Mark Johnson, Abhik Majumdar, Vinod Prabhakaran, June Wang, Wei Wang, Ben Wild, and Chuohao Yeo. Thank you for making the lab such an excellent place to work.

I would like to thank all the professors and students of the Berkeley Wireless Foundation. I highly value the research community supported there.

I would like to acknowledge the support of my friends and family. I would especially like to acknowledge the support of my girlfriend, Helaine Blumenthal. There are too many more to mention here, but I am deeply grateful for all of you.

This research was supported in part by the NSF under grant CCR-0325311. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Curriculum Vitæ

Daniel Hillel Schonberg

Education

2001	University of Michigan, Ann Arbor B.S.E., Electrical Engineering
2001	University of Michigan, Ann Arbor M.S.E., Electrical Engineering: Systems
2007	University of California, Berkeley Ph.D., Electrical Engineering and Computer Science

Experience

2002 – 2007 Graduate Research Assistant, University of California, Berkeley.

- Design and implement a variety of codes for distributed source coding problems. Application to the compression of encrypted video sources and blind distributed source coding.

2003 – 2003 Research Intern, Microsoft Research.

- Develop EyeCerts, a biometric, cryptographically secure, identification system.
- Aid in developing forensics to analyze Stirmark attacks.

2001 – 2002 Graduate Student Instructor, University of California, Berkeley.

- EECS 123: Digital Signal Processing
- EECS 42: Introduction to Digital Electronics
- Run two discussion sections for 20-25 students, and hold weekly office hours.
- Perform course work, including writing homework and grading exams.

2001 – 2001 Summer Intern, MIT Lincoln Laboratory.

- Performed analysis of simulated radar data for the E2-C Hawkeye. Analysis focused on determining error in steering due to wing effects and multi-path.
- Studied principles of airborne radar and adaptive antenna arrays.

2000 – 2000 Research Scientist, Ford Motor Company.

- Wrote Visual Basic model to analyze vapor cold start applications.
- Performed extensive evaluation of latest NTK acoustic fuel vapor sensor prototype.
- Wrote two internal Ford reports upon work completed and presented findings.

Selected Publications

Journal Publications

D. Schonberg, C. Yeo, S. C. Draper, K. Ramchandran, "On Compression of Encrypted Images and Video Sequences," *IEEE Transactions on Information Forensics and Security*, Submitted for Publication.

D. Schonberg, S. S. Pradhan, K. Ramchandran, "One Linear Code can Achieve the Arbitrary Slepian-Wolf Bound: Design and Construction," *IEEE Transactions on Communications*, Accepted for Publication.

D. Schonberg, D. Kirovski, "EyeCerts," *IEEE Transactions on Information Forensics and Security*, Volume: 1, Issue: 2, June 2006, Pages: 144-153.

M. Johnson, P. Ishwar, V. M. Prabhakaran, D. Schonberg, K. Ramchandran, "On Compressing Encrypted Data," *IEEE Transactions on Signal Processing, Supplement on Secure Media I*, Volume: 52, Issue: 10, October 2004, Pages 2992-3006.

Conference Publications

D. Schonberg, S. C. Draper, R. Puri, and K. Ramchandran, "Slepian-Wolf Codes for Parallel Sources: Design and Error-Exponent Analysis," *43rd Annual Allerton Conference*, Monticello, IL, September 2005.

S. Draper, P. Ishwar, D. Molnar, V. Prabhakaran, K. Ramchandran, D. Schonberg, and D. Wagner, "An Analysis of PMF Based Tests For Least Significant Bit Image Steganography," *Proceedings of the Information Hiding Workshop*, June 2005.

D. Schonberg, and D. Kirovski, "Fingerprinting and Forensic Analysis Of Multimedia," *Proceedings of the ACM Multimedia 2004*, pp. 788-795, October 2004.

Biographical Sketch

Daniel Schonberg, received the B.S.E. and M.S.E. degrees in electrical engineering from the University of Michigan, Ann Arbor in 2001 as part of a joint 4 year program. He was granted his Ph.D. degree in electrical engineering from the University of California, Berkeley in 2007. He studied there under the advice of Professor Kannan Ramchandran.

His general research interests are in the fields of Signal Processing and Communications. He has focused on applied coding theory and statistical modeling during preparation for his dissertation research.

Chapter 1

Introduction

The distributed source coding problem, i.e. removing source redundancy in networks, is finding application in a growing variety of areas. Distributed source coding is an extension of the traditional, single source and single destination source coding problem, i.e. removing source redundancy for the purpose of efficient communication. Applications of distributed source coding, for which no traditional source coding solution is sufficient, range from dense sensor networks to robust video transmission for wireless multimedia applications [68], to compression of encrypted data [38]. Though varied, each application could benefit from a robust yet adaptable distributed source code construction.

The problem of lossless distributed compression of finite-alphabet sources goes back to the seminal work of Slepian & Wolf in 1973 [78]. The Slepian-Wolf theorem states, somewhat surprisingly, that there is theoretically no loss of performance due to the distributed nature of the problem. They show that each source can be coded to its conditional entropy as long as the sum rate is at least the joint entropy (defined via the Shannon entropy function). The Slepian-Wolf rate region for two arbitrarily correlated sources x and y is bounded by the following inequalities

$$R_x \geq H(\mathsf{x}|\mathsf{y}), \quad R_y \geq H(\mathsf{y}|\mathsf{x}), \quad \text{and } R_x + R_y \geq H(\mathsf{x},\mathsf{y}).$$

While the seminal information theory for the distributed source coding problem, studied in the papers [78] and [89], characterized the associated fundamental compression bounds for lossless and lossy compression, respectively, for asymptotically long block lengths, the resulting analysis/coding prescriptions were initially limited to the case of independent and identically distributed (i.i.d.) source and side-information correlation models. More

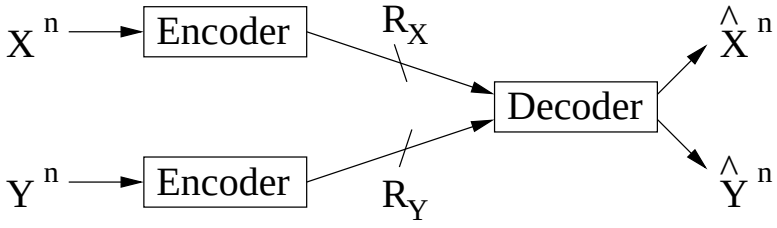


Figure 1.1. A block diagram for the distributed source coding problem. The encoders act separately on each source for joint recovery at a common receiver. There is theoretically no loss of lossless sum rate compression performance due to the distributed nature of the problem, as shown in Slepian and Wolf [78].

importantly, recent practical code construction methods for the distributed source coding problem, starting from Pradhan & Ramchandran [64] to more recent[1] state-of-the-art extensions [2, 94, 48, 14, 16, 79] have almost exclusively focused on a special case of the distributed source coding problem. First, they focus on the i.i.d. source/side-information correlation models and second, they assume an *a priori* knowledge of the Slepian-Wolf rate constraints.

We argue that fully general distributed source codes are important for both the applications mentioned above and others yet to be developed. In the first half of this dissertation, we provide a fully general and fully adaptable distributed source coding solution. We tackle the two assumptions made in prior work in order.

We begin by developing distributed source codes for the problem of source coding with side information to act as a basis for our work. Then, we generalize these source codes with side information to parallel sources, important for the application of distributed source codes to block stationary sources. We then build on this construction to offer a distributed source coding solution that can account for (i) arbitrary correlations, (ii) the entire Slepian-Wolf coding region corresponding to arbitrary rate combinations, and (iii) an arbitrary number of sources. The construction presented is based on *only a single linear block code*, and is adaptable to a wide range of distributed coding problems. Our scheme shows that fixed-length codes can be used for both point-to-point (a departure from more common variable length coding schemes applied to discrete sources) and Slepian-Wolf source coding. We include LDPC codes in our constructions, since they represent a powerful state-of-the-art class of capacity achieving codes. We describe how to incorporate LDPC codes into our

[1]See Section 5.1.1 for an extensive discussion on related work. Though the literature is extensive, they have mostly focused on very limited cases of the distributed source coding problem.

framework and how they are an important tool for building practical distributed source codes.

For a fully adaptable framework for distributed source codes, we next eliminate the assumption of *a priori* knowledge of the minimum rate constraint. Unfortunately, due to the distributed nature of the problem, the distributed source encoders cannot analyze sample data and determine the minimum rate is done in traditional source coding solutions. To resolve this issue, we present a feedback-based scheme for, so-called, blind distributed source coding, i.e. these codes operate without *a priori* knowledge of the source statistics.

We summarize our contributions in this first half of this thesis as follows.

- We contribute a fully general lossless distributed source coding construction. Our construction is adaptable to arbitrary correlation structures, an arbitrary number of sources, and can accommodate arbitrary rate choices.
- Our construction contributed gracefully degrades to special cases of the distributed source coding problem, such as source coding with side information and entropy coding.
- We contribute analysis and design techniques for the application of our construction to block stationary sources.
- To ensure our construction can be used in scenarios lacking foreknowledge of the source statistics, we contribute a transmission scheme for our distributed source coding construction that can operate blind of the source statistics. We both analyze this scheme and demonstrate its performance.

Having developed a general framework for distributed source coding, in the second half of this thesis we focus on the application of distributed source codes for compressing encrypted data. We consider sources characterized by statistical redundancy, such as images and video sequences, that have been encrypted uncompressed. Since encryption masks the source, traditional data compression algorithms are rendered ineffective. However, as has been shown previously, the compression of encrypted data is in fact possible through the use of distributed source-coding techniques. This means that it is possible to reduce data size without requiring that the data be compressed prior to encryption. Indeed, under some reasonable conditions, neither security nor compression efficiency need be sacrificed when compression is performed on the encrypted data (Johnson et al., 2004 [38]). Thus, in the second half of this dissertation we develop distributed source codes for the practical lossless

compression of encrypted data sources. We consider algorithms for a variety of sources, ranging from i.i.d. to images to video sequences. Our methods are designed to leverage the statistical correlations of the source, even without direct access to their instantiations.

We summarize our contributions in this second half as follows.

- We present a practical framework, based on our distributed source coding construction of the first half of this thesis, for compressing a variety of sources. These sources range from simple sources to images.
- We contribute an extension to the compression of encrypted data framework for compressing encrypted video sequences.
- Incorporating the blind transmission scheme presented here, we give a fully automatic system for compressing encrypted video.

This thesis is organized as follows. Chapter 2 is provided as background material for the reader. We begin development of distributed source codes by providing a construction for the source coding with side information problem in Chapter 3. We build on this construction in Chapter 4, developing codes for parallel and non-stationary sources. Chapter 5 extends the construction of Chapter 3 to develop fully arbitrary distributed source codes. In Chapter 6 we develop a blind transmission protocol so that our distributed source codes can operate without *a priori* knowledge of the source statistics.

The second half of this thesis shifts focus to the problem of compressing encrypted data. We describe the compression of encrypted data problem in Chapter 7, and provide a framework for solving the problem for various types of encrypted sources, ranging from simple sources to images. We extend these constructions to encrypted video sequences in Chapter 8. Finally, we conclude this thesis in Chapter 9.

Notation:

For clarity, we specify the notation we use throughout this thesis here. We denote random variables with a sans serif font, e.g. X, and their realizations with a Roman font, e.g. X. We use subscripts and superscripts to indicate a vector, e.g. $X_a^b = \{X_i\}_{i=a}^b = \{X_a, X_{a+1}, \ldots, X_{b-1}, X_b\}$. Note that a dropped subscript implies that the vector starts from 1, e.g. $X^b = \{X_i\}_{i=1}^b = \{X_1, X_2, \ldots, X_{b-1}, X_b\}$. We use bold to indicate vectors matrices, e.g. $\mathbf{X}$. Note, we occasionally use subscripts as the index of several random vectors instead of as an index into a random vector. For example, X_1 has samples $\mathsf{X}_{1,1}^n = \{\mathsf{X}_{1,1}, \mathsf{X}_{1,2}, \ldots\}$ and X_2 has samples $\mathsf{X}_{2,1}^n = \{\mathsf{X}_{2,1}, \mathsf{X}_{2,2}, \ldots\}$. The meanings in these cases is clear from context.

The Hamming weight of the vector x^n is denoted $wt_H(x^n)$. We denote the bitwise "exclusive-OR" (XOR) operation using the $\oplus$ operator. We use $H(\cdot)$ and $I(\cdot;\cdot)$ to represent the Shannon Entropy function and the Shannon Mutual Information functions respectively. Further, we use $H_2(\cdot)$ to represent the binary entropy function.

Additionally, we use some standard notation for linear block channel codes. The (n, k) block code $\mathcal{C}$ has generator matrix $\mathbf{G}$, parity check matrix $\mathbf{H}$, and parameter $m \triangleq n - k$.

Chapter 2

Background

In this chapter, we provide background material to aid the reader of this dissertation. We begin by describing two of the core problems of information theory; source coding (also referred to as data compression) in Section 2.1 and channel coding in Section 2.2. Having discussed channel coding in general, we then discuss a particular type of channel code known as LDPC (Low-Density Parity-Check) codes in Section 2.3. We focus on graphical models and the sum-product algorithm when discussing LDPC codes, important themes throughout this thesis. Finally, we give background on the distributed source coding problem, a focus of this thesis. Note that summaries of this material appear again interspersedly throughout the text to aid the reader.

2.1 Traditional Source Coding

This section presents a brief overview of point to point source coding, the problem of efficient communication of data from a single source to a single destination. We consider two scenarios; lossless source coding and lossy source coding. The goal of lossless source coding recover the original source exactly at the decoder. Lossy source coding aims to recover the original source only to within some distortion constraint. Throughout this dissertation, we focus primarily on lossless source coding and thus discus it first. We briefly discuss lossy source coding in this section to offer the reader a better understanding. For a more complete discussion of these topics, please refer to the standard text [19].

Consider the block diagram in Figure 2.1. Let x^n be an i.i.d. sequence of n random

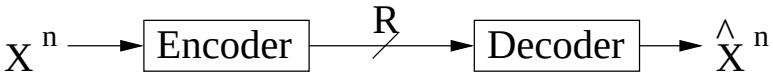


Figure 2.1. A block diagram for single source and single destination source coding. The encoder maps source symbols x^n to a bitstream, which is then fed to the decoder. The decoder reconstructs $\hat{x}^n$. The rate R is the number of bits transmitted per input symbol.

variables drawn from discrete alphabet $\mathcal{X}$ according to some distribution $P(x)$. For rate R (in bits per source symbol), the encoder map is as follows

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

The decoder performs the following mapping

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

For lossless source coding, the decoder aims to recover x^n with high probability. Defining $P_e^{(n)}$ as the probability of decoding error, we mathematically describe the aim of the design of the decoder

$$P_e^{(n)} = P(g(f(x^n)) \neq x^n) \to 0 \text{ as } n \to \infty.$$

The set of possible codewords $(f(1), f(2), \ldots, f(2^{nR}))$ is referred to as the codebook. We thus use R bits per source symbol to specify a codeword to the decoder. In order to provide efficiency, we seek to define the mappings $f(\cdot)$ and $g(\cdot)$ such that we minimize R while having the probability of error $P_e^{(n)} \to 0$ as $n \to \infty$ still holds to be true. Information theory provides that such mappings can be found for any R that satisfies the following inequality

$$R \geq H(x).$$

When some distortion in the reconstructed sequence $\hat{x}^n$ is acceptable, we perform lossy source coding. Given some distortion function, lossy source coding offers the minimal rate, R, given distortion constraint, D. Information theory gives that form of the optimal tradeoff between the minimum rate, R, and the distortion, D. The reader is referred to Cover and Thomas [19] for further details.

2.2 Channel Coding

In this section we consider another of the core problems of information theory, that of channel coding. In channel coding, we seek to efficiently and reliably communicate a data source over a noisy channel. For a more complete discussion of these topics please refer to Cover and Thomas [19].

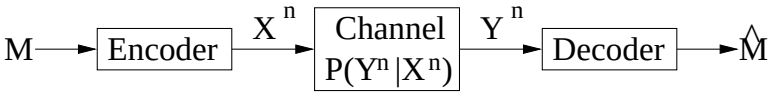


Figure 2.2. A block diagram for channel coding. The source word M is mapped by the encoder to x^n. x^n is then transmitted through the channel and received at the decoder as y^n. To ensure efficient and reliable communication, the goal is to design the mapping of M to x^n such that $\hat{M}$ can be recovered to M from y^n with high probability.

Figure 2.2 shows the basic block diagram of channel coding. The encoder is fed a message, M, from the set of possible messages $\{1, 2, \ldots, \mathcal{M}\}$. It then maps M to $x^n(M)$ and transmits it through the channel. The decoder receives y^n, distributed according to the channel $P(y^n|x^n)$. It then estimates M as $\hat{M}$ through a decoding rule. An error occurs whenever $\hat{M} \neq M$.

Formally, a discrete channel, denoted $(\mathcal{X}, P(y|x), \mathcal{Y})$, consists of finite input alphabet $\mathcal{X}$, finite output alphabet $\mathcal{Y}$, and a set of probability mass functions $P(y|x)$ indexed by x. A $(n, \mathcal{M})$ code $\mathcal{C}$ for channel $(\mathcal{X}, P(y|x), \mathcal{Y})$ consists of a set of messages, an encoding function, and a decoding function. The set of messages is $\{1, 2, \ldots, \mathcal{M}\}$. The encoding function maps the input message to a codewords

$$x^n : \{1, 2, \ldots, \mathcal{M}\} \to \mathcal{X}^n.$$

The decoding function, $g(\cdot)$, determines from y^n the value of x^n most likely sent by the encoder

$$g : \mathcal{Y}^n \to \{1, 2, \ldots, \mathcal{M}\}.$$

The maximum probability of error for any codeword of the $(n, \mathcal{M})$ code is given here

$$P_e^{(n)} = \max_i P(g(y^n) \neq i | x^n = x^n(i)). \tag{2.1}$$

The resulting rate R of the $(n, \mathcal{M})$ is $R = \log_2 \mathcal{M}/n$ bits per transmission. We would like to design $x^n(\cdot)$ and $g(\cdot)$ so as to maximize R while holding $P_e^{(n)} \to 0$ as $n \to \infty$ to

be true. Information theory gives the maximum achievable rate R meeting the constraint above, and defines this number as the channel capacity, C,

$$C = \max_{P(x)} I(x; y).$$

Binary linear block codes are a particular sub-class of channel codes. Defining $k = \log_2 \mathcal{M}$, an (n, k) linear block code $\mathcal{C}$ maps k input bits into n bits for transmission via a linear transformation. We refer to the transform matrix, $\mathbf{G}$, as the generator matrix. We thus see that a codebook is defined as the range space of the matrix $\mathbf{G}$. We further define a matrix, $\mathbf{H}$, called the parity check matrix. We define $\mathbf{H}$ as orthogonal to $\mathbf{G}$ and thus $\mathbf{H}$ describes the null space of $\mathbf{G}$. As a result, all valid codewords of $\mathcal{C}$ have a zero product with the matrix $\mathbf{H}$. The parity check matrix can thus be used to easily check if y^n is a codeword of $\mathcal{C}$ or not. In addition to the convenience of defining these codes in terms of these matrices, linear codes are also comparatively easier to implement then generally random codes. Further, linear codes are easier to analyze. Fortunately, as was shown by Gallager [27], linear codes can achieve channel capacity for the channels of interest in this dissertation (discrete memoryless channels with discrete input and output alphabets). Gallager's results means that linear codes offer no fundamental disadvantage.

2.3 Low-Density Parity-Check Codes and Graphical Models

In this section, we discuss a type of linear block channel code (Section 2.2) known as Low-Density Parity-Check (LDPC) codes. They are a powerful class of codes that can achieve rates within a small gap from capacity. Since they were originally developed by Gallager [26], LDPC codes are also referred to as Gallager codes. They were rediscovered by MacKay and Neal [52] and have since received considerable attention. In this section we give a brief overview of LDPC codes, and we frame our discussion in terms of graphical models.

As a particular type of linear block channel code, LDPC codes have advantageous pseudo random structure and a near Maximum Likelihood[1] (ML), low complexity, decoding algorithm. Factor graphs [42] (a type of graphical model as in Figure 2.3) (with its corresponding code in Table 2.1) are often used to visually represent LDPC codes. These

[1]ML decoding provides the best possible performance with regards to the probability of error as defined in Equation (2.1). Unfortunately, ML decoding is known to be NP hard. It is thus impractical to implement ML decoding for large block lengths, n.

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Table 2.1. The parity check matrix corresponding to the graph shown in Figure 2.3. Whenever the element in the ith row and jth column is 1, an edge connects constraint node i with variable node j. A 0 in the corresponding position implies no edge connects the nodes.

bipartite graphs consist of circles used to represent variables and squares (though more generally rectangles are used) used to represent constraints. For LDPC codes, we take the circles to represent bits and the squares as even parity constraints (applied to the bits to which they are connected).

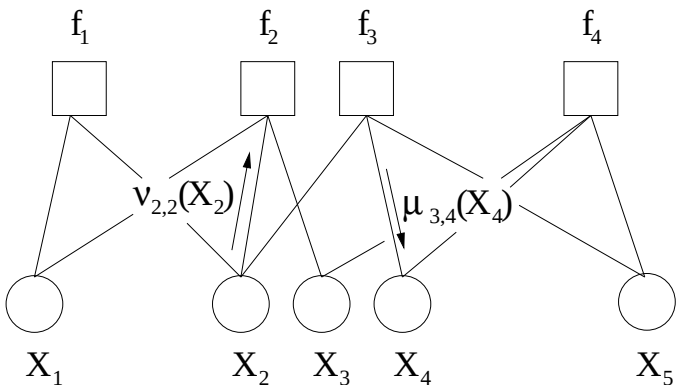


Figure 2.3. A sample factor graph. The five circles represent bits while the four squares represent the parity constraints of the code. Connections between the two types of nodes correspond to the parity check matrix given in Table 2.1. A labeling of the two types of messages passed between nodes when running the sum-product algorithm is given.

Decoding of LDPC codes is achieved using the sum-product algorithm [26, 61, 42]. The algorithm is an iterative inference algorithm that though exact on trees, is known to have good (but sub-optimal) empirical performance on loopy graphs such as those of an LDPC code. The algorithm is related to dynamic programming, and hence exact on trees. A loop is a path through the graph that has both the same starting and end point but travels along each of the edges in its path only once. The sum-product algorithm operates by iteratively estimating the marginal distribution for each bit. The algorithm initializes with estimates

of each marginal distribution (based on *a priori* or estimated statistics). In the first phase of each iteration, each variable node combines the estimates available and sends them along its edges (e.g., $\nu_{2,2}$ in Figure 2.3). In the second phase of each iteration, these estimates are merged, according to the constraint, at each check node for each bit node (e.g., $\mu_{3,4}$ in Figure 2.3). The marginals are used to estimate the bits after each iteration, and the set of estimates is checked against a stopping condition. If no stopping condition is met, the algorithm is stopped after a fixed number of iterations. Further details of the decoding algorithm are given in Section 2.3.1.

2.3.1 The Sum-Product Algorithm and LDPC codes

Though outlined above, in this section we specify the sum-product algorithm's explicit form. The following expressions are given in terms of the graph in Figure 2.3, but can be naturally extended to more general cases. We begin by expressing the update equations in their most general form for any graphical model. We then give the simplified form of the update equations as they apply to LDPC codes in particular. A more complete discussion of these topics can be found in Kschischang, Frey, & Loeliger [42].

Before specifying the update equations, we note that $\mathcal{N}(i)$ denotes all the nodes directly connected to node i (i.e., the neighborhood of i) and $\mathcal{N}(i)\backslash j$ denotes that same set less the node j. Further note that these equations often omit scaling factors. It is straightforward to scale the elements of a message so that they sum to 1, and are a proper distribution.

If each of the variables x_i in Figure 2.3 are distributed over the alphabet $\mathcal{X}$, then the messages to and from the node for x_i are of the following forms respectively

$$\mu_{j,i}(x_i),\ x_i \in \mathcal{X},\ \text{and}$$

$$\nu_{i,j}(x_i),\ x_i \in \mathcal{X}.$$

The variable node update equation to calculate the outgoing message from x_i to f_j, $\nu_{i,j}$ from the incoming messages is below

$$\nu_{i,j}(x_i) = \prod_{t \in \mathcal{N}(i)\backslash j} \mu_{t,i}(x_i).$$

We consider the update equations for the constraint nodes. We denote the mathematical form of the constraint as $f_j(x_{\mathcal{N}(j)})$. For the even parity constraint, for example, $f(x_1 = 1, x_2 = 1) = 1$ because of the even parity sum while $f(x_1 = 1, x_2 = 0) = 0$ because of the

odd parity sum. We give the explicit form here

$$f_j(x_{\mathcal{N}(j)}) = \begin{cases} 1 & \text{if } \bigoplus_{i \in \mathcal{N}(j)} x_i = 0 \\ 0 & \text{otherwise.} \end{cases}$$

The check node update equation to calculate the outgoing message from f_j to x_i, $\mu_{j,i}$ is given below

$$\mu_{j,i}(x_i) = \sum_{x_{\mathcal{N}(j)\setminus i}} \left(f_j(x_{\mathcal{N}(j)}) \prod_{s \in \mathcal{N}(j)\setminus i} (\nu_{s,j}(x_s)) \right).$$

Finally, for the variable node x_i to collect all the messages at a particular iteration into a single estimate of its distribution, it uses the rule below, where $\propto$ means proportional to within a scaling factor

$$P(x_i) \propto \sum_{j \in \mathcal{N}(i)} \mu_{j,i}(x_i).$$

We can make several simplifications to these equations (as in Richardson & Urbanke [70]), since this thesis primarily consider binary codes. We use these simplified forms throughout this thesis. Since we focus on binary alphabets, i.e. $\mathcal{X} = \{0, 1\}$, the message update rules can be also simplified. In particular, for some arbitrary distribution on x_i with $p_0 = P(x = 0)$ and $p_1 = P(x = 1)$, we represent this distribution as follows

$$\tau = \log \frac{p_0}{p_1}.$$

Given τ, recalling that $p_0 + p_1 = 1$, we can calculate p_0 and p_1. This assures us that no information is lost in this representative form

$$p_0 = \frac{e^\tau}{1 + e^\tau}, \quad p_1 = \frac{1}{1 + e^\tau}.$$

Using the simplified message forms, we rewrite $\nu_{i,j}(x_i)$ and $\mu_{j,i}(x_i)$ as $\nu_{i,j}$ and $\mu_{j,i}$ respectively, no longer needing an index for multiple valuables. We give the resulting update equations below. The variable node update equation is below

$$\nu_{i,j} = \sum_{t \in \mathcal{N}(i)\setminus j} \mu_{t,i}.$$

Since we focus on even parity sums, the update equation for the constraints over binary random variables is as follows

$$\mu_{j,i} = \log \frac{1 + \prod_{s \in \mathcal{N}(j)\setminus i} \tanh \nu_{s,j}/2}{1 - \prod_{s \in \mathcal{N}(j)\setminus i} \tanh \nu_{s,j}/2}.$$

2.3.2 LDPC Code Design

In this section, we consider the issue of LDPC code design. We begin with a discussion of how to specify a family of LDPC codes, and how to select a particular LDPC code from the family. Then, we discuss techniques to design those families to have favorable properties. We begin by describing degree distributions and how a pair of degree distributions specifies a family of LDPC codes. We then briefly discuss degree distribution design via density evolution and via EXIT charts.

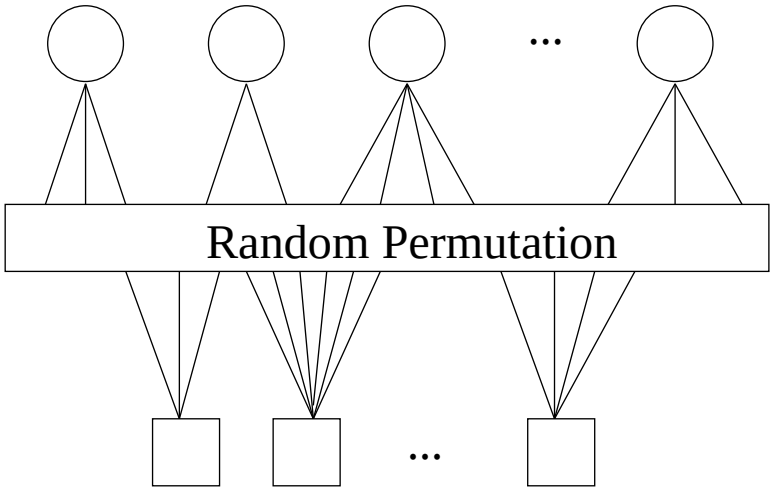


Figure 2.4. A graphical representation of a random irregular LDPC code. The degree of each node is determined according to a degree distribution. Each edge is then randomly connected to a variable node and a check node according to some random permutation.

As opposed to the specific code implied by the graph in Figure 2.3, we now consider a more general approach to LDPC codes as shown in Figure 2.4. Each of variable node and each check node has some number of edges emanating from it. We refer to number of nodes emanating from a node as its degree. We then can say that a degree distribution is the probability distribution on the number of edges emanating from the nodes. That is, in generating a code, the degree of each node is determined by drawing it from the degree distribution for that type of node. We denote the distribution over the variable and check nodes with λ and ρ respectively. Note that if the density of degrees for every node is constant across the variable nodes and is separately constant across all the check nodes, the code is said to be regular. Otherwise, the code is said to be irregular.

For convenience, we write the degree distributions in terms of their generating function

polynomials $(\lambda(x), \rho(x))$ of the following form

$$\lambda(x) = \sum_{i=2}^{d_{v,max}} \lambda_i x^{i-1}, \quad \rho(x) = \sum_{i=2}^{d_{c,max}} \rho_i x^{i-1}.$$

In the above, λ_i and ρ_i represent the fraction of edges that are connected to nodes with degree i, for the variable nodes and check nodes respectively. We define $d_{v,max}$ and $d_{c,max}$ as the maximum variable and check degrees respectively. For these numbers to represent distributions, the values of λ_i and ρ_i must sum to 1

$$\sum_{i=2}^{d_{v,max}} \lambda_i = 1, \quad \sum_{i=2}^{d_{c,max}} \rho_i = 1.$$

Since the number of edges emanating from the variable nodes must equal the number of edges emanating from the check nodes, when we specify the degree distribution pair for a code we also specify its channel coding rate

$$r(\lambda, \rho) = 1 - \frac{\int_0^1 \rho(x)\, dx}{\int_0^1 \lambda(x)\, dx}.$$

Here we have leveraged the convenience of representing the degree distributions with their generating functions. In the following we will discuss techniques for the design of good degree distributions.

When we specify a family of LDPC codes and draw a particular code, the question of selecting a "good" code arises. Fortunately, due to the concentration theorem of Richardson and Urbanke [70], we can be reasonably well assured that most codes in a "good" family are also "good" codes. In practice, we often ensure the quality of a particular code by "pruning" short loops. We search the code for the presence of short loops and remove (or prune) one of the edges to remove the short loop. Though LDPC codes are loopy by their very nature, in practice removal of length 4 short loops performs very well in practice [70].

Density Evolution

Density evolution, developed by Richardson and Urbanke [70], is an analysis tool to determine the average performance of a family of LDPC codes. Density evolution is an iterative process that begins by fixing a particular family (degree distribution pair) of LDPC codes and a parameterized channel model (discrete channels can always be parameterized). In each iteration, density evolution determines whether decoding will on average succeed or not on for a particular channel parameter. A binary search is then performed to determine the "worst" channel parameter over which the code can be expected to successfully

decode. Knowing the "worst" channel a degree distribution pair can decode over gives a way to evaluate the quality of a code. If we compare two families with equal rates, yet one can decode over "worse" channel parameters then it is the better code. Techniques for determining good code using density evolution is discussed in the papers [69, 15].

In this thesis, we do not use density evolution as a design tool. Instead, we use code families found by the Communications Theory Lab (LTHC) at Ecole Polytechnique Fédérale de Lausanne (EPFL) using density evolution and published online [4]. Since the corpus is constantly being improved, better code families appear regularly. The specific degree distributions used in this thesis can be found in Appendix A.

EXIT Charts

Extrinsic Information Transfer (EXIT) charts are another popular technique for LDPC code design. An EXIT chart is constructed by treating the bit nodes and check nodes as two separate decoders that feed their outputs to each other. The behavior of each of these two component decoders is assessed by measuring a statistic of the input and output data. In particular, we measure the average mutual information between the decoder estimates of the codeword and the true codeword, denoted I_A for the input and I_E for the output. By measuring this statistic on both the input and output of each of the component decoders, a characterizing curve for the two decoders is generated. The particular shape of these curves is dependent on the code's degree distributions, and the channel parameters. A sample plot of the curves for both the variable node decoder and the check node decoder are shown in Figure 2.5. To represent the input-output relationship of these two decoders, the curves are plotted on reflected axes. Readers are referred to the paper [7] for more detail about EXIT charts.

It was shown in by Ashikhmin, Kramer, & ten Brink [7] that as long as the variable node decoder curve lies above the check node decoder curve when plotted in the manner of Figure 2.5, then successful decoding, asymptotic in block length, is assured. Decoding can be modeled as following a path in the gap between the two curves. Thus if the two curves intersect, the path is impeded and decoding fails. It was further shown that the area of the gap between the two curves is proportional to the performance gap between the code and capacity. Thus the problem of code design is reduced to that of finding degree distributions pairs that minimize the gap between the two curves, conditioned on maintaining a positive gap. By fixing a series of check node degree distributions, a good pair of degree distributions

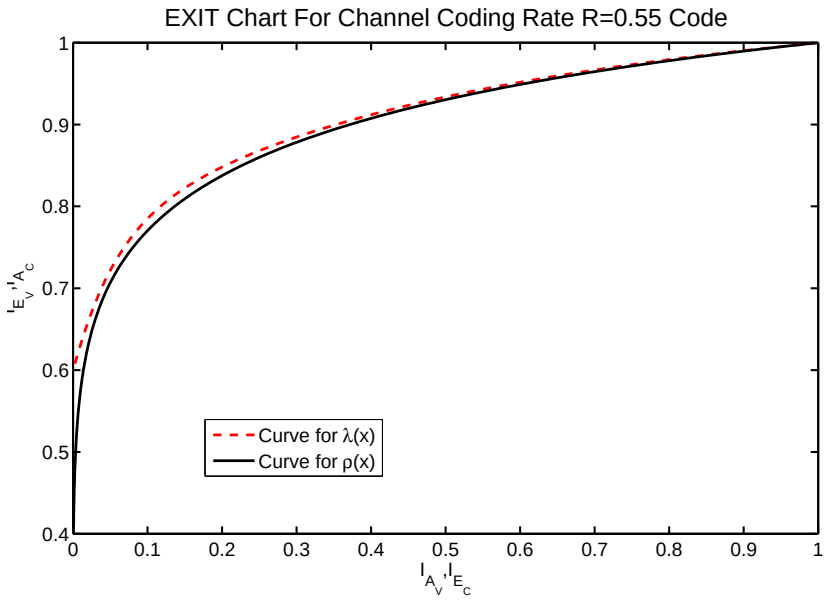


Figure 2.5. An example of a good code degree distribution pair for channel coding rate $R = 0.55$, designed using EXIT charts. The tight fit of the two curves in the plot shows that the code operates near to capacity. The dashed curve corresponds to the variable nodes while the solid curve corresponds to the check nodes. Decoding operates by moving from the left-hand side to the right-hand side of the graph, going through the gap between the two curves. Since the gap between the two curves is positive, decoding will follow the path between the curves successfully.

can be found with a short series of linear programs. In comparison to density evolution, obtaining good degree distributions is far less computationally intensive, but the results are unfortunately less accurate. As stated above, the specific degree distributions used in this thesis can be found in Appendix A.

2.4 Distributed Source Coding

In this section we discuss an important extension of the traditional point to point source coding problem discussed in Section 2.1. We reach this extension by considering a series of smaller extensions. We begin by considering the point to point compression of two sources, x^n and y^n, as shown in Figure 2.6. Here, the encoder is draws samples $\{(x_i, y_i)\}_{i=1}^n$ in an i.i.d. manner with distribution $P(x, y)$. Standard source coding theory says that achievable

rates are bounded by the following

$$R \geq H(x, y).$$

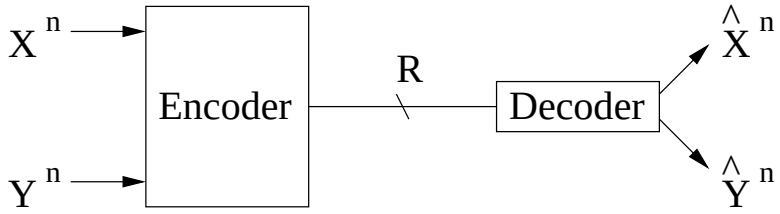


Figure 2.6. An extension of the traditional single source and single destination source coding problem to two sources, x^n and y^n. Here, the minimum rate according to information theory is the joint entropy of the two sources, i.e. $R \geq H(x, y)$.

In order to extend this scenario to distributed source coding, we consider the case of separate encoders for each source, x^n and y^n. Each encoder operates without access to the other source. This scenario is illustrated in Figure 2.7. Using the techniques of Section 2.1, we would expect that each source could only be compressed to its marginal entropy. The anticipated region would be as follows

$$R_x \geq H(x), \quad R_y \geq H(y).$$

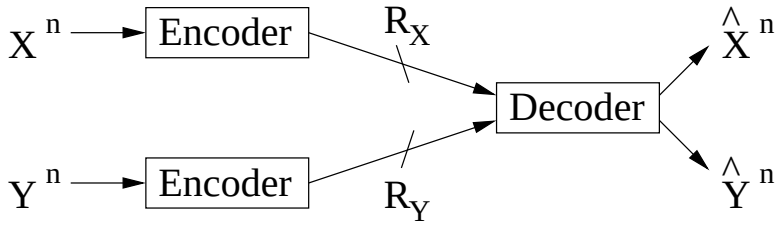


Figure 2.7. The block diagram for the distributed source coding problem. Here, without communication between the two, one encoder maps the source symbols x^n to a bitstream while the other maps y^n to a separate bitstream. The decoder receives both bit streams and reconstructs $\hat{x}^n$ and $\hat{y}^n$. The minimum rate region for R_x and R_y is given in Equation (2.2).

Surprisingly, the work of Slepian and Wolf [78] shows that the achievable rate region is much larger. Specifically, their work shows that the region is bounded by the three inequalities in Equation (2.2),

$$R_x \geq H(x|y), \quad R_y \geq H(y|x), \quad R_x + R_y \geq H.(x, y) \tag{2.2}$$

The achievable rate region is shown graphically in Figure 2.8. Stated simply, the Slepian-Wolf theorem shows that their is no theoretical loss in compression performance (with respect to the sum-rate) between Figure 2.7 and Figure 2.6.

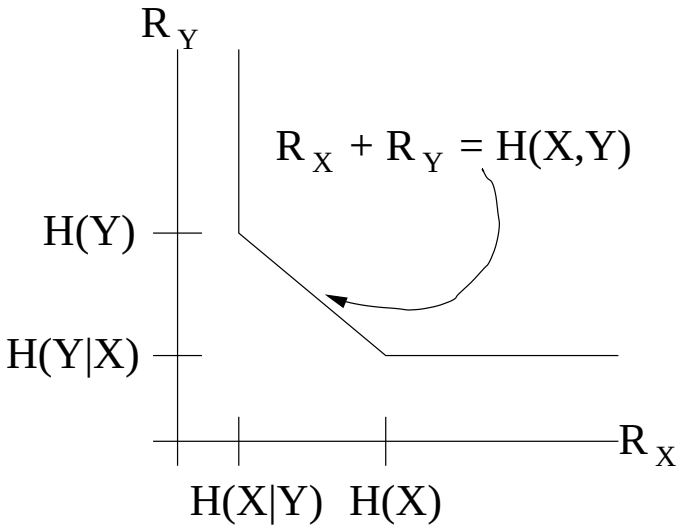


Figure 2.8. A graphical representation of the Slepian-Wolf rate region as specified in Equation (2.2).

An important special case of distributed source coding is called source coding with side information. Also referred to as corner point coding (since the operating point is one of the corner points of the Slepian-Wolf rate region in Figure 2.8), a block diagram is shown in Figure 2.9. In this figure, the source y^n is sent directly to the decoder, but is unavailable to the encoder[2] of x^n. The goal in this scenario is to leverage the availability of y^n at the decoder despite the encoder only having access to x^n. From Equation (2.2), we see that the achievable rate is as given here

$$R_x \geq H(x|y).$$

Though not a focus of this thesis, lossy source coding with side information has been studied extensively in the literature. Wyner and Ziv [89] establish the rate-distortion tradeoff for this scenario. They show that for Gaussian sources, the rate distortion bound for source coding with side information is equivalent to the rate distortion bound for the scenario when the side information is made available to the encoder as well as the decoder. Generally, the achievable rate distortion tradeoff is diminished from the scenario of the encoder having full access to the side information.

[2]We implicitly assume that y^n has been transmitted losslessly to the decoder at rate $H(y)$.

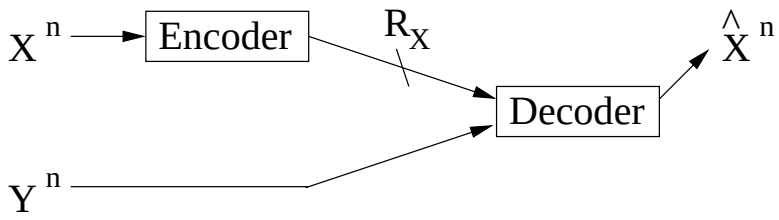


Figure 2.9. A special case of distributed source coding referred to as source coding with side information. Here, the correlated source y^n is available at the decoder, but is unavailable at the encoder. Using the Slepian-Wolf rate region of Equation (2.2), we see that the achievable rate is bounded as $R_x \geq H(x|y)$.

The Slepian-Wolf theorem was essential for establishing the potential benefits of distributed source coding. Unfortunately, the Slepian-Wolf theorem is asymptotic and non-constructive. In the chapters that follow, we discuss how to construct practical codes to achieve the Slepian-Wolf bound.

Chapter 3

A Practical Construction for Source Coding With Side Information

We begin our discussion of practical constructions for distributed source coding in this chapter by focusing on the special case of source coding with side information. As discussed in Section 2.4, source coding with side information is a special case of the general distributed source coding problem. The codes developed in this chapter are able to leverage the correlation between two sources even without direct access to the data itself. The development of these codes is crucial to the development of codes for the fully general distributed source coding problem, and will be used as a building block for the rest of the work in this thesis.

3.1 Introduction

Most proposed frameworks for source coding with side information to date consider only a limited correlation structure. Such restricted structure is convenient for analysis, but is an unrealistic assumption for scenarios of practical interest. A comprehensive solution requires

a single framework flexible enough to operate on *arbitrary correlation* structures, including the special case of *no correlation*[1].

As discussed in Section 2.4, Slepian and Wolf [78] gave information theoretic bounds on lossless compression of two correlated sources with a common decoder. The Slepian-Wolf result is however asymptotic and non-constructive. We summarize here some of the related literature towards a practical construction for source coding with side information. A more complete discussion of practical efforts towards distributed source codes can be found in Section 5.1.1. One early approach to practical codes is Wyner's [90] formulation of a linear coset code for source coding with side information (assuming binary sources and symmetric correlation). Wyner's work further establishes the relationship between distributed source coding and channel coding. The late 1990s work of Zamir & Shamai [92] shows the theoretical feasibility of a lattice code framework for a lossy version of this problem (first considered by Wyner & Ziv [89] in the information theoretic setting). Following that, Pradhan & Ramchandran [66] developed DISCUS, a framework for the Wyner-Ziv problem and described practical codes and achievable error probabilities. Recently, a flurry of work in the field provides practical constructions of various levels of sophistication for the Slepian-Wolf problem [66, 80, 83, 30, 95, 34, 31, 32, 46, 49, 50, 76, 56, 54, 2]. However, the constructions to date in the literature do not tackle the general source coding with side information problem for arbitrary sources. While the scenarios considered in these works are important operating regimes, they fail to cover the complete spectrum of diverse real-world sources and application scenarios requiring more flexible operating points.

In this chapter, we develop a practical and constructive framework for source coding with side information. A block diagram is shown in Figure 3.1. This construction is a foundation for our solution to the arbitrary distributed source coding problem (Chapter 5), the compression of encrypted data (Chapter 7), and the work throughout this dissertation. We provide our solution focusing on binary sources, though it is applicable to larger alphabets. We relate the source coding with side information problem to channel coding and show how to apply Low-Density Parity-Check codes to generate powerful codes for source coding with side information. We provide simulation results that demonstrate how this construction performs when applied to arbitrarily correlated binary random sources. Our algorithm is powerful enough to approach the Slepian-Wolf bound.

This chapter is organized as follows. In section 3.2 we discuss a solution for source

[1] In the case of 2 sources with no correlation between them, the problem of distributed source coding reduces to two separate traditional (single source, single destination) source coding problems.

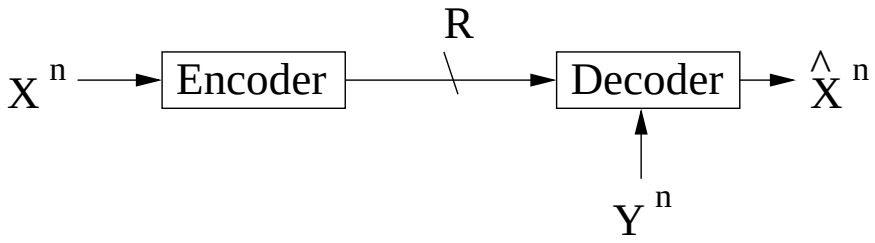


Figure 3.1. The source coding with side information problem. The encoder seeks to compress x given that y is available at the decoder but unavailable to itself.

coding with side information based on linear block codes. Then, in section 3.3, we present again a short summary of LDPC codes, a powerful class of linear block codes. Section 3.4 addresses how LDPC codes can be adapted to our source coding with side information solution and how the algorithm can be successful for arbitrary x, y correlations. Section 3.5 provides results and some related performance analysis. Finally, Section 3.6 concludes the chapter and discuses some open problems.

3.2 Linear Block Codes for Source Codes With Side Information

In this section, we discuss a code construction for source coding with side information. We base this discussion on the work of Pradhan and Ramchandran [66]. For clarity, we focus on binary alphabets. Extension to larger alphabets is largely straightforward, but is not discussed here. We focus on compressing, x, given that side information, y, is available at the decoder. We begin within a simple, yet illustrative example. This example is then generalized to a framework for source coding with side information.

In information theoretic approaches, random binning (hashing) is a widely used technique [19] for compressing a data sequence. When binning, the space of length-n sequences is broken up into 2^{nR} bins, each of size $2^{n(1-R)}$. Compression is achieved by sending bin indices, using nR bits each. Random binning is used in the proof of the Slepian-Wolf theorem (and entropy coding) [18, 19]. So long as the number of bins used satisfies the Slepian-Wolf bounds, the sources can be recovered with high probability using a decoder operating on "joint typicality" principles.

Key to a practical solution for source coding with side information is to use structured binning. Random binning is asymptotic, unstructured, and as such is intractable for practical implementation. Csiszár [20] showed that linear bins (codes) can be used. To leverage practical structured codes, it is useful to view the two sources as related via a "virtual" channel. In this channel, with input x and output y, z is seen as the noise pattern induced by the "channel"

$$y = x \oplus z. \tag{3.1}$$

We have assumed an additive structure here as it is convenient and suffices for binary channels. In order to accommodate arbitrary joint distributions, we do not assume independence between[2] x and z.

We thus use a linear block code $\mathcal{C}$ for binning. The bin index, denoted s^m, of a codeword x^n, is the word's syndrome with respect to $\mathcal{C}$'s parity check matrix

$$s_x = x^n \mathbf{H}_x^T. \tag{3.2}$$

Thus, we encode x^n to s_x via matrix multiplication. An unfortunate result of this approach is that it induces a source of error. To account for atypical source behavior, standard source coders allow variable length codewords. By using fixed rate linear block codes, atypical realizations of the source inevitably induces errors. Fortunately, as we consider larger block lengths such sequences become unlikely.

3.2.1 Illustrative Example

As an example [66], let x and y be two uniform 3-bit sources. They are correlated such that they differ in *at most* 1 of their 3 bits (i.e. $wt_H(x \oplus y) \leq 1$). We calculate the following: $H(x) = H(y) = 3$ bits, $H(x|y) = H(y|x) = 2$ bits, and $H(x, y) = 5$ bits.

With direct access to y, it is easy for the encoder to convey x to the decoder using only 2 bits. In this case the encoder determines which of the difference patterns between x and y has occurred,

$$z \in \{000, 001, 010, 100\}. \tag{3.3}$$

Since there are only 4 possible values for z, the encoder needs only 2 bits to specify which of the 4 it is. The decoder can then reconstruct x by calculating $\hat{x} = y \oplus z$.

[2] E.g. consider the binary non-symmetric channel; here the crossover probabilities are dependent upon the channel input.

When the encoder does not have access to y though, according to the Slepian-Wolf theorem, the encoder still only needs 2 bits to transmit x. To achieve this, we note that given any particular sequence y, there are only 4 possible values for x based on Equation (3.3). To leverage this we create the following cosets such that for any given y only one of the possible values of x lies in each set.

$$\begin{aligned} &\left\{ \begin{array}{ccc} 0 & 0 & 0 \\ 1 & 1 & 1 \end{array} \right\} \Rightarrow 00 \quad \left\{ \begin{array}{ccc} 0 & 0 & 1 \\ 1 & 1 & 0 \end{array} \right\} \Rightarrow 01 \\ &\left\{ \begin{array}{ccc} 0 & 1 & 0 \\ 1 & 0 & 1 \end{array} \right\} \Rightarrow 10 \quad \left\{ \begin{array}{ccc} 0 & 1 & 1 \\ 1 & 0 & 0 \end{array} \right\} \Rightarrow 11. \end{aligned} \tag{3.4}$$

To compress x to 2 bits, we need only specify which one of these 4 cosets it occurs in. With knowledge of y and the coset, the decoder can uniquely determine the value of x.

To compress x to 2 bits, in effect we are using the $(3, 1)$-repetition code to define the parity check matrix $\mathbf{H}$ in Equation (3.2)

$$\mathbf{H} = \left[\begin{array}{ccc} 1 & 0 & 1 \\ 0 & 1 & 1 \end{array} \right].$$

Key to the success of this code is that the $(3, 1)$-repetition code has a minimum distance of 3. This minimum distance is sufficient to isolate all sequences with a 1 bit difference ($wt_H(x \oplus y) \leq 1$) from any particular y. Thus knowledge of y and s_x allows the decoder to determine x exactly.

3.2.2 Block Codes For Source Coding With Side Information

In the example in Section 3.2.1, the $(3, 1)$-repetition code is successful because it is strong enough to handle the channel of corresponding strength to the correlation model considered. More generally, we need to use a linear block channel code strong enough to reach the capacity of the corresponding correlation channel. In this section, we define the code in terms of its generator matrix. As stated above, compression is achieved by finding the syndrome of the source word with respect to the parity check matrix corresponding to the generator matrix defined. Recall the correlation model of Equation (3.1).

As an example, we assume that z is independent of y, and that y is a sequence of Bernoulli-$(1/2)$ random variables while z is a sequence of Bernoulli-(Q) random variables.

The Slepian-Wolf theorem says we can compress x to $R_x = H(x|y) = H_2(Q)$. The channel corresponding to this correlation structure is a binary symmetric channel (BSC) with crossover parameter Q. Thus, a channel code designed for a BSC with parameter Q could be used to successfully bin x for compression. We apply this principle in general to the correlation structure of x and y.

For the rest of this section, we discuss a ML (Maximum Likelihood) decoding algorithm for linear block codes applied to source coding with side information. We assume that the code has been selected for the correlation as appropriate. Though the algorithm presented here is impractical (due to the use of a ML decoder), an implementable (but sub-optimal) algorithm is given in Section 3.4. We give pseudo-code for our decoding algorithm below.

In the first step of the pseudo code, we write the source as a function of the compression code and the compressed value. We use the vector u_x, to select which codeword is necessary to satisfy these equations. We denote the closest codeword x^n in $\mathcal{C}_x$ as $u_x\mathbf{G}_x$.

In Step 1, $\bar{s}_x$ represents a zero-padded version of the value s_x. $\bar{s}_x$ is the element in the co-sets formed by $\mathcal{C}_x$ of minimum Hamming weight. To zero pad, insert zeros so that they align with the systematic bits of the code[3]. For example, consider the code defined by the $(7, 4)$-Hamming code in Equation (3.5). Since the third through sixth columns form the systematic bits, zeros are inserted into the fourth and fifth positions.

$$\text{If } \mathbf{G}_x = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \text{ and } s_x = [\overbrace{1\,0}\ \overbrace{1}], \tag{3.5}$$
$$\text{then } \bar{s}_x = [\overbrace{1\,0}\ 0\,0\,0\,0\ \overbrace{1}].$$

In step 2, we decode the quantity $[\bar{s}_x \oplus y^n]$ with respect to $\mathbf{G}_x$ to give $\hat{z}$ and u_x. Finally, in step 3, combine y^n and $\hat{z}^n$ to recover $\hat{x}^n$.

Pseudo code for the decoding algorithm is provided here:

1. Expand sources equation: $x^n = u_x\mathbf{G}_x \oplus \bar{s}_x$ and $y^n = x^n \oplus z^n = u_x\mathbf{G}_x \oplus \bar{s}_x \oplus z^n$.
2. ML Decode $\hat{z}^n = u_x\mathbf{G}_x \oplus [\bar{s}_x \oplus y^n]$.
3. Recover $\hat{x}^n = y^n \oplus \hat{z}^n$ using knowledge of $\hat{z}^n$ and y^n.

[3]More generally, we need select any k_x linearly independent columns of $\mathbf{G}_x$ since bit ordering does not impact code performance in the memoryless case.

This decoding prescription reinforces our insights that the Slepian-Wolf code design is really a channel coding problem, since Step 2 requires a channel decoder[4]. We leverage this insight to the design requirements of $\mathcal{C}$. This code needs be from a "good" family of codes ("good" if the probability of a decoding error goes to zeros as block length goes to infinity). If the code used is bounded away from capacity, its application to distributed source coding will be correspondingly bounded away from the Slepian-Wolf bound. Using random codes, code design randomly generates $\mathbf{G}_x$ to bin x^n. This constraints can be approached with large block lengths and pseudo-random codes [20]. Though we do not prove that the block codes construction here achieves the Slepian-Wolf limit, we demonstrate it via empirical methods (Section 3.5).

3.3 Low-Density Parity-Check Codes

In this section, we highlight some of the background for LDPC (Low-Density Parity-Check) codes. The reader is referred to Section 2.3 for further details. Low-Density Parity-Check codes are a class of powerful linear block codes and thus a natural choice for our distributed source coding framework. We use factor graphs [42] (as in Figure 2.3) to visually represent LDPC codes. These bipartite graphs consist of circles used to represent bits and squares used to represent constraints.

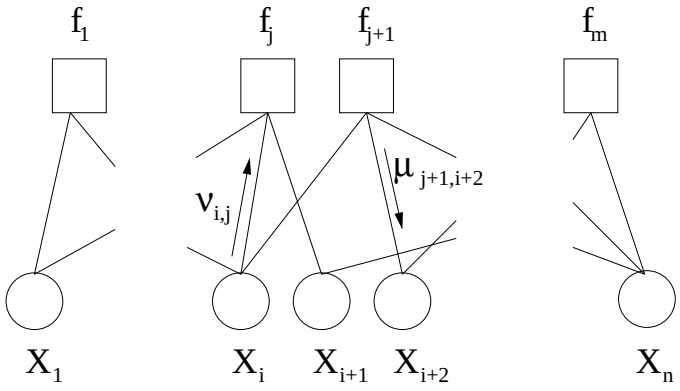


Figure 3.2. A sample factor graph. The circles represent bits while squares represent the parity constraints of the code. A labeling of the two types of messages passed from node to node in the decoding algorithm is given.

[4]A formal derivation of the Slepian-Wolf error exponent from the random coding error exponent is given in Appendix B.

Decoding of LDPC codes is achieved using the sum-product algorithm [26, 61, 42]. The algorithm is an iterative inference algorithm that though exact on trees, is known to have good (but sub-optimal) empirical performance on loopy graphs such as those of an LDPC code. The sum-product algorithm operates by iteratively estimating the marginal distribution for each bit. In the first phase of each iteration, each variable node combines the estimates available and sends them along its edges (e.g., $\nu_{i,j}$ in Figure 3.2). In the second phase of each iteration, these estimates are merged, according to the constraint, at each check node into for each bit node (e.g., $\mu_{j+1,i+2}$ in Figure 3.2). The marginals are used to estimate the bits after each iteration, if below a maximum number of iterations, the set of estimates is checked against a stopping condition.

3.4 Practical Algorithm

In this section we combine the LDPC codes of Section 3.3 with the linear block coding construction of Section 3.2. We first describe the graphical model for source coding with side information. We then show how this model gracefully degrades to entropy coding. Practical decoding is achieved via the sum-product algorithm over the resulting graph. Since this graph is loopy, sum-product is sub-optimal but is empirically good. We assume knowledge of the source statistics at the decoder. Discussion of blind operation is considered in Chapter 6.

The graphical model for source coding with side information is shown in Figure 3.3. The variables considered are the n bits of sources x and y (middle and bottom rows of circles respectively), labeled x_i and y_i, and the m_x bits of the compressed source x (top row), labeled s_{x_j}. These variables are constrained by the LDPC codes used to compress x (top row of squares), labeled f_{x_j}, plus the correlation between sources x and y (bottom row), labeled f_i. The correlation constraints between the sources x and y are distinct and different from the parity constraints considered in Section 3.3.

This model reduces gracefully to an entropy code as a special case of source coding with side information. Whenever side information is either not present or is uncorrelated with the source, the problem reduces to single source and single destination source coding. Fortunately, the graphical model of Figure 3.3 gracefully degrades to the graphical model for entropy coding in Figure 3.4. In this graph, the variable nodes for the second source y are removed. In this graph, the correlation constraints f_i now represent source constraints.

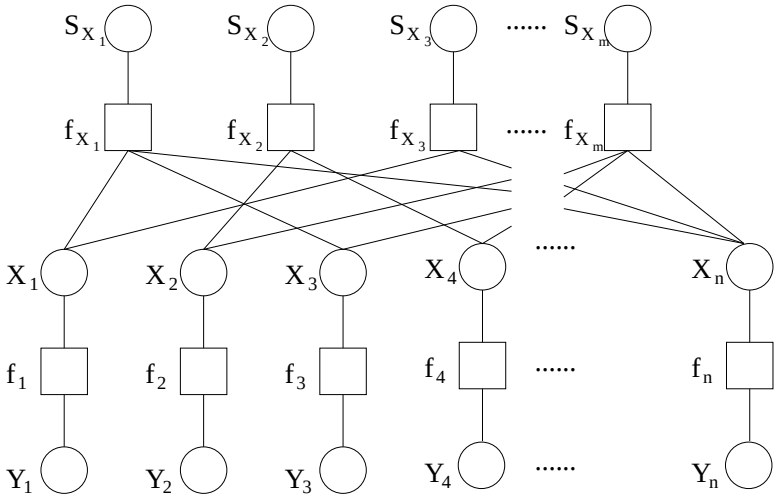


Figure 3.3. The factor graph for source coding with side information. Circles labeled x_i or y_i represent source bits. Circles labeled s_{x_j} represent the compressed bits. Squares labeled f_{x_j} represent the LDPC code applied to the source. Squares labeled f_i represent the correlation between bits x_i and y_i.

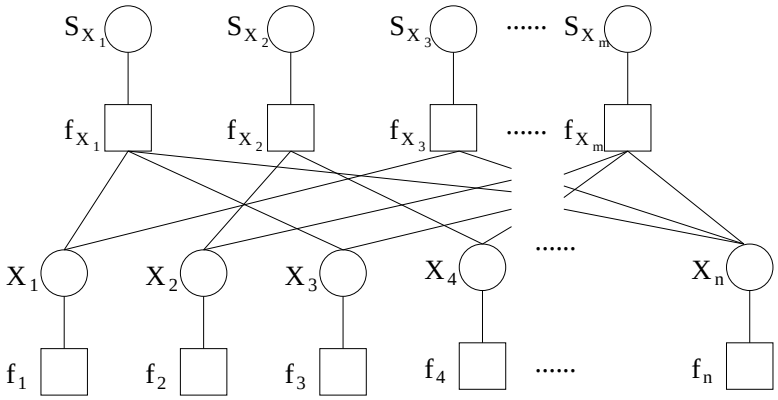


Figure 3.4. The factor graph for entropy coding. Circles labeled x_i represent source bits. Circles labeled s_{x_j} represent the compressed bits. Squares labeled f_{x_j} represent the LDPC code applied to the source. Squares labeled f_i represent the source constraint on x_i. This model develops as a graceful degradation of the source model of Figure 3.3 in the absence of correlated side information.

3.4.1 Application of Sum-Product Algorithm

We present the message update equations here. We first describe the update equations, and then summarize with pseudo code. Throughout this section, take $i \in \{1, \ldots, n\}$ and

$j \in \{1, \ldots, m_x\}$. The bits of sources x are labeled x_i, the compressed bits are labeled s_{x_j}, the LDPC constrains are labeled f_{x_j}, and the correlation constraints are labeled f_i. Since we consider bits, messages are equal to the ratio of the two probabilities. We use μ and ν to denote messages to and from variable nodes respectively (as in Figure 3.2) and subscripts indicate the source and destination respectively. For example, the form of the message from the check node f_i to variable node x_i is below

$$\mu_{f_i,x_i} = \log P(x_i = 0)/P(x_i = 1).$$

Messages are initialized using $P(x, y)$.

Since the compressed bits are known and are connected to only one other node, the messages passed to them are insignificant. Their output messages are their observed value. The messages passed to and from the LDPC constraint node f_{x_j} and the source bit x_i are in Equation (3.6) and are based on the formulas in Section 2.3.1. Here $\mathcal{N}(i)$ is the set of nodes connected to node i and $\mathcal{N}(i)\backslash j$ is $\mathcal{N}(i)$ less node j. These updates are similar to those used for channel coding, modified to include the compressed bits;

$$\begin{gathered} \mu_{f_{x_j},x_i} = (-1)^{s_j} \log \frac{1+\prod_{x_k \in \mathcal{N}(f_{x_j})\backslash X_i} \tanh \nu_{x_k,f_{x_j}}/2}{1-\prod_{x_k \in \mathcal{N}(f_{x_j})\backslash X_i} \tanh \nu_{x_k,f_{x_j}}/2} \\ \text{and } \nu_{x_i,f_{x_j}} = \mu_{f_i,x_i} + \textstyle\sum_{f_{x_k} \in \mathcal{N}(x_i)\backslash f_{x_j}} \mu_{f_{x_k},x_i}. \end{gathered} \tag{3.6}$$

Finally we consider the correlation constraint updates in Equation (3.7). We ignore the message sent to the variables y_i since they are terminal nodes. The messages updates in Equation (3.7) convert the likelihood of y_i to the likelihood of x_i based on the correlation between the sources

$$\mu_{f_i,x_i} = \log \frac{(2-P(x_i=1|y_i=0)-P(x_i=1|y_i=1))+(P(x_i=1|y_i=1)-P(x_i=1|y_i=0))\tanh \nu_{y_i,f_i}/2}{(P(x_i=1|y_i=0)+P(x_i=1|y_i=1))+(P(x_i=1|y_i=0)-P(x_i=1|y_i=1))\tanh \nu_{y_i,f_i}/2}. \tag{3.7}$$

We summarize the full algorithm with the pseudo code below.

1. Initialize messages.
   x_i nodes: $\nu_{x_i,f_i} = \log P(x_i = 0)/P(x_i = 1)$,
   and $\nu_{x_i,f_{x_j}} = \log P(x_i = 0)/P(x_i = 1)$.
   s_{x_j} nodes: $\nu_{s_{x_j},f_{x_j}} = s_{x_j}$.
   y_i nodes:
   $$\nu_{y_i,f_i} = \log P(y_i = 0)/P(y_i = 1) = \begin{cases} +\infty & \text{if } \ y = 0 \\ -\infty & \text{if } \ y = 1 \end{cases}.$$

2. Update messages sent by constraint nodes.

   f_{x_j} nodes:

$$\mu_{f_{x_j},x_i} = (-1)^{s_j} \log \frac{1 + \prod_{x_k \in \mathcal{N}(f_{x_j}) \setminus X_i} \tanh \nu_{x_k,f_{x_j}}/2}{1 - \prod_{x_k \in \mathcal{N}(f_{x_j}) \setminus X_i} \tanh \nu_{x_k,f_{x_j}}/2}.$$

   f_i nodes:

$$\mu_{f_i,x_i} = \log \frac{(2-P(x_i=1|y_i=0)-P(x_i=1|y_i=1))+(P(x_i=1|y_i=1)-P(x_i=1|y_i=0))\tanh \nu_{y_i,f_i}/2}{(P(x_i=1|y_i=0)+P(x_i=1|y_i=1))+(P(x_i=1|y_i=0)-P(x_i=1|y_i=1))\tanh \nu_{y_i,f_i}/2}.$$

3. Update messages sent by variable nodes.

   x_i nodes:

$$\nu_{x_i,f_{x_j}} = \mu_{f_i,x_i} + \sum_{f_{x_k} \in \mathcal{N}(x_i) \setminus f_{x_j}} \mu_{f_{x_k},x_i},$$

   and $\nu_{x_i,f_i} = \sum_{f_{x_k} \in \mathcal{N}(x_i)} \mu_{f_{x_k},x_i}$

   s_{x_j} nodes: Unchanged from Step 1.

   y_i nodes: Unchanged from Step 1.

4. Determine best estimate of x^n.

$$\hat{x}_i = \begin{cases} 0 & \text{if} \quad \mu_{f_i,x_i} + \sum_{f_{x_k} \in \mathcal{N}(x_i)} \mu_{f_{x_k},x_i} \geq 0 \\ 1 & \text{if} \quad \text{otherwise} \end{cases}.$$

5. Check stopping condition.

   If the maximum number of iterations is met, quit and return $\hat{x}$.

   Else if $s_x = \hat{x}^n \mathbf{H}_x^T$, quit and return $\hat{x}$.

   Otherwise, return to Step 2.

In the following section, we give the results of simulations using this algorithm.

3.5 Simulations and Results

In this section, we present simulation results. We measure the quality of a simulation with its empirical probability of error, the proportion of bits decoded in error. Since out scheme is asymptotically lossless, we make comparisons with the lossless bounds of Slepian-Wolf despite our measured probabilities of error. Though we focus on BER (bit error rate), we also consider SER (symbol error rate, a symbol error occurring whenever any one or more bits of a block are in error). Bit Error Rate is a more appropriate measure for source coding problems since each reconstructed bit is of use to the receiver. In channel coding

$P(x,y)$	$y=0$	$y=1$
$x=0$	$(1-\gamma)(1-\alpha)$	$(1-\gamma)\alpha$
$x=1$	$\gamma\beta$	$\gamma(1-\beta)$

Table 3.1. Structure of the joint distribution between x and y considered for Section 3.5.

for contrast, SER is the appropriate since not one source bit can be recovered reliably when the symbol is in error. We observed that in our simulations empirical SER and BER measures resulted in *very* similar behavior. We present both SER and BER results for our first experiments, but omit them after this demonstration (Figures 3.5 and 3.6).

Each simulation consisted of selecting a particular code, source distribution, and rate combination. Each empirical probability is generated by simulation of *at least* one million bits, broken into blocks. We limited decoding to at most 50 iterations and used LDPC code designs as described in Appendix A.

We consider arbitrary distributions over two correlated binary sources, parameterized via $0 \leq \alpha, \beta, \gamma \leq 1$. The distribution considered is $P(x=0, y=0) = (1-\gamma)(1-\alpha)$, $P(x=0, y=1) = (1-\gamma)\alpha$, $P(x=1, y=0) = \gamma\beta$, and $P(x=1, y=1) = \gamma(1-\beta)$. We give a diagram of this in Table 3.5.

3.5.1 Compression With Side Information Results

We present simulation results of source coding with side information in this section. In each simulation, we assume y is available perfectly at the decoder (and has been transmitted at $R_y = H(y)$) and $R_x = 0.5$. In the following plots, we plot the minimum rate (conditional entropy) on the horizontal axis and the resulting probability of error on the vertical axis. For example, consider a block of $n = 100,000$ bits from a source with $(\alpha, \beta, \gamma) = (0.1, 0.1, 0.5)$, $H(x) = H(y) = 1$, and $H(x|y) = 0.47$. This example results in a plot point of $(0.47, 4\times10^{-3})$ for the BER.

Initially, we restricted the joint distribution such that $\alpha = \beta$ and $\gamma = 0.5$. The results consider a variety of block lengths; 10^3, 5×10^3, 10^4, 2×10^4, 5×10^4, and 10^5. The results are plotted in terms of BER in Figure 3.5 and SER in Figure 3.6 (the results for block lengths 5×10^4 and 10^5 are omitted). The results improve for both BER and SER as the number of bits used above the Slepian-Wolf minimum and as block length grows. Note that for these two plots we give single standard deviation error bars. The error bars for the other plots in this chapter are similar to those of the BER result plot of Figure 3.5.

Figure 3.7 plots the results obtained after considering arbitrary joint distributions, for a block length of 10^4. Points are grouped in terms of the marginal entropy of source x, and plotted in terms of the minimum rate. Performance improves as the gap from the Slepian-Wolf bound grows, similar to the results presented in Figure 3.5.

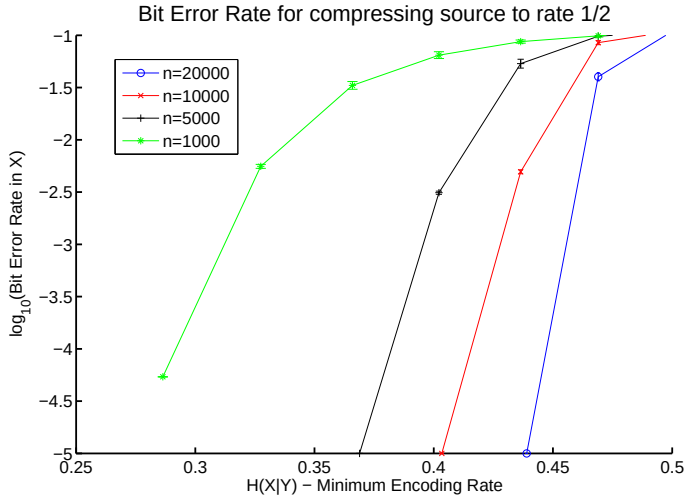


Figure 3.5. Source coding with side information simulation results for several block lengths and varying the correlation parameter $\alpha(=\beta)$. Source x is compressed to rate 1/2 while source y is available at the decoder. Each data point is the result of the simulation of at least 10^6 bits. The results in this figure are plotted in terms of their bit error rate (BER). In addition, we plot one standard deviation error bars. Similar error bars result for all BER results in this chapter, but are omitted for clarity.

3.5.2 Compression of a Single Source

The results from compression of a single source are presented in Figure 3.8. The source entropy is varied and tests are run for several block lengths; 10^3, 5×10^3, and 10^4. The results similar to those obtained above for source coding with side information (see Figure 3.5). Performance improves with growth in both the code rate used above entropy and with the block length.

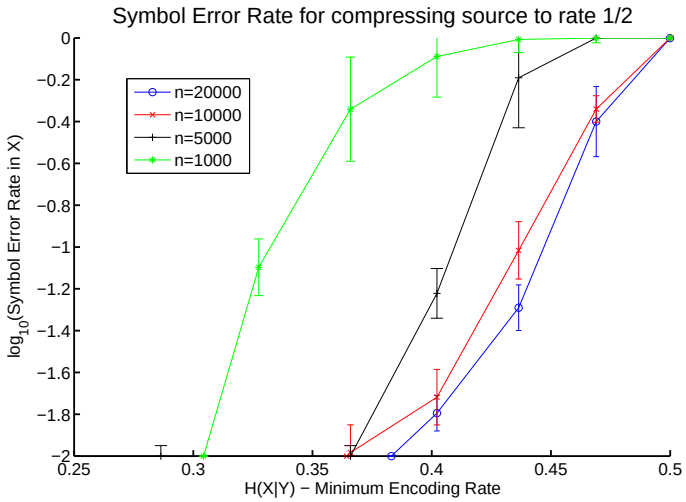


Figure 3.6. Results of the same experiments as in Figure 3.5 plotted in terms of the symbol error rates (SER). Each value is the result of the simulation of at least 10^3 symbols (blocks). In addition, we plot one standard deviation error bars.

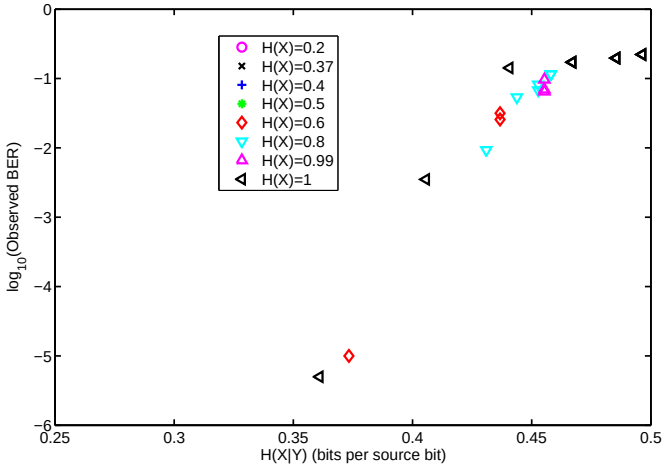


Figure 3.7. Simulation results compressing source x to rate $1/2$ while source y is available at the decoder. The simulations are of source coding with side information results, as in Figure 3.5. Each data point is the result of the simulation of at least 10^6 bits. The block length is fixed to 10,000 and a more general correlation structure is considered. The type of data point used indicates the marginal entropy of the source x.

3.6 Conclusions

This chapter has provided a practical solution for the problem of source coding with side information. We began by providing a general linear block coding solution and then

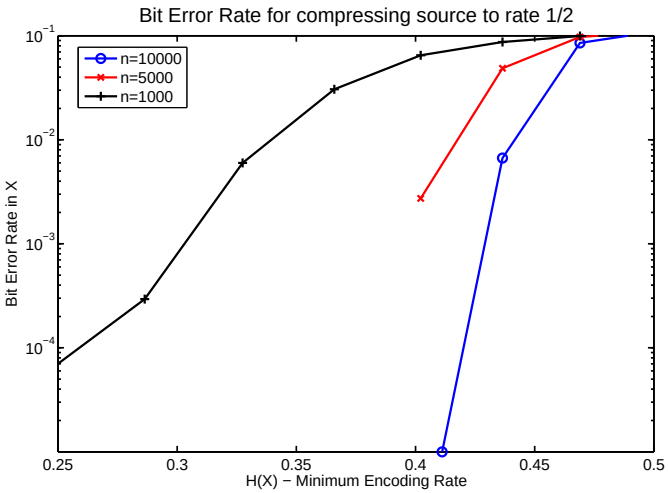


Figure 3.8. Simulation results from entropy coding source x to rate $1/2$. This plot considers single source results for several block lengths and source statistics. Each data point is the result of the simulation of at least 10^6 bits. Resulting coding performance is similar to that obtained when source coding with side information as in Figure 3.5.

specialized to LDPC codes. The simulation results demonstrated the strength of this construction. In the following chapters we build upon the construction presented here. In the next chapter, we extend the construction presented here to source coding with side information for block stationary sources.

Chapter 4

Slepian-Wolf Coding for Parallel Sources: Design and Error-Exponent Analysis

In this chapter we extend the codes developed in the last chapter to parallel sources. Most of the constructions proposed for distributed source coding are designed for independent identically distributed random source models, which fail to capture the high degree of correlation inherent in real-world sources like image and video signals. Motivated by its success in the modeling of real-world sources, in this chapter we invoke a parallel source model for the distributed source coding problem. The traditional way to deal with parallel source models has been through the so-called water-filling prescription of using multiple shorter codes, each matched to one of the source components. Our main contribution in this chapter is the proposal of a single long-block-length distributed source code that takes into account the parallel nature of the source. We first present an information-theoretic analysis of the gains made possible by this approach by using the well-developed theory of error exponents. More significantly, our study exposes a new code design problem which we describe for an LDPC framework. We show simulation results to validate our design.

4.1 Introduction

The distributed source coding problem (and the special case of source coding with side information) is finding application in a growing variety of areas. As previously discussed, these applications range from dense sensor networks to robust video transmission for wireless multimedia applications [68], to compression of encrypted data [38]. Unfortunately, the analysis and coding prescriptions for distributed source coding has initially been limited to the case of independent and identically distributed (i.i.d.) source and side-information correlation models. More importantly, recent practical code construction methods for the distributed source coding problem, starting from [64] to more recent state-of-the-art extensions [2, 94, 48, 14, 16, 79] (as discussed further in Chapter 5) have almost exclusively focused on the i.i.d. source/side-information correlation models.

On the other hand, real-world image and video sources, that have been suggested as promising application ground for distributed source coding, are characterized by a high degree of correlation/memory, and thus differ significantly from i.i.d. models. As has been witnessed in the success of block-based transform coding for contemporary image and video compression standards [1, 17, 85], parallel/mixture source models better describe real world sources. This motivated the work in [62], extending the above-mentioned information theoretic analysis to the case of colored source/side-information correlation models. Though this work has implications for both lossy and lossless coding problems, in the interests of clear explanations, in this chapter we consider only the lossless Slepian-Wolf coding of an idealized source model.

Parallel sources have been examined in the context of Slepian-Wolf design before. In particular, the standard approach to the design of Slepian-Wolf codes for a non-white source is to transform it into a number of uncorrelated parallel sources, and then code these residual sources independently. In this chapter we propose to code across such parallel sources, which can result in significant gains when the number of samples of each source is relatively small. To understand where the gains come from in this strategy, we turn to the underlying information-theory of the problem.

As is already well-understood [90, 64], Slepian-Wolf code design is really a channel coding problem (as is discussed in Section 3.2)[1]. Information theorists have also long understood that independent coding across parallel channels is capacity-achieving. Thus, at first blush,

[1] A formal derivation of the Slepian-Wolf error exponent from the random coding error exponent is given in Appendix B.

one might think there is nothing to gain from coupling the code design across sources. However, what is also well-understood from an information-theoretic perspective, is that while not helping from a capacity perspective, achieving capacity with a low probability of decoding error requires a long block-length. Thus, in applications where the block-lengths of each of the parallel channels is small we can significantly reduce the probability of decoding error by coding across the parallel channels because the resulting block-length is the sum of the individual block-lengths. Alternately, we can achieve a target reliability at a higher communication rate.

Because Slepian-Wolf coding is a channel coding problem, these ideas bear directly on the design of Slepian-Wolf codes for parallel sources. In particular, in many media applications, sources are very non-stationary. Therefore, the block-lengths of each of the parallel sources can be quite short. Hence, when designing Slepian-Wolf codes for parallel sources, each of which has a short block-length, we improve reliability by coding across the sources. Or, alternately, we can achieve a target reliability at a lower encoding rate.

It is in this context that we propose a novel coding strategy - the main contribution of this chapter, which offers a single long block-length code that accounts for the parallel nature of the source/side-information correlation model. This approach exposes a new code design problem, for which we present a practical solution for based on the Low-Density Parity-Check code (LDPC) framework [26]. We focus on the source coding with side information problem here, and build on the solution presented in Chapter 3.

This chapter is organized as follows. In Section 4.2, we present a error-exponent based theoretical analysis that quantifies the performance gains of coding across parallel sources. In Section 4.3, we present the new code design problem in the LDPC framework. In Section 4.4 we provide experimental results to validate the proposed code design. Finally, Section 4.5 offers conclusions and discusses some open problems.

4.2 Error Exponent Analysis

In this section we introduce the source coding problem as we consider it, and analyze its error exponent behavior. The source is composed of M parallel sources, each a sequence of i.i.d. symbols distributed according to p_{x_k,y_k} for $k = 1, 2, \ldots M$. We observe n_k samples of source k, giving a total block-length $N = \sum n_k$. The length-N vector of source samples $\mathbf{x} = x_1^N$ is observed at the encoder, while the vector of side informa-

tion samples $\mathbf{y} = y_1^N$ is observed at the decoder. Their joint distribution is given by $p_{\mathbf{x},\mathbf{y}}(\mathbf{x},\mathbf{y}) = \prod_{k=1}^{M}\prod_{l=1}^{n_k} p_{X_k,Y_k}(x_{k,l},y_{k,l})$, where $(x_{k,l},y_{k,l})$ is the lth symbol pair of source k.

In [28] Gallager bounds the error probability of maximum-likelihood decoding of a randomly binned source $\mathbf{x}$ when side information $\mathbf{y}$ is observed at the decoder as

$$\Pr[\text{error}] \leq \exp\left\{-\rho NR + \log \sum_{\mathbf{y}}\left[\sum_{\mathbf{x}} p_{\mathbf{x},\mathbf{y}}(\mathbf{x},\mathbf{y})^{\frac{1}{1+\rho}}\right]^{1+\rho}\right\}. \tag{4.1}$$

The bound holds for all $0 \leq \rho \leq 1$. Substituting in the distribution for parallel sources, and defining $\lambda_k \triangleq n_k/N$ to be the fraction of observed samples that are from source k gives

$$\Pr[\text{error}] \leq \exp\left\{-N \sup_{0\leq\rho\leq 1}\left(\rho R - \sum_k \lambda_k \log\left[\sum_y\left(\sum_x p_{X_k,Y_k}(x,y)^{\frac{1}{1+\rho}}\right)^{1+\rho}\right]\right)\right\}. \tag{4.2}$$

To understand why coding across parallel sources helps, we now relate Equation (4.2) to the error exponent achieved by the independent encoding of each source. In particular, assume that source k is encoded at rate R_k. If the total number of bits available for source coding is NR, the R_k are constrained so that $\sum_k n_k R_k \leq NR$. We assume equality. Since $n_k = \lambda_k N$, the constraint gives $\sum_k \lambda_k R_k = R$ where R is the average Slepian-Wolf coding rate. Substituting $\sum_k \lambda_k R_k$ for R in Equation (4.2) gives the following expression for coding across the parallel sources

$$\Pr[\text{error}] \leq \inf_{0\leq\rho\leq 1}\prod_{k=1}^{M}\exp\left\{-\lambda_k N\left(\rho R_k - \log\left[\sum_y\left(\sum_x p_{X_k,Y_k}(x,y)^{\frac{1}{1+\rho}}\right)^{1+\rho}\right]\right)\right\}. \tag{4.3}$$

This expression lets us understand why coding across parallel sources decreases the error probability. Each of the M factors in Equation (4.3) is a bound on the error probability of the respective parallel source if encoded independently. If encoded independently, we could further optimize each bound separately over ρ. However, if the λ_k were fixed and we let n grow, the overall probability of error would be limited by the worst-case source, giving the following expression for the overall error resulting from separate coding of the parallel sources

$$\Pr[\text{error}] \leq \sup_k \inf_{\substack{0\leq\rho\leq 1,\\ R_k \text{ s.t. } \sum_k \lambda_k R_k = R}} \exp\left\{-\lambda_k N\left(\rho R_k - \log\left[\sum_y\left(\sum_x p_{X_k,Y_k}(x,y)^{\frac{1}{1+\rho}}\right)^{1+\rho}\right]\right)\right\}. \tag{4.4}$$

Contrasting Equation (4.3) with Equation (4.4), we see that while in the latter we can optimize the ρ parameter for each source, in Equation (4.3) we gain from the longer block-length. Indeed, Equation (4.3) strictly dominates Equation (4.4). This can be seen by letting

k^* be the (worst-case) source that maximizes Equation (4.4) with minimizing $\rho = \rho^*$, and rate allocations R_k^* such that $\sum_k \lambda_k R_k^* = R$. Substituting these values into Equation (4.3) results in a lower bound on the probability of error. This is because each of the M factors in Equation (4.3) must be strictly lower than one for all ρ, which is because each R_k must be greater than the conditional entropy $H(\mathsf{x}_k|\mathsf{y}_k)$, else Equation (4.4) would not have a positive error exponent. Thus, as long as the rate is above the conditional entropy, the error exponent is positive for all ρ (and hence for ρ^* even though ρ^* isn't necessarily the best ρ for source $k \neq k^*$).

To aid understanding of the comparison of the bounds in Equations (4.3) and (4.4), consider the plot shown in Figure 4.1. For this plot, we consider the scenario of two sources ($M = 2$) with $N = 10,000$ and $\lambda_1 = \lambda_2 = 0.5$ (This scenario is also simulated in Section 4.4). For the two sources we consider the following distributions

$$p_{\mathsf{x}_1,\mathsf{y}_1}(x,y) = \begin{cases} 0.4530 & \text{if} \quad x = y \\ 0.0470 & \text{if} \quad x \neq y \end{cases},$$

and

$$p_{\mathsf{x}_2,\mathsf{y}_2}(x,y) = \begin{cases} 0.4363 & \text{if} \quad x = y \\ 0.0637 & \text{if} \quad x \neq y \end{cases}.$$

The bound of Equation (4.3) is plotted with a dashed line while the bound of Equation (4.4) is plotted with a solid line. As can be seen in this figure, joint coding across parallel sources offers a far better error bound.

4.3 LDPC Code Design

In this section we shift away from information theoretic analysis to the practical problem of code design for parallel sources. In Section 4.2 we used Gallager's error exponent analysis for the Slepian-Wolf problem. In contrast, we now consider the code design problem using channel code design techniques. The relationship of the Slepian-Wolf code design problem to the channel coding problem has long been recognized [90], and Appendix B makes explicit the relationship between the error exponents for these two problems.

Section 4.2 only makes consideration of two approaches (fully separate coding or joint coding across sources). Practical code design raises other approaches. One such approach would be to use the best available rate R code for coding across sources. In this section, we consider a code design approach that leverages the parallel nature of the source for better

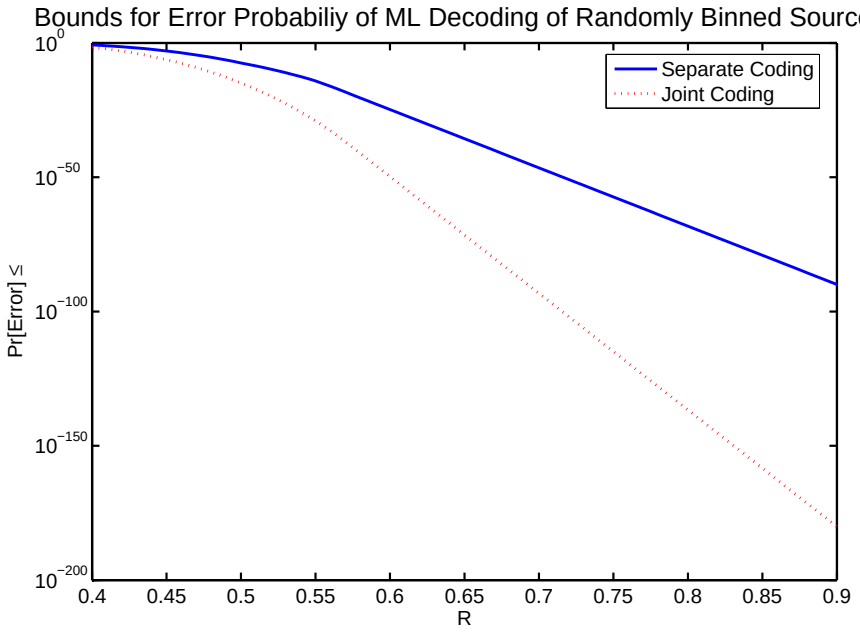


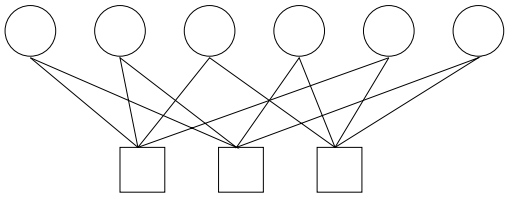
Figure 4.1. A comparison of the bound presented in Equation (4.3) for joint coding versus the bound presented in Equation (4.4) for separate coding. The details of the scenario considered are discussed in the text. For each value of R, the values of R_1 and R_2 are chosen to give the least upper bound on the probability of error. This demonstrates the improved performance achievable with joint coding across parallel sources.

performance. Note, throughout the following we discuss the rate, R, of a code as its source coding rate. Assuming binary random variables, R is related to the channel coding rate, R_c as $R = 1 - R_c$.

Although several types of codes can be used for source coding with side information, we consider LDPC codes here. These codes are particularly suited to the application of coding across parallel sources, due to their performance over large block lengths and the natural adaptability of their decoding algorithm. Details on how to apply LDPC as linear transformation codes (with transform matrices denoted $\mathbf{H}$) to the Slepian-Wolf problem can be found in Chapter 3.

We now give a brief overview of LDPC codes. See Section 2.3 for further discussion. These codes are a class of graph-based capacity-approaching linear block codes. They can be decoded using the iterative sum-product algorithm, an inference algorithm that is exact on trees. Although not exact on "loopy" graphs (such as those that describe LDPC codes), in practice decoding performance is very good. These codes are often defined in terms

of their parity check matrix, and this matrix corresponds to a graphical representation. An example of this is shown in Figure 4.2. Here the circles represent bits, are referred to as variable nodes, and correspond to columns of the matrix. Squares represent parity constraints, are referred to as check nodes, and correspond to rows of the matrix. Parallel sources, and their varying likelihoods, can be naturally accommodated by grouping the bit nodes from each source. In order to use LDPC codes for the practical coding strategies listed above then, a method of designing LDPC codes for parallel sources is necessary.



$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 4.2. A sample LDPC graphical model representing the matrix at right. The circles represent bits and correspond to matrix columns while the squares represent parity constraints and matrix rows. Elements of the matrix indicate edges between nodes. Whenever the element in the ith row and jth column is 1, an edge connects constraint node i with variable node j. A 0 in the corresponding position implies no edge connects the nodes.

4.3.1 EXIT Chart Based Code Design

One method of designing LDPC codes is the EXIT (Extrinsic Information Transfer) chart method of Ashikhmin, Kramer, and ten Brink [7]. We give a brief overview of it here. For greater depth and explanations of the notation, see Section 2.3.2. The EXIT chart design method focuses on developing degree distributions for LDPC codes that decode reliably, asymptotically as block length goes to infinity. We use $\lambda(x)$ for the degree distribution of the bit nodes and $\rho(x)$ for the check nodes.

The EXIT chart method operates by treating the bit nodes and check nodes as two separate decoders that feed their outputs to one another. The behavior of each of these two component decoders is modeled with a curve of the average mutual information between the decoder estimates and the original data, denoted I_A for the input and I_E for the output, is used. The shape of these curves are dependent upon the code's degree distributions, and the source correlation. A sample plot of the curves for both the variable node decoder and the check node decoder are shown in Figures 4.3 and 4.4. To represent the input-output relationship of these two decoders, the curves are plotted on reflected axes.

It was shown [7] that as long as the variable node decoder curve lies above the check node decoder curve when plotted in the manner of Figures 4.3 and 4.4, then successful decoding, asymptotic in block length, is assured. Decoding operates a long a path between the two curves. It was further shown that the area of the gap between the two curves is proportional to the performance gap between the code and the Slepian-Wolf bound. Thus design is reduced to finding degree distributions pairs that minimize the gap between the two curves, conditioned on a positive gap. By fixing a series of check node degree distributions, a good pair of degree distributions can be found with a series of linear programs.

We begin our parallel source code design by considering codes for two sources. Using EXIT chart techniques, we can design LDPC matrices H_{kk} of dimension $n_k R_k$ by n_k for the separate coding. Assuming 2 parallel sources ($M = 2$), we label the degree distributions $(\lambda_1(x), \rho_1(x))$ and $(\lambda_2(x), \rho_2(x))$. We present EXIT charts for LDPC codes designed such, one for rate[2] $R_1 = 0.45$ in Figure 4.3 and one for rate $R_2 = 0.55$ in Figure 4.4. Here, the gaps between the curves are minimal, showing that they are good codes.

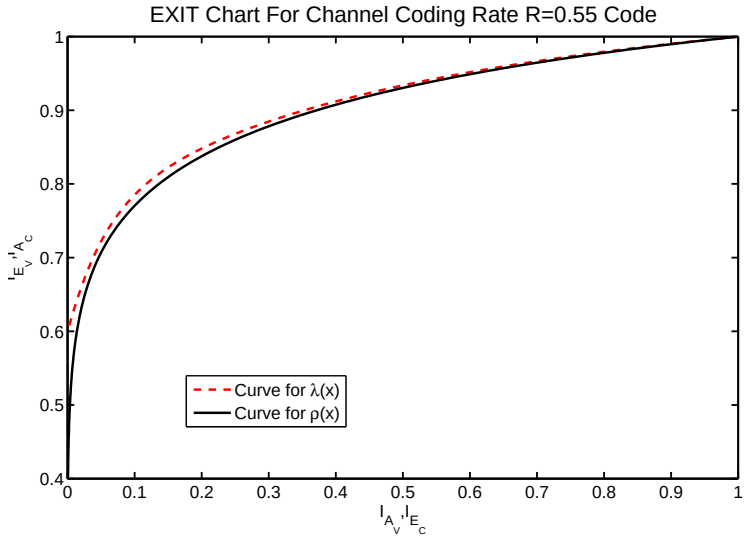


Figure 4.3. One example of a code designed using EXIT charts for a rate R=0.45 source code. The tight fit of the two curves shows that the degree distributions are good. The dashed curve corresponds to the variable nodes while the solid curve corresponds to the check nodes. Decoding operates by moving from the left-hand side to the right-hand side of the graph, going through the gap between the two curves. Since the gap between the two curves is positive, decoding will follow the path between the curves successfully.

[2]Note, as mentioned above, the rates here are the codes' source coding rates. Assuming binary variables, the rate R is related to the channel coding rate R_c as $R = 1 - R_c$.

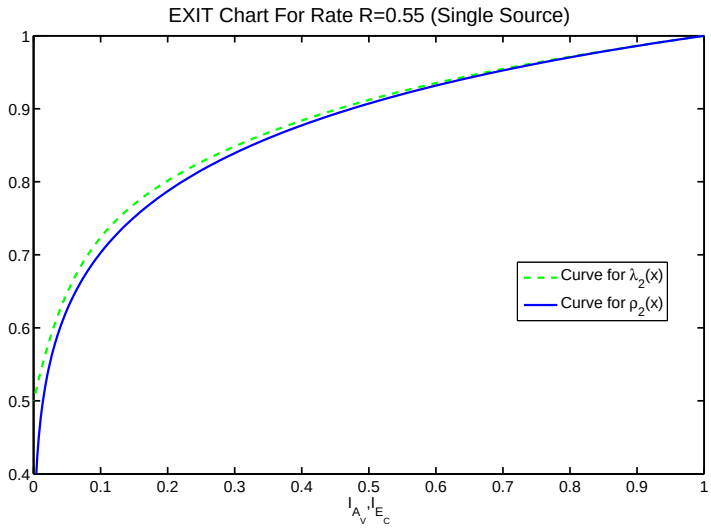


Figure 4.4. A second example of a code using EXIT charts, as in Figure 4.3. This code is a rate R=0.55 source code. Since the gap between the two curves is positive here as well, decoding will follow the path between the curves successfully.

Using one framework to consider the separate coding of the two matrices, we have a larger matrix partitioned in the form in Equation (4.5)

$$\mathbf{H} = \begin{pmatrix} \mathbf{H}_{11} & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_{22} \end{pmatrix}. \tag{4.5}$$

We extend to the joint coding solution by easing this restricted formulation. A more general matrix structure is shown in Equation (4.6)

$$\mathbf{H} = \begin{pmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} \\ \mathbf{H}_{21} & \mathbf{H}_{22} \end{pmatrix}. \tag{4.6}$$

A first design approach for these off diagonal matrices would be to maintain the degree distributions of the separate coding solution but extend them across the entire matrix. We can examine this approach by studying the EXIT charts of Figures 4.3 and 4.4, shown on a single plot in Figure 4.6. It is clear from these charts that the decoders associated with $\mathbf{H}_{11}$, $\mathbf{H}_{22}$, and $\mathbf{H}_{21}$ (the curves for $\lambda_1(x)$ and $\rho_2(x)$) would all result in successful decoders. In contrast, $\mathbf{H}_{12}$ (the curves for $\lambda_2(x)$ and $\rho_1(x)$) would result in a failed decoding, since a positive gap between the two curves is not maintained. An alternate design method is necessary.

Here, we propose a joint EXIT chart design strategy. In our approach we fix several

$\rho_1(x)$ and $\rho_2(x)$ degree distributions and then minimize the gaps between the curves for $\lambda_1(x)$ and $\lambda_2(x)$ while maintaining positive gaps throughout. This force all 4 curves to closely follow each other, and as a result will induce $\rho_1(x) = \rho_2(x)$. Considering an equal share ($\lambda_1 = \lambda_2 = 0.5$) of the two sources considered in Figures 4.3 and 4.4, we present the resulting EXIT chart in Figure 4.6. In the following, we consider performance results from implementations of this solution.

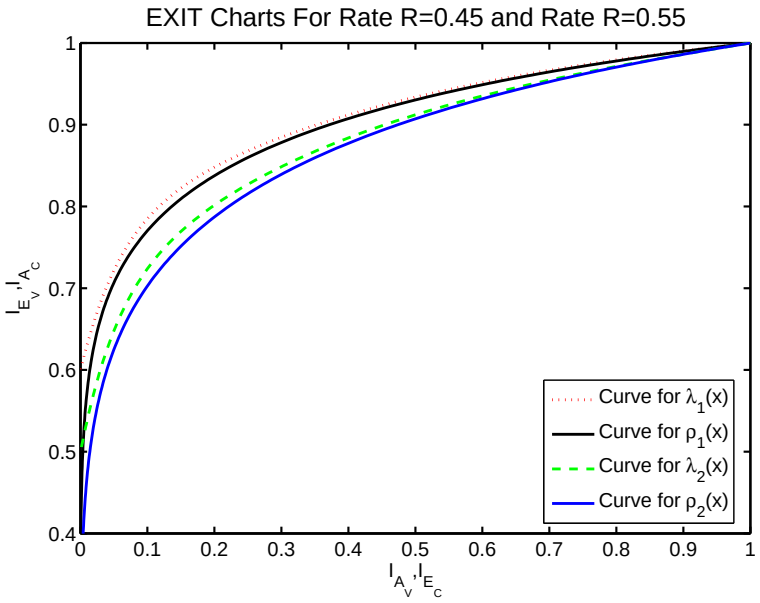


Figure 4.5. A comparison of the EXIT charts for separate coding of two sources. These curves are taken from Figures 4.3 and 4.4. Note the intersection of the curves for $\lambda_2(x)$ and $\rho_1(x)$ prevents the use of these curves for a joint code. Since the gap between these two curves is not maintained, decoding will fail.

4.4 Simulation Results

In this section, we present simulation results of coding two parallel sources. For the simulations, we consider a particular joint distribution[3] such that $y_{k,l} = x_{k,l} \oplus z_{k,l}$ where $x_{k,l}$ and $z_{k,l}$ are independent and $z_{k,l}$ is distributed according to a Bernoulli-Q_k. From the Slepian-Wolf theorem, separately we could code each of the two sources at rates $R_k = H_2(Q_k)$ and code across both sources at rate $R = \lambda_1 H_2(Q_1) + \lambda_2 H_2(Q_2)$. We set $n_1 = n_2 = 5,000$ where $z_{1,l}$ has a Bernoulli-Q_1 distribution and $z_{2,l}$ has a Bernoulli-Q_2 distribution.

[3]For more general source models and code rates, techniques as presented in Chapter 5 should be used.

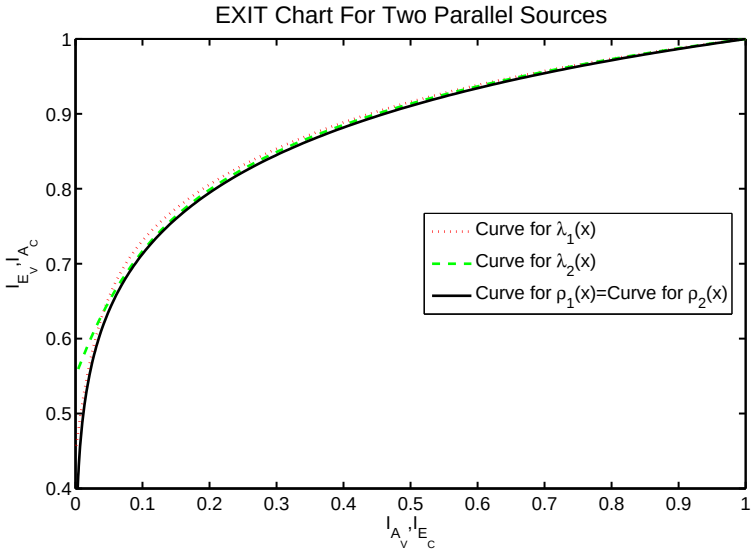


Figure 4.6. A plot of the EXIT chart for the best possible degree distributions found for coding over parallel sources. For these curves, we have considered one check node curve (solid line) and matched both variable node curves (dashed lines) to it. Since the gap between the variable node curves and the check node curve is always positive, decoding will succeed.

In each of the following set of simulations, we operate by selecting a particular code (or pair of codes for separate coding) and then vary both Q_1 and Q_2 over a range of values. In total, we use $5,000$ (rate 0.5) bits to represent the compressed sequences in each case. For separate coding, we use $2,250$ (rate 0.45) bits on the first source and $2,750$ (rate 0.55) on the second source.

We simulated three scenarios of parallel source coding. The results for separate coding are plotted in Figure 4.7, the results for coding across sources using solely a rate $R = 0.5$ code are plotted in Figure 4.8, and results for a coding designed using the method of Section 4.3 are plotted in Figure 4.9. The degree distributions for the first two scenarios can be found

in Appendix A. The degree distribution designed using the technique of Section 4.3 is here

$$\lambda_1(x) = \begin{cases} 0.1120154560x + 0.2582850885x^2 + 0.0001468088x^3 \\ +0.0264927449x^4 + 0.0003299464x^5 + 0.0580453365x^6 \\ +0.0273602982x^7 + 0.1744986414x^8 + 0.0003597095x^9 \\ +0.0001281492x^{10} + 0.0000652846x^{11} + 0.0000417584x^{12} \\ +0.0000298120x^{13} + 0.0000232059x^{14} + 0.0000190287x^{15} \\ +0.0000163757x^{16} + 0.0000145926x^{17} + 0.0000132616x^{18} \\ +0.0000123405x^{19} + 0.0000117698x^{20} + 0.0000113592x^{21} \\ +0.0000111619x^{22} + 0.0000110523x^{23} + 0.0000111133x^{24} \\ +0.0000113830x^{25} + 0.0000116269x^{26} + 0.0000121447x^{27} \\ +0.0000127688x^{28} + 0.0000135517x^{29} + 0.0000146539x^{30} \\ +0.0000160846x^{31} + 0.0000180324x^{32} + 0.0000205581x^{33} \\ +0.0000242711x^{34} + 0.0000298491x^{35} + 0.0000391718x^{36} \\ +0.0000579812x^{37} + 0.0001141510x^{38} + 0.3416494758x^{39} \end{cases}$$

$$\lambda_2(x) = \begin{cases} 0.1192580551 + 0.3487206986x^2 + 0.0000211693x^3 \\ +0.0000368451x^4 + 0.0000448929x^5 + 0.1599996270x^6 \\ +0.0010628532x^7 + 0.1418750045x^8 + 0.0002589765x^9 \\ +0.0001592179x^{10} + 0.0001008248x^{11} + 0.0000724090x^{12} \\ +0.0000737363x^{13} + 0.0000923443x^{14} + 0.0001615279x^{15} \\ +0.0015715951x^{16} + 0.0547919510x^{17} + 0.1702804060x^{18} \\ +0.0008040490x^{19} + 0.0001965115x^{20} + 0.0000959664x^{21} \\ +0.0000648864x^{22} + 0.0000470811x^{23} + 0.0000355826x^{24} \\ +0.0000278140x^{25} + 0.0000223420x^{26} + 0.0000183836x^{27} \\ +0.0000155004x^{28} + 0.0000132890x^{29} + 0.0000115733x^{30} \\ +0.0000102124x^{31} + 0.0000091282x^{32} + 0.0000082153x^{33} \\ +0.0000074589x^{34} + 0.0000068605x^{35} + 0.0000063400x^{36} \\ +0.0000059279x^{37} + 0.0000055754x^{38} + 0.0000051674x^{39} \end{cases}$$

$$\rho_1(x) = \rho_2(x) = 0.2x^8 + 0.8x^9.$$

In these plots, the horizontal axis shows the range of $H_2(Q_1)$ and the vertical axis $H_2(Q_2)$. In these plots the probability of error ranges from low in the dark regions to high in the light regions. As a sample point, consider the point $H_2(Q_1 = 0.084) = 0.416$ and $H_2(Q_2 = 0.087) = 0.427$. Here we measure $P(error) = 10^{-1.5}$ in Figure 4.7, $P(error) = 10^{-3.3}$ in Figure 4.8, and $P(error) = 10^{-4.8}$ in Figure 4.9. As can be seen in these plots, when each source is coded separately, the probability of error is large whenever either $H_2(Q_1)$ *or*

$H_2(Q_2)$ are large. In comparison, when coding across the parallel sources, the probability of error only gets large when $H_2(Q_1)$ *and* $H_2(Q_2)$ are large.

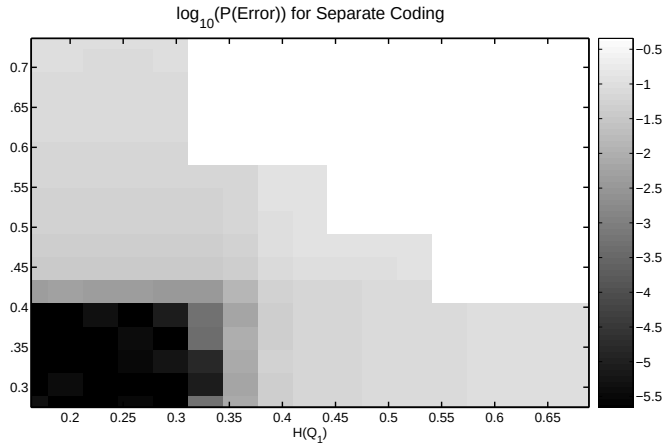


Figure 4.7. $\log_{10}(P(Error))$ results of the simulations for separate coding. The first code is designed with rate $R_1 = 0.45$ (Figure 4.3) while the second code is designed with rate $R_2 = 0.45$ (Figure 4.4). The binary entropy ranges of Q_1 and Q_2 are shown on the horizontal and vertical axes. Dark regions of the plot indicate low probability of error while light regions indicate high probability of error. In these plot, the probability of error is high whenever $H_2(Q_1)$ *or* $H_2(Q_2)$ are large

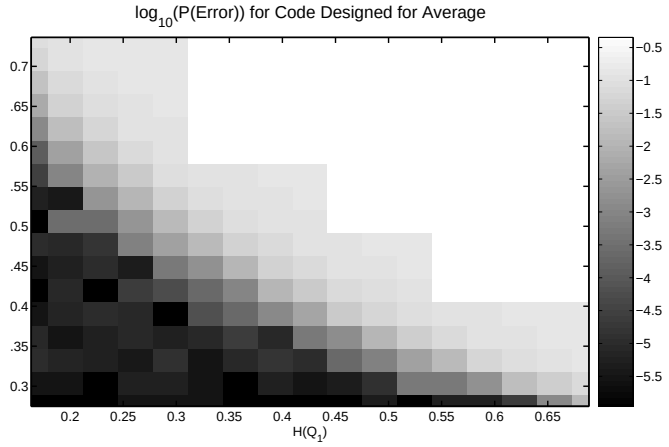


Figure 4.8. $\log_{10}(P(Error))$ results of the simulations using a code designed for rate $R = 0.5$. The binary entropy ranges of Q_1 and Q_2 are shown on the horizontal and vertical axes respectively. Dark regions of the plot indicate low probability of error while light regions indicate high probability of error. Unlike the separate coding results shown in Figure 4.7, the probability of error only gets large when $H_2(Q_1)$ *and* $H_2(Q_2)$ are large.

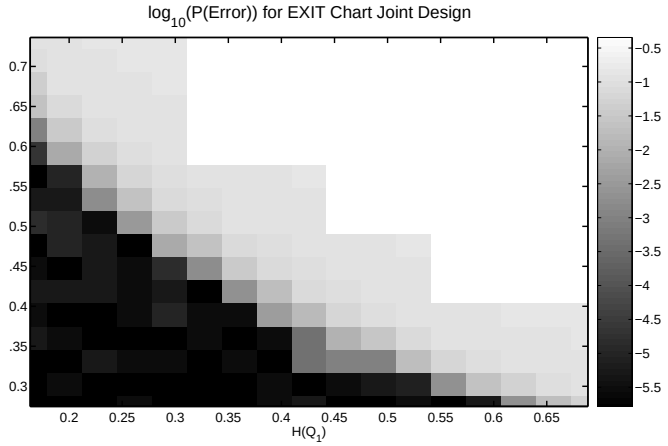


Figure 4.9. $\log_{10}(P(Error))$ results of the simulations for the EXIT chart based design method (Figure 4.6). The binary entropy ranges of Q_1 and Q_2 are shown on the horizontal and vertical axes respectively. Dark regions of the plot indicate low probability of error while light regions indicate high probability of error. Like the single code simulations shown in Figure 4.8, the probability of error only gets large when $H_2(Q_1)$ *and* $H_2(Q_2)$ are large.

In order to aid in the comparison of these results, we use the plots of Figures 4.10 and 4.11. For these plots, we divide each of the plots of Figures 4.7, 4.8, and 4.9 into two regions; decoding success where $P(error) < 10^{-2}$ and decoding failure where $P(error) \geq 10^{-2}$. The comparison between the regions for separate coding and the EXIT chart designed joint code are shown in Figure 4.10 while the comparison with the EXIT chart designed code and the rate $R = 0.5$ code are shown in Figure 4.11. In both of these plots, all decoding succeeds in the lower left corner and fail in the upper right corner. The middle regions show where the EXIT chart designed code results in a decoding success while the other regions result in a decoding failure. The EXIT chart designed code gives us the best performance in each case. Although the gain of the EXIT chart technique over the rate $R = 0.5$ is slim, this is due to the relative closeness of the considered statistics and the limitation to only two sources. These gains would gain in significance when these techniques are applied to more general scenarios.

4.5 Conclusions & Open Problems

In this chapter, we have introduced the problem of Slepian-Wolf coding of parallel sources and have presented a full analysis. We have presented an information theoretic

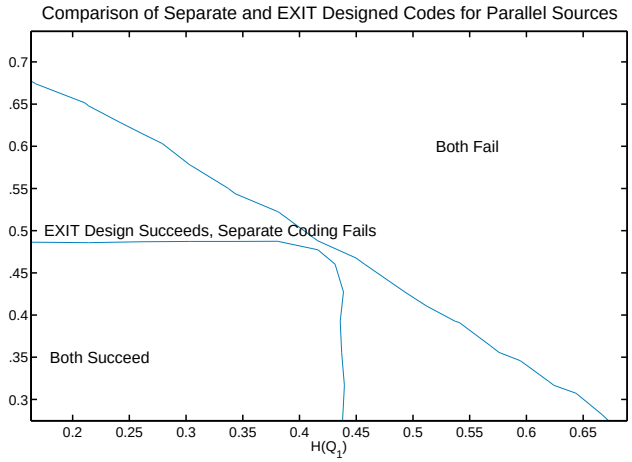


Figure 4.10. Comparison of the results from Figures 4.7 and 4.9. The region of $P(error) > 10^{-2}$ is considered a decoding failure while everywhere else is a success. In this plot, decoding success is seen in the lower left corner and decoding failure in the upper right corner. The middle region represent where the EXIT chart designed code succeeds and the separate coding fails. This figure shows the advantage of joint compression.

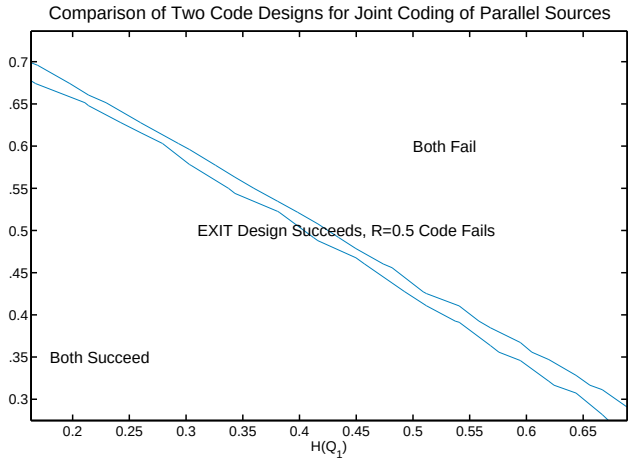


Figure 4.11. Comparison of the results from Figures 4.8 and 4.9. The regions of $P(error) > 10^{-2}$ is considered a decoding failure while everywhere else is a success. As in Figure 4.10, in this plot decoding success is seen in the lower left corner and decoding failure in the upper right corner. The middle region represent where the EXIT chart designed code succeeds and the simple rate $R = 0.5$ code fails. This figure shows the advantage of designing the code for the parallel sources.

view of the problem, showing the potential gains of coding across parallel sources. We have considered the new code design problem raised by coding across parallel sources, and have presented a coding theoretic design technique for the problem. Finally we have presented practical results of LDPC codes using our scheme, built on the construction of Chapter 3, demonstrating the performance gain over other coding strategies.

This work suggests many areas for future work. Consideration of the problem in the case of continuous sources, particular as the problem relates to video is an important extension. A further area of future work is the extension of this work to be blind to the source statistics, requiring no knowledge of the source parameters. In particular, learning the boundaries of each block presents a significant challenge.

Finally, an important necessary step is towards the development of more powerful code design methods. One problem that arises with the EXIT chart based design method presented in this chapter is when there is a large skew in the statistics of the parallel sources. In Section 4.3 the design rates were $R_1 = 0.45$ and $R_2 = 0.55$ giving a small skew (i.e., the absolute difference) $|R_1 - R_2| = 0.1$. If we alter the design rates such that $R_1 = 0.05$ and $R_2 = 0.95$ and $|R_1 - R_2| = 0.9$, the EXIT chart design method performs poorly. This is due to the fact that the choice of the design rates induce a particular I_{E_V} intersect value (as can be seen in Figures 4.3 through 4.6). When there is a large difference in the design rates, minimizing the gap between the curves becomes difficult. Very high degree variable nodes are necessary, for which numerical accuracy concerns result in the failure of the EXIT chart analysis.

A promising solution to this problem is to use density evolution [70]. As discussed in Section 2.3.2, density evolution is a more accurate, though more computationally intensive, design tool. Density evolution is more naturally able to consider the high degree nodes required for a large design rate skew. Further, other coding techniques could be incorporated into the density evolution technique. For example, consider that for a design rate such as $R_2 = 0.95$, since the source and the side information are nearly uncorrelated, it may be beneficial to supply the decoder some values of the source uncompressed (not in the form of a parity sum) in order to aid decoding. This technique, which we refer to as "doping," is discussed in much greater detail in Chapter 7. Due to its flexibility, the notion of doping could be more naturally incorporated into density evolution than into EXIT charts.

In the following chapter, we return to consideration of stationary sources. We proceed to develop codes for the arbitrary distributed source coding problem. Though we omit

discussion of parallel sources in the following chapters, the techniques presented here can be incorporated in a straightforward manner.

Chapter 5

Symmetric Distributed Source Coding

In this chapter, we build on the construction of Chapter 3 and propose a distributed linear block code construction for the Slepian-Wolf problem. The proposed construction is derived from a *single code* for attaining *any point* in the achievable rate region for *arbitrarily correlated* discrete sources. Our prescription allows for any arbitrary memoryless joint probability distribution (even with sources that are marginally compressible) over any arbitrary number of distributed sources and allows for any arbitrary rate combination in the Slepian-Wolf achievable region *without source splitting or alphabet expansion.* Since our framework derives from a single code, codes for similar rate choices are also similar. Our framework reduces effectively for the special cases of a single source and source coding with side-information. In this chapter, we further describe how to incorporate Low-Density Parity-Check (LDPC) codes in our framework to solve the general Slepian-Wolf problem constructively, and present simulation results. We are able to achieve error rates below 10^{-4} with block lengths of $10,000$ while being within 0.30 bits per sample of a 1.15 bits per sample Slepian-Wolf bound (for a total of 1.45 bits per sample, 26.1% above the minimum rate as specified by the Slepian-Wolf bound) for symmetric coding of two arbitrarily correlated sources.

5.1 Introduction

As discussed in previous chapters, the Slepian-Wolf result is asymptotic and non-constructive. Quite a bit of work has been done to extend the work of Slepian and Wolf. One early approach to practical codes is Wyner's [90] formulation of a linear coset code for the corner points[1] (assuming binary sources and symmetric correlation). Wyner's work further establishes the relationship between distributed source coding and channel coding. The late 1990s work of Zamir & Shamai [92] shows the theoretical feasibility of a lattice code framework for a lossy version of this problem (first considered by Wyner & Ziv [89] in the information theoretic setting). Following that, Pradhan & Ramchandran [66] developed DISCUS, a framework for the Wyner-Ziv problem and described practical codes and achievable error probabilities. Recently, a flurry of work in the field provides practical constructions of various levels of sophistication for the Slepian-Wolf problem [66, 64, 80, 83, 30, 95, 34, 96, 31, 32, 46, 49, 50, 43, 79, 76, 75, 56, 54, 2] (greater detail on related work is provided below). However, the constructions to date in the literature do not tackle the general Slepian-Wolf rate region for arbitrary sources. While these are important operating regimes, they fail to cover the complete spectrum of diverse real-world sources and application scenarios requiring more flexible operating points.

Most[2] existing distributed source coding works focus on achieving corner points, arguing that arbitrary points can be achieved by using either source splitting [92, 87, 71] or time sharing. As motivation for fully flexible individual rate choices, consider multiple description coding (studied extensively in the literature [29, 81]). In Pradhan, Puri, & Ramchandran [63, 67], connections between the best-known rate regions to date for the general n-channel ($n > 2$) multiple description problem and the distributed compression problem are identified. Due to multiple description coding's non-ergodic nature, no corner point based solutions are sufficient. Further, time-sharing has asymptotic disadvantages (i.e., smaller error exponents [21]). Better error exponents can be achieved, with increased complexity and requiring detailed knowledge of the source statistics, by using source splitting.

We argue that the fully general Slepian-Wolf problem can benefit from a holistic solution. The main contribution of this chapter is to provide such a solution. Specifically this

[1] As a reminder from Section 2.4, corner points are rate pairs that lie at vertex points of the Slepian-Wolf rate region.

[2] The works [31, 96, 79, 35, 73, 74] all propose practical codes for achieving rates other than the corner points. Their constructions all face a constraint wherein $R_x + R_y \geq 1$, either implicit in their construction or their correlation model considered. We consider scenarios and develop codes not subject to this constraint.

implies (i) arbitrary correlations, (ii) the entire Slepian-Wolf coding region corresponding to arbitrary rate combinations, and (iii) an arbitrary number of sources. The construction presented in this chapter is based on only a single linear block code. This is in contrast to existing solutions requiring unique codes be defined at each operating point, our solution allows neighboring points in the Slepian-Wolf rate region to have similar solutions. Further, our construction simplifies to the constructions of Chapter 3. We focus on LDPC codes in this chapter, since they represent a powerful state-of-the-art class of capacity achieving codes. We describe how to incorporate LDPC codes into our framework by building on the construction of Chapter 3. We provide an efficient algorithm for decoding two sources without loss of generality (assuming knowledge of the source statistics) based on a two-machine framework[3] with extensions to an arbitrary number of sources.

This chapter is organized as follows. Section 5.2 provides a description of the algebraic solution for linear codes to solve the Slepian-Wolf problem. A brief summary of LDPC codes is provided in Section 5.3 and the application of LDPC codes to the construction is described in Sections 5.4 and 5.5. Section 5.6 presents results and Section 5.7 concludes the chapter.

5.1.1 Related Work

The first reported use of syndromes for source coding was in 1969 [8]. This concept is studied extensively in [59, 25, 39, 36, 5, 6]. As mentioned above, the work [90] considers linear codes to achieve the Slepian-Wolf bound for the distributed compression of doubly binary symmetric sources. Witsenhausen & Wyner [88] consider linear codes for source coding with side information at the decoder. Alon & Orlitsky [60, 3] consider several constructive approaches for the Slepian-Wolf problem in a graph-theoretic setting. Csiszár [20] shows that linear codes can achieve the Slepian-Wolf bound in the universal setting using random coding arguments. Uyematsu [80] shows that the decoding complexity is polynomial in block-length. Zamir, Shamai, and Erez [92] show that asymptotically linear codes can achieve the optimal rate-distortion performance for the binary case under a Hamming distortion criterion and lattice codes for the jointly Gaussian case. Pradhan & Ramchandran in [66, 64] develop a constructive framework for the Wyner-Ziv problem.

We can broadly classify the constructive approaches in the recent literature into three categories: (a) LDPC codes-based, (b) Turbo codes-based and (c) Entropy codes-based. An

[3]The work of [96] proposes a similar architecture for joint source-channel coding with a limited correlation model.

article by Xiong, Liveris, and Cheng [91] provides an overview of several techniques from these categories in the context of sensor networks.

LDPC codes-based approaches: Liveris, Xiong and Georghiades [47, 46, 45, 79] presented a variety of schemes for achieving optimal performance in distributed source coding. Particular emphasis was placed on the corner points in the Slepian-Wolf rate region, but allowing for multiple sources and some alphabets larger than binary. The work of Schonberg, Ramchandran, and Pradhan [76] focused on the corner points of the Slepian-Wolf rate region, as well as suggesting an approach for classical single source entropy coding. Caire, Shamai, and Verdu [12, 10, 13, 11] proposed a coding scheme wherein the encoder does additional computation to improve performance and insure correct decoding for the classical single-source compression problem. Sartipi and Fekri [73, 74], considered an alternate method for applying LDPC codes to the Slepian-Wolf problem, but considered only a limited source model. Coleman, Lee, Médard, and Effros [16] presented a scheme for achieving arbitrary rate pairs based on source splitting. Gehrig and Dragotti present Slepian-Wolf codes based on non-systematic linear codes. Zhong, Lou, and García-Frías present a framework based on related LDGM codes.

Turbo codes-based approaches: García-Frías and Zhao [31, 32, 33, 94] proposed a system using punctured turbo codes for the Slepian-Wolf problem. Aaron and Girod considered similar codes for the problem of source coding with side information [2]. Both the teams of Mitran and Bajcsy [56, 55] and the team of Liveris, Xiong, and Georghiades [50] made further contributions along these lines. Finally, considering the arbitrary Slepian-Wolf problem, Li, Tu, and Blum [44] applied entropy codes to the turbo-code outputs in order to achieve additional compression gains.

Entropy-codes based approaches: Koulgi, Tuncel, Ragunathan, and Rose [40, 41] and Zhao and Effros [93] proposed several approaches for the Slepian-Wolf source coding problem using methods which are inspired from the theory of entropy codes for the classical source coding problem.

As stated above, none of these works fully considers a fully arbitrary scenario. In this chapter a linear code based solution is provided focusing on arbitrary correlation models for distributed source coding.

5.2 Distributed Source Coding

In this section, we provide the intuition, framework, and operational details of an algebraic construction for distributed source coding. We focus on binary alphabets. Extension to larger alphabets is largely straightforward, but is not discussed here. We focus first on two sources, x and y and provide a simple example. Then we describe the code generation and decoding algorithm. Finally, we generalize this to a setting involving an arbitrary number of sources. This section builds on the work of Section 3.2.

Recall from Section 3.2, we use a linear block code $\mathcal{C}$ for binning. The bin index, denoted s_x, of a codeword x, is the word's syndrome with respect to $\mathcal{C}_x$'s parity check matrix

$$s_x = x\mathbf{H}_x^T.$$

Thus, we encode x to s_x via matrix multiplication.

5.2.1 Illustrative Example

We now consider an example [64] to motivate our distributed source code construction. This example is similar to the example in Section 3.2.1, but illustrates issues that arise in general distributed source coding that do not arise in source coding with side information.

Let x and y be two uniform 7-bit sources. They are correlated such that they differ in *at most* 1 of their 7 bits (i.e. $wt_H(x \oplus y) \leq 1$). We calculate the following: $H(x) = H(y) = 7$ bits, $H(x|y) = H(y|x) = 3$ bits, and $H(x, y) = 10$ bits.

To code at the rates $(R_x, R_y) = (3, 7)$ bits, as in Chapter 3, we use the $(7, 4)$-Hamming code. Source y sends its 7 bits uncompressed, while source x sends its 3 bit syndrome with respect to the code (recall, $s_x = x\mathbf{H}_x^T$). Since the Hamming code has a minimum distance of 3, knowledge of y^7 and z^7 allows the decoder to determine x^7 exactly.

To code at rates $(R_x, R_y) = (5, 5)$ bits, we will define the codes $\mathcal{C}_x$ and $\mathcal{C}_y$ to compress each source respectively below. In order to show that a unique decoding is possible, consider source pairs (x_1, y_1) and (x_2, y_2) that compress equivalently as below

$$x_1\mathbf{H}_x^T = x_2\mathbf{H}_x^T, \quad \text{and } y_1\mathbf{H}_y^T = y_2\mathbf{H}_y^T.$$

Define $e_1 = x_1 \oplus y_1$ and $e_2 = x_2 + y_2$. From the definition above, we bound the Hamming weight of e_1 and e_2 as below

$$wt_H(e_1) \leq 1, \quad \text{and } wt_H(e_2) \leq 1.$$

Defining $e_3 = e_1 \oplus e_2$, we can write the following

$$x_1 \oplus x_2 \in \mathcal{C}_x, \quad \text{and } x_1 \oplus x_2 \oplus e_3 \in \mathcal{C}_y.$$

We bound $wt_H(e_3) \leq 2$. In order to allow unique decoding, we specify the generator matrices of $\mathcal{C}_x$ and $\mathcal{C}_y$ (compressing with respect to the corresponding parity check matrices for compression). We define $\mathbf{G}_x$ by the first two rows of the Hamming code's generater matrix and $\mathbf{G}_y$ with the other two rows;

$$\mathbf{G}_x = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix},$$

$$\text{and } \mathbf{G}_y = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Since these are sub-codes of the Hamming code, and Hamming codes have a minimum distance of 3, only $e_3 = 0$ satisfies the above constraints. In this case we have the following

$$x_1 = x_2, \quad \text{and } y_1 = y_2.$$

This property insures that unique decoding is possible.

Key to this example is the partition of a generator matrix to develop codes for non-corner points. By choosing the partition, codes for the entire rate region can be obtained from a single code. The intuition of this example is extended more generally in the following.

5.2.2 Code Generation

We develop a general construction for two sources in this section. We focus first on corner points and then generalize to arbitrary rate points. For a fixed block length we construct codes by defining their generator matrices. Rates are lowered, to the Slepian-Wolf bound, by adding rows to each generator matrix. Then, we alter our perspective to show how the codes derive from a single code, as shown in Figure 5.1. Single code derivation offers the advantage that codes for similar operating points are similar. In contrast, in [20], a unique code is randomly generated for each and every rate pair considered.

Assume, without loss of generality, that $H(x) \leq H(y)$. Focusing on the corner point, our goal is to construct generator matrices $\mathbf{G}_x$ and $\mathbf{G}_y$ containing $n(1-H(x|y))$ and $n(1-H(y))$ rows respectively for the point[4] $(R_x, R_y) = (H(x|y), H(y))$.

[4]We ignore the issue of fractional numbers of rows. In practice, additional rate would be required to round up. We further assume that each generator matrix is of full row rank. Thus the design rate and actual rate of the codes are equal.

We begin by considering a generator matrix $\mathbf{G}_c$ of size $n(1-H(y)) \times n$, which we use to bin y. Set $\mathbf{G}_y = \mathbf{G}_c$ to get $R_y = H(y)$. Now construct $\mathbf{G}_a$ of size $n(H(y)-H(x)) \times n$ linearly independent of $\mathbf{G}_c$. Define $\mathbf{G}_x$ as below to get $R_x = H(x)$

$$\mathbf{G}_x = [\overbrace{\mathbf{G}_c^T}^{n\times n(1-H(y))} \quad \overbrace{\mathbf{G}_a^T}^{n\times n(H(y)-H(x))}]^T.$$

These matrices compress each source to its marginal entropy.

To do distributed source coding, we increased the size of $\mathbf{G}_x$ to reduce R_x to $H(x|y)$ bits/sample. We construct $\mathbf{G}_s$ with $n(H(x)-H(x|y))$ rows linearly independent of $\mathbf{G}_c$ and $\mathbf{G}_a$, and construct a larger $\mathbf{G}_x$ by stacking $\mathbf{G}_c$, $\mathbf{G}_a$ and $\mathbf{G}_s$ as in Equation (5.1). Using $\mathbf{G}_x$ and $\mathbf{G}_y$ to compress each source, we achieve $(R_x, R_y) = (H(x|y), H(y))$

$$\mathbf{G}_x = [\overbrace{\mathbf{G}_c^T}^{n\times n(1-H(y))} \quad \overbrace{\mathbf{G}_a^T}^{n\times n(H(y)-H(x))} \quad \overbrace{\mathbf{G}_s^T}^{n\times n(H(x)-H(x|y))}]^T. \tag{5.1}$$

To generate these codes from a single code, consider another viewpoint. Above we view each generator matrix as a stacks of smaller matrices. In contrast, consider the union of the rows of the two matrices (trivially $\mathbf{G}_x$ above). We consider this union as a single code $\mathbf{G}$. From this code, $\mathbf{G}_x$ and $\mathbf{G}_y$ are obtained by selecting subsets of rows from $\mathbf{G}$. This general idea is illustrated in Figure 5.1. Note that the subsets must be chosen so as to satisfy the Slepian-Wolf bounds.

To understand why it is necessary that some commonalities must exist between the two generator matrices, consider codes so derived yet having no rows in common ($\mathbf{G}_c = \mathbf{0}$ and $k_x + k_y = k$). The rates induced by these codes are below

$$(R_x, R_y) = ((n-k_x)/n, (n-k_y)/n).$$

Knowing that a channel code's generator matrix must satisfy $k < n$ and combining the above expressions gives $R_x + R_y > 1$. This restriction is a particular impediment when arbitrary source distributions are considered. If $H(x,y) < 1$, then this restriction bounds the achievable rate region away from the Slepian-Wolf bounds. Thus row sharing is essential for a general solution.

To shift this solution away from a corner point and achieve arbitrary rate pairs, we alter the distribution of rows to each matrix (as in Figure 5.1). We can view this as a partition of $\mathbf{G}_s$ into $\mathbf{G}_{sx}$ and $\mathbf{G}_{sy}$ and assign the partitions to each respective source. When $\mathbf{G}_{sy} = \mathbf{G}_s$, then $(R_x, R_y) = (H(x), H(y|x))$ bits/sample.

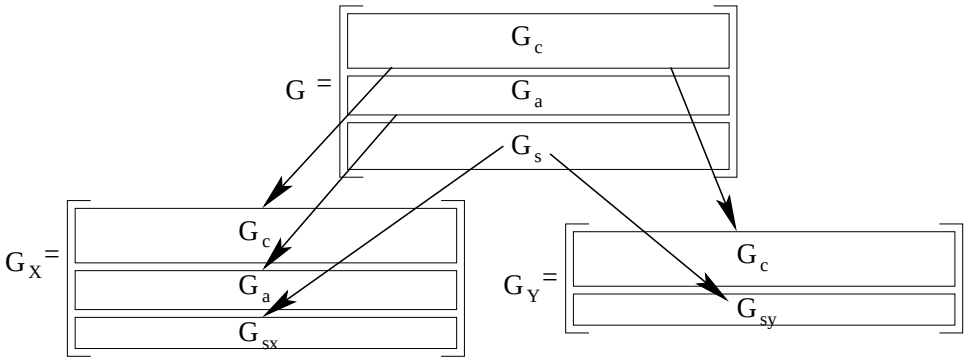


Figure 5.1. The partition and distribution of a single code's generator matrix for the codes for compressing x and y. $\mathbf{G}_c$ is shared between both resulting codes to allow for arbitrary rate point selection.

5.2.3 Decoder

We present here a decoding prescription for two sources operating at an arbitrary rate pair based on two successive decodings. The algorithm presented naturally simplifies for special cases such as source coding with side information and entropy coding, though these cases are not discussed here. Referring to a decoding element as a single-machine decoder, this algorithm is a *successive two-machine algorithm*. This algorithm recovers both sources only after both decodings, since both encoders send "partial" information. No one traditional decoder can recover either source alone. Though the algorithm presented here is impractical (due to the use of two ML decoders), an implementable (but sub-optimal) *simultaneous two-machine algorithm* is given in Section 5.5.

Decoding focuses first on recovering $z = x \oplus y$. It is reasonable to assume that x and y are highly correlated, otherwise Slepian-Wolf coding gains could be nominal at best. We can thus expect that z has low Hamming weight and is compressible. Once z is recovered, x and y are then reconstructed by a second machine. Pseudo code for the successive two-machine algorithm is presented below, where the partitions of $\mathbf{G}$, $\mathbf{G}_x$, and $\mathbf{G}_y$ from Figure 5.1 are used.

In the first step of the pseudo code, as in Section 3.2.2, we write each source as a function of the compression code and the compressed value. We use the variable u to select which codeword is necessary to satisfy these equations. We denote the closest codeword x (y) in $\mathcal{C}_x$ ($\mathcal{C}_y$) as $u_x\mathbf{G}_x$ ($u_y\mathbf{G}_y$). We further divide these equations into their component parts as

follows

$$u_x\mathbf{G}_x = [u_{c,x}|u_{a,x}|u_{s,x}][\mathbf{G}_c^T\mathbf{G}_a^T\mathbf{G}_{sx}^T]^T = u_{c,x}\mathbf{G}_c \oplus u_{a,x}\mathbf{G}_a \oplus u_{s,x}\mathbf{G}_{sx}$$

$$\text{and} \quad u_y\mathbf{G}_y = [u_{c,y}|u_{s,y}][\mathbf{G}_c^T\mathbf{G}_{sy}^T]^T = u_{c,y}\mathbf{G}_c \oplus u_{s,y}\mathbf{G}_{sy}.$$

In Step 1, $\bar{s}_x$ ($\bar{s}_y$) represents a zero-padded version of the value s_x (s_y). $\bar{s}_x$ ($\bar{s}_y$) is the element in the co-sets formed by $\mathcal{C}_x$ ($\mathcal{C}_y$) of minimum Hamming weight, as in Section 3.2.2. We repeat the description of zero padding from Section 3.2.2 here. To zero pad, insert zeros so that they align with the systematic bits of the code[5]. For example, consider a code defined with the first two rows of the Hamming code Equation (5.2). Since the fourth and fifth columns (emphasized) form the systematic bits, zeros are inserted into the fourth and fifth positions.

$$\text{If} \quad \mathbf{G}_x = \begin{bmatrix} 0 & 1 & 1 & \textit{1} & \textit{0} & 0 & 0 \\ 1 & 0 & 1 & \textit{0} & \textit{1} & 0 & 0 \end{bmatrix} \quad \text{and} \quad s_x = [\overbrace{1\,0\,1}\,\overbrace{0\,1}], \tag{5.2}$$
$$\text{then} \quad \bar{s}_x = [\overbrace{1\,0\,1}\;0\,0\;\overbrace{0\,1}].$$

Step 2 of the decoding combines $\bar{s}_x$ and $\bar{s}_y$. The first machine runs in Step 3, the ML decoding of $\bar{s}$ with respect to $\mathbf{G}$ to give $\hat{z}$ and u. Knowledge of u gives knowledge of $u_{s,y}$. Insert $u_{s,y}$ into the equation for y. With greater knowledge of y, in Step 4 we ML decode $u_{s,y}\mathbf{G}_{sy} \oplus \bar{s}_y$ with respect to $\mathbf{G}_c$ to obtain $\hat{y}$ and $u_{c,y}$. Finally, combine $\hat{y}$ and $\hat{z}$ to recover $\hat{x}$ in Step 5.

Pseudo code for the decoding algorithm is provided here:

1. Expand sources equations: $y^n = u_{c,y}\mathbf{G}_c \oplus u_{s,y}\mathbf{G}_s y \oplus \bar{s}_y$ and $x^n = u_{c,x}\mathbf{G}_c \oplus u_{a,x}\mathbf{G}_a \oplus u_{s,x}\mathbf{G}_s x \oplus \bar{s}_x$.

2. Composite values: $\bar{s} = \bar{s}_x \oplus \bar{s}_y$ and $u = [u_{c,x} \oplus u_{c,y}|u_{a,x}|u_{s,x}|u_{s,y}]$.

3. ML Decode[6] $\hat{z} = x \oplus y = u\mathbf{G} \oplus \bar{s}$.

4. ML Decode $\hat{y} = u_{c,y}\mathbf{G}_c \oplus [u_{s,y}\mathbf{G}_{sy} \oplus \bar{s}_y]$ using knowledge of $u_{s,y}$.

5. Recover $\hat{x} = \hat{y} \oplus \hat{z}$ using knowledge of $\hat{z}$ and $\hat{y}$.

[5]More generally, we need select any k_x (k_y) linearly independent columns of $\mathbf{G}_x$ ($\mathbf{G}_y$) since bit ordering does not impact code performance in the memoryless case.

[6]In this construction, care needs to be take to insure that the dimensions of $\mathbf{G}$ are sufficient for the statistics of $\boldsymbol{z}$. Note though, in cases of interest where significant correlation between the sources is present, this will indeed be the case.

This prescription gives insights on the design requirements of $\mathcal{C}$ (i.e., $\mathbf{G}$ and $\mathbf{G}_c$). Each of these codes needs be a "good" code ("good" if the probability of a decoding error goes to zeros as block length goes to infinity). Using random codes, code design first randomly generates $\mathbf{G}_c$ to act as an entropy coder for either source given their sum. Then random code $\mathbf{G}_{sy}$ is generated to bin the bins induced by $\mathbf{G}_c$. The resulting sub-bins need to be strong enough for the statistics of the sum of the two sources. Though difficult in practice, these constraints can be approached with large block lengths and pseudo-random codes [20]. Though we do not prove that the block codes construction here achieves the Slepian-Wolf limit, we demonstrate it via empirical methods (Section 5.6).

5.2.4 Multiple Sources

We now extend the construction to L sources. As in Section 5.2.2, we begin by stacking generator matrices and then shift to a single code viewpoint. We show that codes $\mathcal{C}_i$ ($i \in \{1, 2, \cdots, L\}$) can be derived from a single code $\mathcal{C}$.

Since there is no inherent ordering of the sources, without loss of generality, we label them $\{x_1, \ldots, x_L\}$ so that the following is true

$$H(x_1|x_2, \ldots, x_L) \leq \ldots \leq H(x_i|x_{i+1}, \ldots, x_L) \leq \ldots \leq H(x_L).$$

We generate L codes to achieve the corner point below by recursive matrix definitions

$$\begin{aligned}&(R_1, R_2, \ldots, R_L)\\ &= (H(x_1|x_2, \ldots, x_L), \ldots, H(x_i|x_{i+1}, \ldots, x_L), \ldots, H(x_L)).\end{aligned}$$

The recursion initializes defining matrix $\mathbf{G}_L$ with $n(1 - H(x_L))$ linearly independent rows. The recursion defines the matrices from $\mathbf{G}_{L-1}$ to $\mathbf{G}_1$ in reverse order, setting $\mathbf{G}_{i-1} = [\mathbf{G}_i^T \ \mathbf{A}_{i-1}^T]^T$. $\mathbf{A}_{i-1}$ is full rank with $n(H(x_i|x_{i+1}, \ldots, x_L) - H(x_{i-1}|x_i, \ldots, x_L))$ rows linearly independent of $\mathbf{G}_i$. Thus $\mathbf{G}_{i-1}$ is full rank. This recursion is illustrated in Figure 5.2.

We define a single code $\mathbf{G}$ as the union of the rows of the generator matrices, $\{\mathbf{G}_1, \ldots, \mathbf{G}_L\}$ ($\mathbf{G} = \mathbf{G}_1$ above). Arbitrary points are achieved by modifying the assignment of rows from $\mathbf{G}$. The resulting codes achieve the Slepian-Wolf bound.

ML decoding (though impractical) for an arbitrary number of sources requires a *successive L-machine algorithm*. As above, the sources are only obtained after the final machine is run. A practical (but suboptimal) *simultaneous L-machine algorithm* is described in Section 5.5.

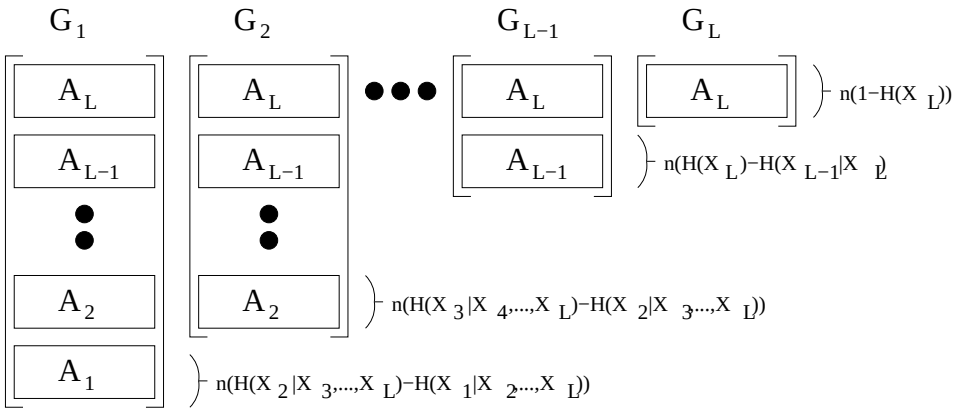


Figure 5.2. Iterative generator matrix definition process for achieving a corner point for the distributed compression of L sources. The process initializes with the L-th matrix on the right-hand side and iterates until the left-most matrix is defined. After defining these codes, arbitrary points can be achieved by moving rows between generator matrices.

5.3 Low-Density Parity-Check Codes

In this section, we highlight some of the background for LDPC (Low-Density Parity-Check) codes. The reader is referred to Section 2.3 for further details. Low-Density Parity-Check codes are a class of powerful linear block codes and thus a natural choice for our distributed source coding framework. We use factor graphs [42] (as in Figure 2.3) to visually represent LDPC codes. These bipartite graphs consist of circles used to represent bits and squares used to represent constraints.

Decoding of LDPC codes is achieved using the sum-product algorithm [26, 61, 42]. The algorithm is an iterative inference algorithm that though exact on trees, is known to have good (but sub-optimal) empirical performance on loopy graphs such as those of an LDPC code. The sum-product algorithm operates by iteratively estimating the marginal distribution for each bit. In the first phase of each iteration, each variable node combines the estimates available and sends them along its edges (e.g., $\nu_{i,j}$ in Figure 5.3). In the second phase of each iteration, these estimates are merged, according to the constraint, at each check node into for each bit node (e.g., $\mu_{j+1,i+2}$ in Figure 5.3). The marginals are used to estimate the bits after each iteration, if below a maximum number of iterations, the set of estimates is checked against a stopping condition.

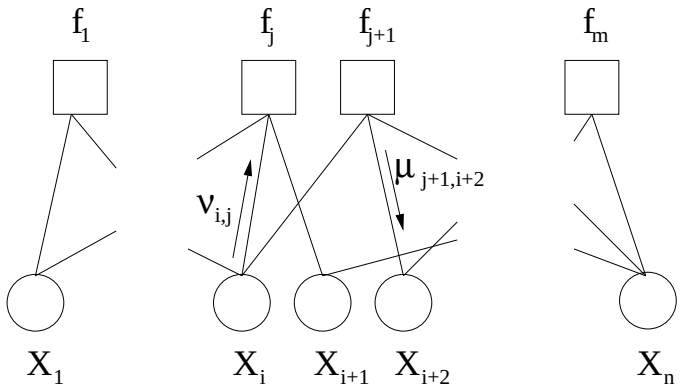


Figure 5.3. A sample factor graph. The circles represent bits while squares represent the parity constraints of the code. A labeling of the two types of messages passed node to node in the decoding algorithm is given.

5.4 Adaptation of Coding Solution to Parity Check Matrices

This section considers the generator matrix partitioning scheme of Section 5.2 in terms of parity check matrices. Such a consideration is valuable for codes such as LDPC codes, defined in terms of their parity check matrices. We develop the "column dropping" procedure, analogous to the matrix partitioning technique above. Specific application of LDPC codes for distributed source coding is given in Section 5.5. In the "column dropping" technique, columns of the base parity check matrix (the "single" code) are zeroed (dropped from the parity equations) and an extra row is added on to the parity check matrix. We develop the technique using a generic systematic code, and then extend it to general codes.

Take $\mathbf{G} = [\mathbf{I}_k|\mathbf{P}]$ and $\mathbf{H} = [\mathbf{P}^T|\mathbf{I}_m]$ to be a systematic generator and parity check matrix pair. We write out the partition of Figure 5.1 in this form in Equations (5.3) and (5.4)

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_{k_c} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{P}_c \\ \mathbf{0} & \mathbf{I}_{k_a} & \mathbf{0} & \mathbf{0} & \mathbf{P}_a \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{k_{sx}} & \mathbf{0} & \mathbf{P}_{sx} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_{k_{sy}} & \mathbf{P}_{sy} \end{bmatrix}, \tag{5.3}$$

$$\text{and } \mathbf{H} = \begin{bmatrix} \mathbf{P}_c^T & \mathbf{P}_a^T & \mathbf{P}_{sx}^T & \mathbf{P}_{sy}^T & \mathbf{I}_m \end{bmatrix}. \tag{5.4}$$

Algebraically we determine expressions for $\mathbf{H}_x$ and $\mathbf{H}_y$

$$\mathbf{H}_x = \begin{bmatrix} \mathbf{P}_c^T & \mathbf{P}_a^T & \mathbf{P}_{sx}^T & \mathbf{0} & \mathbf{I}_m \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_{k_{sy}} & \mathbf{0} \end{bmatrix},$$

$$\text{and } \mathbf{H}_y = \begin{bmatrix} \mathbf{P}_c^T & \mathbf{0} & \mathbf{0} & \mathbf{P}_{sy}^T & \mathbf{I}_m \\ \mathbf{0} & \mathbf{I}_{k_a} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{k_{sx}} & \mathbf{0} & \mathbf{0} \end{bmatrix}.$$

Examine $\mathbf{H}_x$ to see the "column dropping" procedure. Compared to $\mathbf{G}$, $\mathbf{G}_x$ has the rows containing $\mathbf{P}_{sy}$ removed. Comparing $\mathbf{H}$ to $\mathbf{H}_x$, we see that $\mathbf{P}_{sy}$ has been replaced by $\mathbf{0}$ while k_{sy} additional rows have been added to $\mathbf{H}_x$. In the new rows, all elements are zero except $\mathbf{I}_{k_{sy}}$ which is below the $\mathbf{0}$ in place of $\mathbf{P}_{sy}$. We see that in the transition from $\mathbf{H}$ to $\mathbf{H}_x$, columns of $\mathbf{H}$ have been "dropped" into the added rows.

Since permuting the order of the bits of a code does not affect its performance over memoryless channels, we can "drop" arbitrary sets of columns. This allows the extension of this technique to an arbitrary number of sources since the columns need not be adjacent.

To consider non-systematic codes, consider an alternate interpretation of the "column dropping" procedure. Examining the "column dropped" parity check matrices, we note that a portion of the syndrome bits are exactly equal to some of the source bits. The decoder thus knows the exact value of the bits of the "dropped columns." The codes used for each source are thus the main code with a portion of the original bits sent uncoded. We will explore this idea in greater depth in Section 7.3.4. Consider the non-systematic parity check matrix $\mathbf{H} = [\mathbf{P}_c\ \mathbf{P}_a\ \mathbf{P}_{sx}\ \mathbf{P}_{sy}\ \mathbf{P}_0]$. We use the alternate perspective of the "column dropping" procedure to write the corresponding $\mathbf{H}_x$ and $\mathbf{H}_y$ as follows

$$\mathbf{H}_x = \begin{bmatrix} \mathbf{P}_c^T & \mathbf{P}_a^T & \mathbf{P}_{sx}^T & \mathbf{P}_{sy}^T & \mathbf{P}_0 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_{k_{sy}} & \mathbf{0} \end{bmatrix},$$

$$\text{and } \mathbf{H}_y = \begin{bmatrix} \mathbf{P}_c^T & \mathbf{P}_a^T & \mathbf{P}_{sx}^T & \mathbf{P}_{sy}^T & \mathbf{P}_0 \\ \mathbf{0} & \mathbf{I}_{k_a} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{k_{sx}} & \mathbf{0} & \mathbf{0} \end{bmatrix}.$$

5.5 Integration of LDPC Codes Into Distributed Coding Construction

We now present a *simultaneous multi-machine algorithm* for use with LDPC codes. We first describe the graphical model for coding two correlated sources, then generalize for an arbitrary number of sources. Simultaneous decoding is achieved via the sum-product algorithm over these graphs. Since these are graphs loopy, sum-product is sub-optimal but

empirically good. We assume *a priori* knowledge of the source statistics. Chapter 6 presents a method by which we can relax this assumption.

The general graphical model for distributed source coding of two sources is in Figure 5.4. The variables considered are the n bits of sources x and y (middle top and bottom rows of circles respectively), the m_x bits of the compressed source x (top row), and the m_y bits of the compressed source y (bottom row). These variables are constrained by the LDPC codes used to compress x (top row of squares) and y (bottom row) plus the correlation between sources x and y (middle row). The correlation constraints between the sources x and y are distinct and different from the parity constraints considered in Section 5.3.

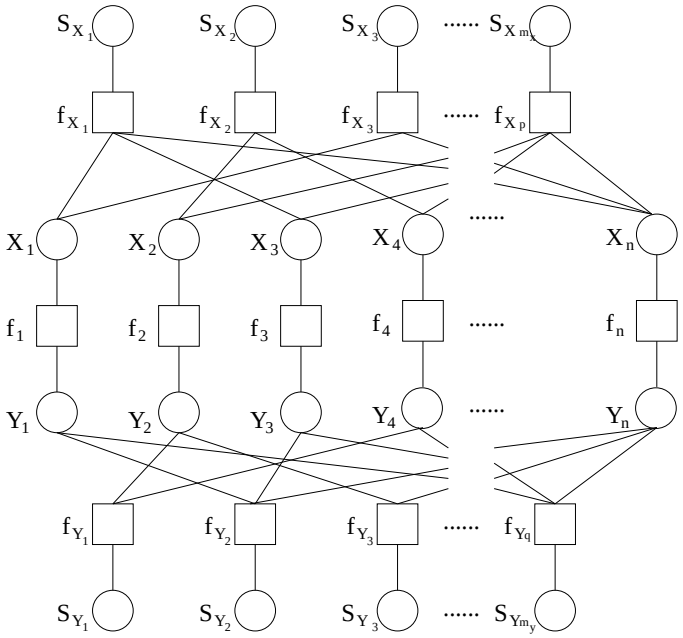


Figure 5.4. Factor graph for distributed source coding. Circles labeled x_i or y_i represent source bits. Circles labeled S_{x_j} or S_{y_j} represent the compressed bits. Squares labeled f_{x_j} or f_{y_j} represent the LDPC code applied to each source. Squares labeled f_i represent the correlation between bits x_i and y_i.

In order to adapt this graphical model to other problems, we consider the model as several connected subgraphs. One subgraph, the "X operations", is composed of the x bit nodes, the code applied to x, and the compressed bits of x. The respective graph elements for y form the "Y operations" subgraph. Finally, the correlation constraints form a "correlation operations" graph. The abstracted graph is thus composed of the source subgraphs connected via the correlations subgraph. We refer to the source subgraphs as

"machines." This abstracted factor graph is shown in Figure 5.5. Recall that we use rectangles as well as squares to represent constraints in factor graphs.

To extend this graphical model to code L sources, we attach a machine for each source to the graph. Each of the L machine is connected to the others via the correlations subgraph. Using the sum-product algorithm over this graph gives the simultaneous multiple machine algorithm.

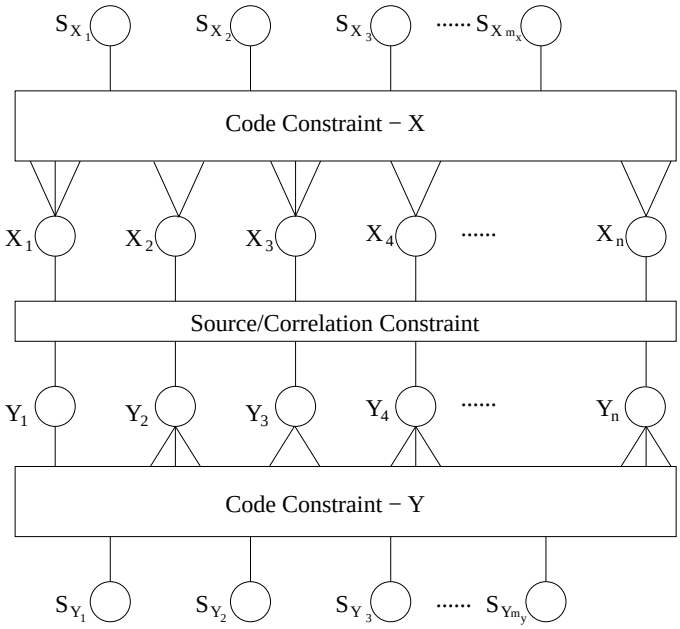


Figure 5.5. An abstracted version of the factor graph of Figure 5.4. Recall that we use rectangles as well as squares to represent constraints. Here the graph is abstracted into the operations on each source and the correlation interface between the two. This graph can be extended to an arbitrary number of sources by interfacing additional sources with the generalized correlation constraint.

We further consider two special cases; entropy coding and source coding with side information. For entropy coding, remove the "Y operations" subgraph from Figure 5.4. This effectively reduces it to the entropy coding graph in Figure 3.4. The correlation constraints become source constraints due to the lack of side information. A source coding with side information model is made by replacing the "Y operations" machine with just the y bit nodes, as in Figure 3.3 in Chapter 3. Since the source y is uncompressed, the interaction with the correlation nodes is straight forward. Note, the graphs of Figures 5.4 and 5.5 naturally simplify as appropriate for both of these special cases.

As in Section 5.2.3, the codes used in Figure 5.4 derive from a single code. Interpreting

Section 5.4, we use the same code twice, and account for the additional uncoded bits in the decoding algorithm. We achieve this by inserting certainty in the distribution estimates for the "dropped column" bits.

As above, the single code must meet two design constraints. First, the overall code must be strong enough for the statistics of the sum of the sources. Second, the reduced code (the code resulting from consideration of the remaining columns) must be strong enough to recover either source given their difference. In general, a nested LDPC design algorithm could be used to first build the inner matrix and then the entire matrix.

5.5.1 Application of Sum-Product Algorithm

As the factor graph of Figure 5.5 is related to the factor graphs of Chapter 3, so are the update equations used. We refer the reader to Section 3.4.1 to see the form of the update equations. The only extension to the pseudo code given in that section is in the actions of the "Y operations" nodes. Fortunately, these update equations simply mirror the form of the "X operations" nodes. As an example, consider the message from correlation node f_i to y_i. The message is of the following form

$$\mu_{f_i,y_i} = \log \frac{(2-P(y_i=1|x_i=0)-P(y_i=1|x_i=1))+(P(y_i=1|x_i=1)-P(y_i=1|x_i=0))\tanh \nu_{x_i,f_i}/2}{(P(y_i=1|x_i=0)+P(y_i=1|x_i=1))+(P(y_i=1|x_i=0)-P(y_i=1|x_i=1))\tanh \nu_{x_i,f_i}/2}. \tag{5.5}$$

Alteration of the other update equations is similarly straightforward. In the following section, we present results obtained from simulating this algorithm.

5.6 Simulations and Results

In this section, we present simulation results. We focus on simulations of the general distributed source coding problem. For results considering the special cases mentioned in this chapter, including entropy coding and source coding with side information, see Section 3.5.

We measure the quality of a simulation with its empirical probability of error, the proportion of bits decoded in error. Since out scheme is asymptotically lossless, we make comparisons with the lossless bounds of Slepian-Wolf despite our measured probabilities of error. We focus here on BER (bit error rate). For a discussion of SER (symbol error rate) as a measure of simulation quality, please see Section 3.5. Each simulation consisted of selecting a particular code, source distribution, and rate combination. Each empirical

$P(x,y)$	$y=0$	$y=1$
$x=0$	$(1-\gamma)(1-\alpha)$	$(1-\gamma)\alpha$
$x=1$	$\gamma\beta$	$\gamma(1-\beta)$

Table 5.1. Structure of the joint distribution between x and y considered for Section 5.6.

probability is generated by simulation of *at least* one million bits, broken into blocks. We limited decoding to at most 50 iterations and used LDPC code designs as described in Appendix A. We omit error bars in these plots for clarity. The resulting error bars are similar to those obtained for the BER results from Section 3.5.

As in Section 3.5, we consider arbitrary distributions over two correlated binary sources, parameterized via $0 \leq \alpha, \beta, \gamma \leq 1$. The distribution considered is $P(x=0, y=0) = (1-\gamma)(1-\alpha)$, $P(x=0, y=1) = (1-\gamma)\alpha$, $P(x=1, y=0) = \gamma\beta$, and $P(x=1, y=1) = \gamma(1-\beta)$. We give a diagram of this in Table 5.6.

We now present results for compressing 10^4 bit blocks from two sources at arbitrary rate combinations (Figure 5.4). To consider a range of arbitrary source distributions we consider all 27 possible combinations that arise from fixing $\alpha \in \{0.01, 0.05, 0.09\}, \beta \in \{0.005, 0.09, 0.15\}, \gamma \in \{0.1, 0.25, 0.5\}$. We use a base code of rate 0.5, from Appendix A. For practical considerations, we omit the dual objective design as discussed in Section 5.4 in favor of simpler code design. We use the "column dropping" method of Section 5.4 for generating the two codes from the base code. Empirical performance is good. We omit consideration of the successive two-machine algorithm since we lack a practical implementation of ML decoding for this block length.

For example, consider $(\alpha, \beta, \gamma) = (0.09, 0.005, 0.25)$. The resulting distribution is $P(x = 0, y = 0) = 0.6825$, $P(x = 0, y = 1) = 0.0675$, $P(x = 1, y = 0) = 0.0013$, and $P(x = 1, y = 1) = 0.2487$. This distribution has entropy properties $H(x) = 0.8113$ bits, $H(x|y) = 0.2497$ bits, $H(y) = 0.9003$ bits, $H(y|x) = 0.3387$ bits, and $H(x, y) = 1.150$ bits. We consider rate combinations with a gap (from the Slepian-Wolf bound) ranging from 0.0 bits to 0.4 bits. Resulting bit error rates are listed in Table 5.6. The rates used are listed atop each column (R_x) and leftmost in each row (R_y).

More extensive results are plotted in Figure 5.6. The results of each (α, β, γ)-tuple and each rate pair chosen in the previous example are considered. We plot the difference between the sum rate and the joint entropy ($R_x + R_y - H(x, y)$) on the horizontal axis and

P(bit error) ($\times 10^{-4}$)			
$R_x \diagdown R_y$	0.60	0.70	0.80
0.55	4732.9	832.1	70.5
0.65	678.3	67.3	10.2
0.75	73.6	9.5	0.2

Table 5.2. Simulation results for several arbitrary rate combinations in the Slepian-Wolf region for a block length of 10,000. The BER$\times 10^4$ results are given for several rate combinations. As the sum rate grows, the resulting error rates improve.

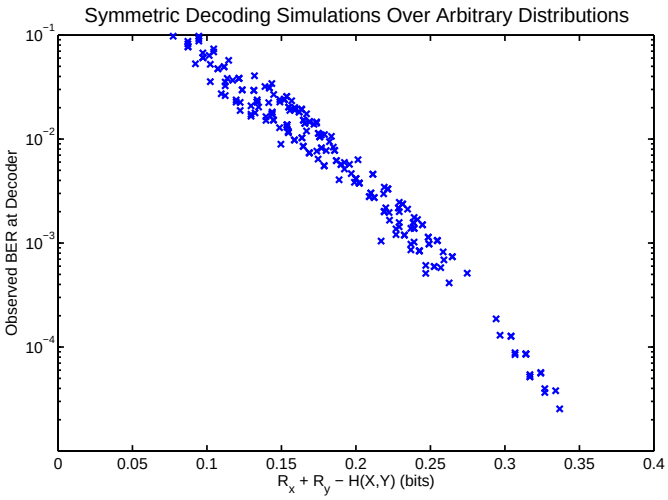


Figure 5.6. Simulation results for several arbitrary rate combinations in the Slepian-Wolf region for a block length of 10,000. Each data point is the result of the simulation of at least 10^6 bits. The BER results are plotted versus the difference between the sum rate and the Slepian-Wolf bound. As the difference grows, resulting error rates improve.

the observed bit error rate on the vertical axis. As expected, performance improves as the distance between the sum rate and the Slepian-Wolf bound grows.

5.7 Conclusions & Open Problems

In this chapter we presented a construction using linear codes for achieving the Slepian-Wolf bound. This construction allows for arbitrary distributions, arbitrary rate pairs, an arbitrary number of sources, and derives from a single code. We presented this construction using either generator or parity check matrices, enabling application of a range of codes.

Key to our scheme is the particular partition of a channel code to produce Slepian-Wolf codes.

Further, we illustrated how to incorporate LDPC codes into the construction and how a two-machine construction can be used to decode to arbitrary rate pairs. We described how to generalize the two-machine construction to accommodate an arbitrary number of sources for arbitrary rate combinations. Finally, we presented simulation results validating the performance of the two-machine decoder.

This work suggests many areas for future work. As in Chapter 4, a particularly promising area of study is in code design. Application of density evolution [70] to the "column dropping" procedure could better describe how to partition the single code to each source. Another area of study is in the application of the algorithm presented here for larger alphabets. Although extension to larger alphabets is straightforward, computational complexity unfortunately also rises. Application of approximations to the algorithm presented here could potentially provide faster performance.

Throughout this chapter and the previous chapters, we have assumed that the source statistics are known *a priori*. In practical implementations these statistics may not actually be available. As a result, tn the following chapter, we describe a protocol for blind distributed source coding.

Chapter 6

A Protocol For Blind Distributed Source Coding

6.1 Introduction

To this point, in Chapters 3 through 5, we have assumed that both the encoder and decoder know the source statistics. With this knowledge, the system can determine ahead of time at what rate to operate. In this chapter, we relax this assumption and present a scheme that adapts its rate to that required by the source statistics. Since distributed source encoders lack a manner by which to learn the joint source statistics, feedback is necessary to gain the full compression gains. Our protocol relies on the availability of a small amount of decoder-to-encoder feedback.

Consider the following example of two distributed sources. The first source, x, is i.i.d. with a Bernoulli-(1/2) distribution. We define the second source, y, such that for every time instance they are exactly equal, i.e. $x_i = y_i$. If the encoders know this correlation prior to encoding, then only one encoder needs to be active at a time. In this case, we could easily the achieve the Slepian-Wolf bound of $R_x + R_y = 1$. Without knowledge of the correlation between x and y, each encoder can only apply traditional source coding techniques. Since the marginal entropy of each source if $H(x) = H(y) = 1$, in this case the sum rate achieved is $R_x + R_y = 2$. Without feedback, there is no way for the encoders to learn of the correlation between the x and y. This example demonstrates the value of feedback, since the encoders could quickly be informed of a more efficient transmission scheme.

Our approach is motivated by the ideas developed in Draper [23]. As we demonstrate, the protocol quickly converges to the rate the encoder would use if it had access to the source statistics. Although our analysis is for source coding with side information for binary sources with a binary symmetric correlation, we explain how to extend this algorithm to other scenarios. We demonstrate such an extension here and in Chapter 8.

This chapter is organized as follows. In Section 6.2 we specify our blind distributed compression protocol and describe the source scenario we analyze it on. We then analyze our blind transmission protocol using error exponents in Section 6.3. Then, in Section 6.4 we discuss how to implement our protocol in practice, and give simulation results in Section 6.5. We finally conclude this chapter in Section 6.6.

6.2 Protocol

For this analysis, the source model is as follows. The sources x_i and y_i are sequences of i.i.d. Bernoulli-(1/2) binary random variables, while z_i is a sequence of i.i.d. Bernoulli-(Q) binary random variable independent of x_i. We relate these variables as follows

$$y_i = x_i \oplus z_i. \tag{6.1}$$

We consider encoding x_i and presume that y_i is available losslessly at the decoder. As described in Slepian & Wolf [78], by using Slepian-Wolf codes, one need only transmit at a rate equal to $H(x|y) = H(z \oplus y|y) = H(z) = H_2(Q)$. Thus if Q is known, the encoder can compress the source as much as if it had observed both sources, x and y.

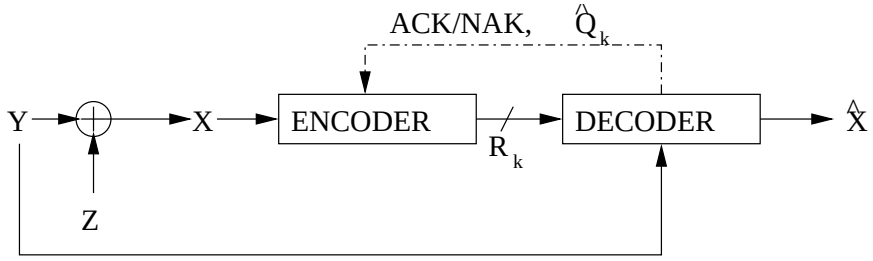


Figure 6.1. A block diagram for blind source coding with side information. For the analysis, we assume the simple correlation structure for x_i and y_i as shown here. During each cycle, the source is transmitted at rate R_k over the forward channel. The decoder feeds back an acknowledgment and an estimate of the parameter $\hat{Q}_k$.

We allow the compression system a reverse channel through which estimates of Q can be

fed back to the encoder. At time k the encoder maintains a state variable, $\hat{Q}_k$, corresponding to its current estimate of the joint statistics of x_i and y_i, and a rate-margin design parameter $\epsilon_k > 0$. We set $\hat{Q}_0 = 0.5$. The system works as follows:

1. The encoder divides the sequence x_i into block of length l_k, indexed by $k = 1, 2, \ldots$. We use $n_k = \sum_{j=1}^{k} l_j$ to denote the cumulative block-length. The k-th block corresponds to symbols $\mathsf{x}_{n_{k-1}+1}^{n_k}$. By choosing the cumulative block-length to double every transmission, i.e., $n_k = 2n_{k-1}$, the error analysis simplifies somewhat. We assume this choice for the duration of the analysis.

2. At step k, the encoder encodes $\mathsf{x}_{n_{k-1}+1}^{n_k}$ using a rate-R_k Slepian-Wolf code, where R_k depends on the estimate $\hat{Q}_k$ and margin ϵ_k as

$$R_k(\hat{Q}_{k-1}, \epsilon_k) = \begin{cases} \max[H_2(\hat{Q}_{k-1} + \epsilon_k), H_2(\hat{Q}_{k-1} - \epsilon_k)], & \text{if } \epsilon_k < |\hat{Q}_{k-1} - 0.5|, \\ 1, & \text{else,} \end{cases} \tag{6.2}$$

   i.e., the rate used is $H_2(\hat{Q}_{k-1})$ plus a margin. The various cases in Equation (6.2) come into play depending on whether $\hat{Q}_{k-1}$ is greater than or less than one-half, or within the margin ϵ_k of one-half.[1]

3. The decoder attempts to decode. If it can, it sends an acknowledgment to the transmitter, along with the updated estimate $\hat{Q}_k$. The estimate $\hat{Q}_k$ is simply the proportion of 1s (or 0s) observed in z_i thus far (i.e., the Hamming weight of $\mathsf{x}_i \oplus \mathsf{y}_i$). The feedback following the kth block can therefore be accomplished with $1 + \log_2 l_k$ bits. If the decoder cannot decode reliably, it sends a "NAK", i.e., a request for retransmission. It holds off sending the updated estimate until it receives the retransmission.[2]

4. If the encoder receives an acknowledgment, it moves onto the next block, using the updated estimate of Q. If the encoder receives a request for retransmission, it sends the sequence $\mathsf{x}_{n_{k-1}+1}^{n_k}$ uncompressed.[3]

If c_k is the transmission rate (in bits per symbol) of block k, we want to minimize the expected transmission rate, $E[\mathsf{c}_k]$, of the scheme. $E[\mathsf{c}_k]$ equals the expected rate $E[\mathsf{R}_k]$ of

[1] If we used the more standard choice, i.e., $H(\hat{Q}_{k-1}) + \epsilon_k$, we could avoid the multiplicity of cases of Equation (6.2). However, by expressing the margin in the current manner, our analysis simplifies, resulting in some nice close-form error expressions. Note that we can choose to add the margin directly to $\hat{Q}$ only because we focus on binary sources.

[2] We assume the code has perfect error-detection capabilities.

[3] More efficient hybrid-ARQ-type retransmission strategies can be used. But, this strategy simplifies the error analysis, and only leads to a marginal loss in efficiency.

the Slepian-Wolf code plus the probability of a decoding error times $\ln 2$ (1 bit/symbol or $\ln 2$ nats/symbol to account for its uncompressed retransmission). We use e_k to denote the event of a decoding error on block k. Ideally, we minimize the following cost,

$$E\left[\mathsf{c}_k\right] = \Pr[\mathsf{e}_k] \ln 2 + E\left[\mathsf{R}_k\right]. \tag{6.3}$$

With some abuse of notation we use R_k both to represent the rate-determination function, as in Equation (6.2) and the resulting rate, as in Equation (6.3). We choose to express the rate in nats to simplify notation in subsequent analysis.

6.3 Analysis

In this section we show that choosing $\epsilon_k = K(Q)/\sqrt{l_k}$ (for some function $K(\cdot)$) minimizes the cost Equation (6.3). For some constants ϕ_1, ϕ_2, and ϕ_3, we will develop a bound for $E\left[\mathsf{c}_k\right]$ of the following form

$$E\left[\mathsf{c}_k\right] \leq H_2(Q) + \phi_1 \epsilon_k + \phi_2 \exp\left\{-\phi_3 \epsilon_k^2\right\}.$$

By choosing ϵ_k of the form $\epsilon_k = K(Q)/\sqrt{l_k}$, we obtain the fastest asymptotic behavior of this bound. As one would expect, the form of ϵ_k should be chosen dependent on Q. However, Q is unknown, so in practice we would pick the constant as $K(\hat{Q}_{k-1})$. The dependence on $\sqrt{l_k}$ is also somewhat intuitive. The standard deviation in the best estimate of Q drops as $1/\sqrt{n_{k-1}} = 1/\sqrt{l_k}$.

Without loss of generality, in this analysis we assume that $Q + \epsilon_k < 0.5$. In Section 6.3.1 we first bound the first term of Equation (6.3), the probability of a decoding error. In Section 6.3.2 we bound the second term, the expected rate used during the Slepian-Wolf transmission. In Section 6.3.3 we put the results together to choose ϵ_k.

6.3.1 Probability of Decoding Error

We assume that the scheme uses Slepian-Wolf codes that succeed as long as the entropy rate of the kth block is below the transmission rate, i.e., $H(\mathsf{x}_{n_{k-1}+1}^{n_k})/l_k < \mathsf{R}_k$. We use standard large deviation techniques following the results of [22] to bound the probability of

decoding error on the kth block.

$$\Pr[\mathsf{e}_k] = \Pr[H(\mathsf{x}_{n_{k-1}+1}^{n_k})/l_k > \mathsf{R}_k] \tag{6.4}$$

$$= \sum_P \sum_{x^{n_{k-1}} \in \mathcal{T}_P} p(x^{n_{k-1}}) \Pr[H(\mathsf{x}_{n_{k-1}+1}^{n_k})/l_k > R_k] \tag{6.5}$$

$$= \sum_P \sum_{x^{n_{k-1}} \in \mathcal{T}_P} p(x^{n_{k-1}}) \sum_{\tilde{P} \text{ s.t. } H(\tilde{P}) > R_k(P,\epsilon_k)} \sum_{x_{n_{k-1}+1}^{n_k} \in \tilde{P}} p(x_{n_{k-1}+1}^{n_k}). \tag{6.6}$$

In Equation (6.4) the rate R_k is random, depending on the empirical distribution of the first n_{k-1} source symbols and the margin ϵ_k via Equation (6.2). In Equation (6.5) we use P to denote this empirical distribution and $\mathcal{T}_P$ to indicate the corresponding typical set. Since $\hat{Q}_{k-1} = P$, plugging P into Equation (6.2) gives the, now non-random, rate used $R_k(P, \epsilon_k)$. We continue as

$$\Pr[\mathsf{e}_k] \leq \sum_P \sum_{\tilde{P} \text{ s.t. } H(\tilde{P}) > H(P+\epsilon_k)} \exp\{-n_{k-1} D(P\|Q) - l_k D(\tilde{P}\|Q)\} \tag{6.7}$$

$$\leq \sum_P \sum_{\tilde{P}} \exp\{-l_k \min_{P, \tilde{P} \text{ s.t. } H(\tilde{P}) \geq H(P+\epsilon_k)} [D(P\|Q) + D(\tilde{P}\|Q)]\} \tag{6.8}$$

$$\leq (l_k + 1)^2 \exp\{-l_k [D(P^*\|Q) + D(P^* + \epsilon_k \| Q)]\} \tag{6.9}$$

In Equation (6.7) we use $p(x^{n_{k-1}}) = \exp\{-n_{k-1}[H(P) + D(P\|Q)]\}$ for all sequences $x^{n_{k-1}} \in \mathcal{T}_P$, and $|\mathcal{T}_P| \leq \exp\{n_{k-1} H(P)\}$, see [22]. In Equation (6.8) we use $l_k = n_{k-1}$, and the minimization is over all distributions, not just those that are types. In Equation (6.9) P^* is the minimizing distribution ($P = P^*$ and $\tilde{P} = P^* + \epsilon_k$). After minimization the exponent does not depend on P or $\tilde{P}$. We sum over all binary types of length l_k, of which there are $l_k + 1$.

The error exponent of the probability of decoding error depends on the unknown distribution Q. We study this exponent to determine a good choice for ϵ_k. To do this, we solve for P^* assuming a fixed ϵ_k.

$$\begin{aligned}
&\frac{d}{dP}[D(P\|Q) + D(P+\epsilon_k\|Q)] \\
&= \frac{d}{dP}\left[P \ln \frac{P}{Q} + (1-P)\ln\frac{1-P}{1-Q} + (P+\epsilon_k)\ln\frac{P+\epsilon_k}{Q} + (1-P-\epsilon_k)\ln\frac{1-P-\epsilon_k}{1-Q}\right] \\
&= \ln\left[\frac{P(P+\epsilon_k)(1-Q)^2}{(1-P)(1-P-\epsilon_k)Q^2}\right].
\end{aligned} \tag{6.10}$$

Setting Equation (6.10) equal to zero, and solving for P using the quadratic equation gives

$$P^* = -\frac{\epsilon_k}{2} - \frac{2Q^2 - \sqrt{\epsilon_k^2(1-2Q)^2 + 4Q^2(1-Q)^2}}{2(1-2Q)}. \tag{6.11}$$

For any choice of ϵ_k, and any source distribution Q, this value of P^* determines the dominant source of decoding error. Using Equation (6.11) in Equation (6.9) yields a bound on the decoding error for this protocol. Note that because $D(P\|Q)$ is convex in its arguments, $P^* \leq Q \leq P^* + \epsilon_k$.

The error exponent $D(P^*\|Q) + D(P^* + \epsilon_k\|Q)$ has a particularly simple form when ϵ_k is small. We define $P^* = Q - \bar{\epsilon}$ and $P^* + \epsilon_k = Q + \bar{\bar{\epsilon}}$, where $\epsilon_k = \bar{\epsilon} + \bar{\bar{\epsilon}}$. By the convexity property just discussed $\bar{\epsilon}, \bar{\bar{\epsilon}} > 0$. With these definitions, we approximate the error exponent when ϵ_k is small.

$$\begin{aligned}
&D(P^*\|Q) + D(P^* + \epsilon_k\|Q) = D(Q - \bar{\epsilon}\|Q) + D(Q + \bar{\bar{\epsilon}}\|Q) \\
&= (Q - \bar{\epsilon}) \ln\left[1 - \frac{\bar{\epsilon}}{Q}\right] + (1 - Q + \bar{\epsilon}) \ln\left[1 + \frac{\bar{\epsilon}}{1-Q}\right] \\
&+ (Q + \bar{\bar{\epsilon}}) \ln\left[1 + \frac{\bar{\bar{\epsilon}}}{Q}\right] + (1 - Q - \bar{\bar{\epsilon}}) \ln\left[1 - \frac{\bar{\bar{\epsilon}}}{1-Q}\right] \\
&\simeq \frac{\bar{\epsilon}^2 + \bar{\bar{\epsilon}}^2}{2Q(1-Q)} + \frac{(\bar{\epsilon}^3 - \bar{\bar{\epsilon}}^3)(1-2Q)}{2Q^2(1-Q)^2} \qquad (6.12) \\
&\geq \frac{\epsilon_k^2}{4Q(1-Q)} \geq \epsilon_k^2 \qquad (6.13)
\end{aligned}$$

In Equation (6.12) we use $\ln[1+x] \simeq x - x^2/2$. Writing $\bar{\epsilon}^2 + \bar{\bar{\epsilon}}^2 = (\bar{\epsilon} + \bar{\bar{\epsilon}})^2 - 2\bar{\epsilon}\bar{\bar{\epsilon}} = \epsilon_k^2 - 2\bar{\epsilon}\bar{\bar{\epsilon}}$, one can see that Equation (6.12) is minimized under the constraint for small ϵ_k by selecting $\bar{\epsilon} = \bar{\bar{\epsilon}} = \epsilon_k/2$. Choosing $Q = 0.5$ lower-bounds the quadratic approximation of the error exponent.

In Figure 6.2 we plot the error exponent and quadratic approximation to it for $Q = 0.05$ and $Q = 0.3$. The approximation Equation (6.13) is quite good, even for large values of ϵ_k. For $Q = 0.3$, one can barely distinguish the quadratic approximation to the full solution. The lowest curve in Figure 6.2 is the lower-bound to the quadratic approximation with $Q = 0.5$.

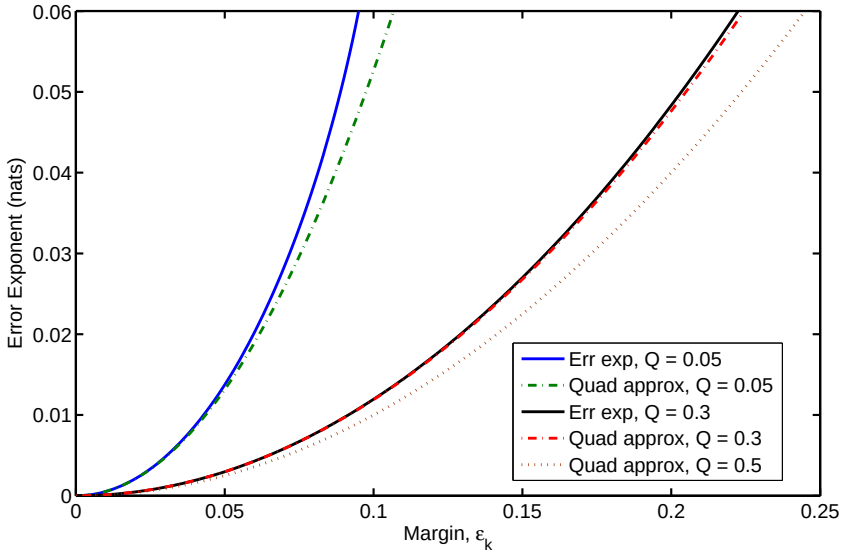


Figure 6.2. Error exponents and quadratic approximations for $Q = 0.05$ and $Q = 0.3$. Note, for $Q = 0.3$, the approximation is so close to the error exponent that the lines are on top of each other. This demonstrates that the quadratic approximation is quite good.

6.3.2 Bounding the Expected Slepian-Wolf Rate

In order to minimize the cost Equation (6.3) we must also take into account the second term of Equation (6.3), $E[R_k]$.

$$E[R_k] \leq \Pr[H(x^{n_{k-1}}) \leq H(Q+\gamma)]H(Q+\gamma+\epsilon_k) + \Pr[H(x^{n_{k-1}}) > H(Q+\gamma)]\ln 2 \quad (6.14)$$

$$\leq H(Q+\gamma+\epsilon_k) + \ln 2 \sum_{P \text{ s.t. } H(P) > H(Q+\gamma+\epsilon_k)} \sum_{x^{n_{k-1}} \in \mathcal{T}_P} p(x^{n_{k-1}}) \quad (6.15)$$

$$\leq H(Q+\gamma+\epsilon_k) + \ln 2 \sum_{P \text{ s.t. } H(P) > H(Q+\gamma)} \exp\{-n_{k-1} D(P\|Q)\}$$

$$\leq H(Q+\gamma+\epsilon_k) + \ln 2(l_k+1)\exp\{-l_k D(Q+\gamma\|Q)\} \quad (6.16)$$

In Equation (6.14) we split the expected rate into two events. The first is a high-probability event that occurs when the realized entropy is below the source entropy plus a margin γ. The second is a low-probability event that occurs when the realized entropy is large. In the former case, the code rate is upper bounded by $H(Q+\gamma+\epsilon_k)$, while in the latter it is upper

bounded by $\ln 2$. In Equation (6.15) we upper bound the probability of the high-probability event by one, and analyze the low-probability event using techniques similar to those used in Section 6.3.1. As in that section, we also examine the small-γ region which gives,

$$D(Q+\gamma\|Q) \simeq \frac{\gamma^2}{2Q(1-Q)}. \tag{6.17}$$

6.3.3 Optimizing the Rate Margin

We can now understand how ϵ_k should be chosen. It is easiest to see this in the small-ϵ_k region. Indeed, the small-ϵ_k region is of great interest as we want to be as efficient as possible for large data files.

Noting that probability of $\Pr[\mathsf{e}_k]$ in Equation (6.4) and $\Pr[H(x^{n_{k-1}}) > H(Q+\gamma)]$ in Equation (6.14) can both be upper bounded by one, then substituting Equation (6.9), Equation (6.11), Equation (6.16), and Equation (6.17) into Equation (6.3) gives

$$\begin{aligned} E[\mathsf{c}_k] &= E[\mathsf{R}_k] + \ln 2 \Pr[\mathsf{e}_k] && (6.18)\\ &\leq H(Q+\gamma+\epsilon_k) \\ &\quad + \ln 2 \min\left[1, (l_k+1)\exp\left\{-l_k \frac{\gamma^2}{2Q(1-Q)}\right\}\right] \\ &\quad + \ln 2 \min\left[1, (l_k+1)^2\exp\left\{-l_k \frac{\epsilon_k^2}{4Q(1-Q)}\right\}\right]. && (6.19) \end{aligned}$$

To give the best bound, we want to pick γ as small as possible in Equation (6.18). Picking $\gamma = \epsilon_k/\sqrt{2}$ balances the exponents. We combine the exponential terms and use a Taylor series expansion of the entropy around Q to simplify the bound of Equation (6.18), giving:

$$E[\mathsf{c}_k] \leq H(Q) + \frac{1+\sqrt{2}}{\sqrt{2}} \ln\left[\frac{1-Q}{Q}\right] \epsilon_k + 2\ln 2 \min\left[1, (l_k+1)^2\exp\left\{-l_k \frac{\epsilon_k^2}{4Q(1-Q)}\right\}\right]. \tag{6.20}$$

In Equation (6.20), the second term is linear in ϵ_k, so we want to pick ϵ_k as small as possible. However, the third term constraints this choice. The margin ϵ_k must go to zero slower than $1/\sqrt{l_k}$, else the polynomial in l_k that pre-multiplies the exponent will dominate[4]. Thus, the algorithm presented here will quickly converge to the case of an encoder aware of the source statistics.

[4]Note that to study the trade-off for small Q, one must use a better approximation to the entropy function than the quadratic one we used. For example $c_k - H(Q)$ should always be less than one bit.

6.4 Practical Considerations

In this section, we discuss the practical impact of two assumptions made in Sections 6.2 and 6.3. We first consider aspects of the protocol induced by the specific source structure and correlation model. Second, we then consider the idealized codes used. Relaxation of these assumptions is necessary for any implementation of the protocol presented here.

Though Sections 6.2 and 6.3 consider two sources with a specific correlation structure, we would like to consider more generalized source structures. For example, we would like a blind algorithm for arbitrary correlation structures over an arbitrary number of sources. Fortunately, the protocol presented can be naturally adapted to such sources just by modifying the contents of the feedback.

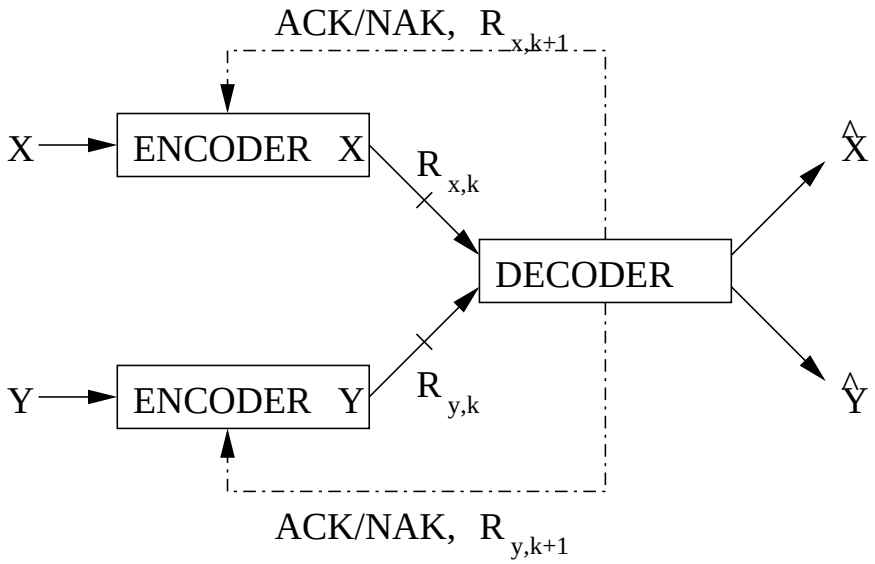


Figure 6.3. A block diagram for adaptation of our blind distributed source coding algorithm to the general distributed source coding problem. During each cycle, the source is transmitted at rate R_k over the forward channel. The decoder feeds back an acknowledgment and a rate request, R_{k+1}, for the next cycle.

In the protocol presented in Section 6.2, the feedback consists of the acknowledgment and the estimate of the source parameter, $\hat{Q}_k$. In our description of the protocol, there is no specific reason the encoder needs this detailed information. We modify our proposed protocol by altering the feedback to provide rate requests, R_{k+1}, instead of summary statistics. This structure can be adapted to arbitrary correlation structures and for an arbitrary number of sources. A block diagram for the 2 source scenario considered in Chapter 5 is

given in Figure 6.3. Since the accuracy of the rate request can be to a constant precision, the rate of feedback becomes negligible as the block length grows.

Second, we consider the assumption on the properties of the codes used from Sections 6.2 and 6.3. In those sections, three assumptions were made about the codes. First, we assume that the codes used respond to every source sequence of equivalent type equivalently. Second, we assume that the codes are capacity achieving. Finally, we assume that decoding errors are perfectly detectable.

In order to implement this algorithm with practical codes, we use LDPC based codes as introduced in Section 2.3 and used throughout this dissertation. These codes are a class of powerful linear block codes that are capacity approaching. They are decoded using the iterative sum-product algorithm, and that as the algorithm progresses it either converges to a solution that satisfies the code's constraints (most likely the maximum likelihood solution) or fails to converge at all.

Though LDPC codes do not satisfy the three assumptions stated here, they have properties that allow them to perform approximately meet the assumptions quite well. Though LDPC codes do not respond to all sequences of a particular type equivalently and are not capacity achieving, they perform well in practice. We leverage this empirical performance here knowing that we will pay a small rate penalty. Further, since LDPC based Slepian-Wolf systems approach the minimal compression rate bound only for large block lengths, as discussed in Chapters 3 and 5, extra redundancy is needed for the short block-lengths used for the first few blocks of our protocol. Finally, we leverage the iterative decoder of LDPC codes by setting a small number as the maximum number of iterations before a decoding failure is declared. We can use these declared decoding failures as a strong approximation of the exact error detection assumed in Section 6.2. As will be seen in Section 6.5, these effects are minimal.

6.5 Results

In this section we present results from running our blind algorithm. In each of the simulations, blocks of $100,000$ source symbols are generated according to a distribution. The initial block-length is set to 100, and each successive block length equals the number of bits sent thus far. As discussed below, we use a selection of 38 codes. Since not all rates are available, some additional redundancy is introduced as rates are rounded up to the nearest

available rate. The redundancy parameter ϵ_k is set to $\epsilon_k = 2.0/\sqrt{n_{i-1}}$ (arbitrarily, though resulting in good performance).

For these experiments, we generate a finite set of LDPC codes from a selection of LDPC degree distributions [70]. We use 38 different degree distributions (each both check and variable irregular), with compression rates ranging from 0.025 to 0.95, with a spacing of approximately 0.025 between codes. Most degree distributions are obtained from the binary symmetric channel design section of the LTHC website [4]. The rest of the degree distributions we use are designed using EXIT (extrinsic information transfer) charts [7], though similar performance is seen by selecting codes from LTHC designed for other channel models (such as the binary additive white Gaussian noise channel). Using these degree distributions, the transform matrices are generated randomly, and then pruned to remove short loops (of length 4 or less). For greater detail, see Appendix A

We begin by considering the simple 2 source scenario presented in Section 6.2 and shown in Figure 6.1. We plot the results in Figure 6.4. In these plots the average cumulative redundancy in percent (averaged over 25 simulations) is plotted versus cumulative block-length, n_k. Cumulative redundancy is defined as $\sum_{j=1}^{k} \frac{[d_j - H(Q)]l_j}{H(Q)n_k}$. The results of the system are plotted for 2 different z_i entropy rates: 0.1791 and 0.1944. As an example, to transmit 10^5 bits for a z_i entropy $H(Q) = 0.1791$ bits required an average redundancy of 49%, or 8,875 bits more than the 17,912 bit minimum (26, 787 bits total). In these plots, as n_k grows the overall redundancy declines. To demonstrate consistency between simulations, we plot one standard deviation error bars. In addition, for a z_i of entropy 0.1791 (the other z_i is omitted for clarity, but is similar), a bound on the expected cumulative redundancy using Equation (6.18) is also plotted, as well as the performance assuming full knowledge of the source statistics (based on the results of [76]). Despite the limitations mentioned, and the fact that our bound omits the cost of feedback, our results perform well in relation to our bound.

For the next simulation, we consider a correlation structure requiring the rate request structure of Section 6.4. Here, we retain the binary sum of z_i to x_i to obtain y_i as depicted in Equation (6.1). In contrast, we introduce Markov memory to z_i. We now parameterize the source as follows

$$p = Pr(z_i = 1),$$

$$h_0 = Pr(z_i = 1|z_{i-1} = 0), \quad \text{and } h_1 = Pr(z_i = 1|z_{i-1} = 1).$$

Greater discussion of this model can be found in Chapter 7.

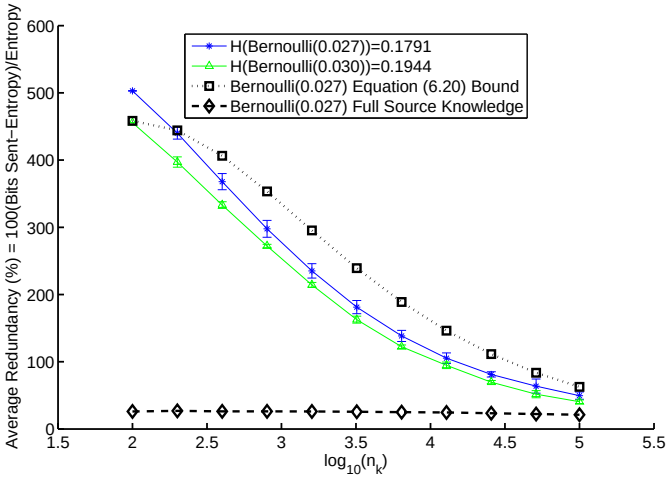


Figure 6.4. Averaged results (over 25 trials) of blind source coding with side information of sources as modeled in Section 6.2. The horizontal axis is the number of cipher-text bits considered, n_k, in log-scale, and the vertical axis is the system redundancy (the percent of bits used above the entropy rate). We present one standard deviation error bars to demonstrate consistency. As the number of bits transmitted grows performance improves. Note, the bound from Equation (6.18) plotted here excludes feedback costs. Ideally, we would like to transmit at 0% redundancy (the entropy rate).

In Figure 6.5, we plot the results of the blind source compression with side information of x_i when z_i has the Markov memory structure described here. We parameterize z_i as $(p, h_0, h_1) = (0.3571,\ 0.1103,\ 0.8014)$. This results in a conditional entropy rate of $H(x_i|y_i) = 0.5787$. On average, these simulations transmitted $80,000$ bits in $47,951$ bits, a 3.56% redundancy. For purpose of contrast, we also plot the redundancy of transmitting x_i uncompressed. As can be seen in Figure. 6.5, the blind transmission scheme here approaches the entropy rate for correlated sources.

6.6 Conclusions & Open Problems

In this chapter we presented a protocol for blind distributed source coding. We analyzed the protocol in the case of a simple correlation model and showed its convergence to the case of full knowledge of the source statistics. We then discussed how to actually implement this protocol, and demonstrated its practical performance.

Some issues stand out for future study. First, we would like to obtain a lower bound on

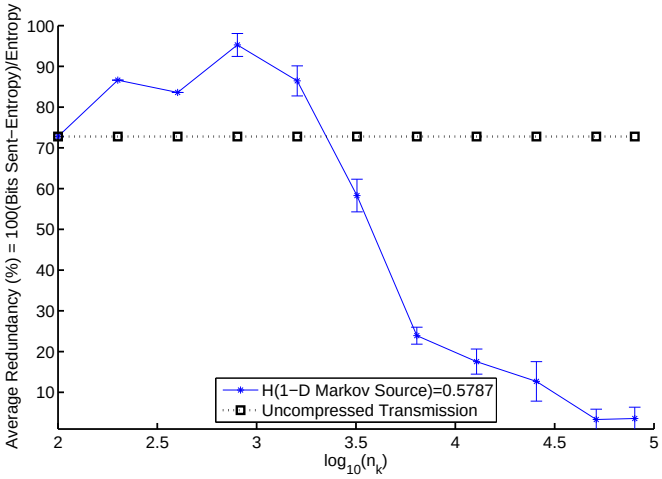


Figure 6.5. Averaged results (over 25 trials) of blind source coding with side information where the source model of Section 6.2 is altered such that the samples of z_i exhibit Markov memory. Similar to Figure 6.4, the horizontal axis is the number of cipher-text bits considered, n_k, in log-scale, and the vertical axis is the system redundancy (the percent of bits used above the entropy rate). As the number of bits transmitted grows performance improves.

transmission redundancy to help establish the optimality of our scheme. Second, we would like to implement the algorithm with other classes of codes and decoders. For example, the linear programming LDPC Decoder [24] offers a guarantee of maximum likelihood decoding or error detection offers promise.

Having developed the tools to perform practical distributed source coding throughout the previous chapters, we now consider a practical application of these codes. We apply these techniques to the compression of encrypted data to demonstrate how these codes can be used to solve new problems.

Chapter 7

The Compression of Encrypted Data: A Framework for Encrypted Simple Sources to Encrypted Images

In this chapter, and the next, we switch our focus from generally applicable distributed source codes to developing codes for the application of compressing encrypted data. In these chapters, we consider sources characterized by statistical redundancy, such as images, that have been encrypted uncompressed. Since encryption masks the source, traditional data compression algorithms are rendered ineffective. However, as has been shown previously, the compression of encrypted data is in fact possible through the use of distributed source-coding techniques. This means that it is possible to reduce data size without requiring that the data be compressed prior to encryption. Indeed, under some reasonable conditions, neither security nor compression efficiency need be sacrificed when compression is performed on the encrypted data (Johnson et al., 2004 [38]).

Building on this foundation, in this chapter we develop algorithms for the practical lossless compression of encrypted data sources. We consider algorithms for a variety of sources, ranging from independent and identically distributed to images. Our methods are designed to leverage the statistical correlations of the source, even without direct access to

their instantiations. As an example, we are able to compress a encrypted binary version of the world map to 43% of its original size, giving a 57% rate savings. Traditional data compression algorithms are unable to compress the encrypted image at all.

7.1 Introduction

Since *good* encryption makes a source look completely random, traditional algorithms are unable to compress encrypted data. Typical systems therefore must compress before they encrypt. However, Johnson et al. in [38] show that the compression of encrypted data can be cast as a problem of source coding with side information [78]. It is further shown in [38] that neither compression performance nor security need be lost under some reasonable conditions. A block diagram of this system structure is presented in Figure 7.1. In this figure, a length n source message, x^n, is encrypted with, k^n, to produce the cipher-text, y^n. The cipher-text is compressed into nR bits ($R < 1$) and the decoder makes a source estimate, $\hat{x}^n$, based on the nR bits and k^n.

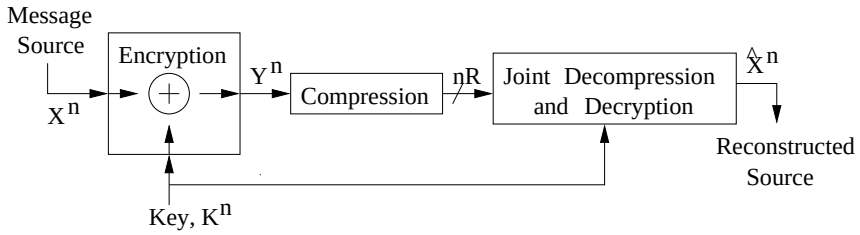


Figure 7.1. The source is first encrypted and *then* compressed. The compressor does not have access to the key used in the encryption step. The decoder jointly decompresses and decrypts. The key serves as the decoder side information.

To better understand the problem framework, consider the image in Figure 7.2. In the left-hand plot is a binary image of the world map. We refer to this as the plain-text since it is unencrypted. Each of the 100×100 pixels in this frame takes a value of either 0 or 1, giving a raw images size of $10,000$ bits. The middle plot of the figure shows an image of equal dimension to the first, but consisting of bits selected uniformly at random (i.e., independent and identically distributed (i.i.d.) Bernoulli-0.5 random variables). This image is the stream cipher key. We encrypt the image by applying a bitwise exclusive-OR (XOR) between the key bits and the source (this is an example of a "stream cipher," and has the same security properties as the Shannon one-time pad [77]). The resulting encrypted image

shown in the right-hand plot is referred to as the cipher-text. The problem of compressing the highly structured unencrypted image on the left has been extensively researched, and many effective solutions exist. However, none of these can compress the *marginally random* image on the right. The cipher-text, in isolation, is independent of the source, is itself a Bernoulli-0.5 i.i.d. image, and is thus not compressible.

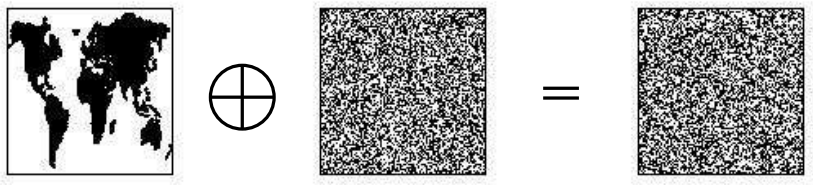


Figure 7.2. A sample 100×100 world map image is on the left ($10,000$ bits total). To encrypt this image, the $10,000$ bit random key in the center is added to the unencrypted image on the left (using a bit-wise exclusive-OR).

While the plain-text and cipher-text are independent, the cipher-text can still be compressed using techniques of source coding with side information. To understand why, note that compression can be achieved by *leveraging the dependence* between the cipher-text, y^n, and the key, k^n. This dependence can be understood by viewing the cipher-text as a noisy version of the key stream. For example, if we know that the source has a low Hamming-weight[1], then the set of possible cipher-texts are clustered around the key sequence. Viewing the key sequence as a noisy version of the cipher-text, we use a Slepian-Wolf decoding process [78, 22] on the compressed bits to de-noise the key and uncompress the cipher-text. The source estimate is produced by decrypting the uncompressed cipher-text.

In addition to establishing the connection between compression of encrypted data and Slepian-Wolf coding, Johnson et al. [38] demonstrates a practical scheme for compressing an encrypted i.i.d. source. This scheme is based on graphical codes, and draws from the work presented in Chapter 3. However, the main target applications for these techniques are in media coding, where the statistics of sources such as images are far from i.i.d.

In this chapter, we develop a framework for the compression of encrypted media sources. Since we operate on encrypted data, the encoder has no access to the original (compressible) source. Therefore, only the decoder can exploit source structure. The accuracy of the

[1]That is, it is mostly zeros and contains only a few ones. Though most data sources do not have low Hamming-weight, we can leverage source redundancy to the same effect.

statistical model used is thus essential to the resulting compression performance. Earlier work [38] focuses on memoryless models and ignores any spatial and temporal dependencies. Our contribution is to develop and implement a practical solution for compressing encrypted media.

We consider the issues in compressing media sources. We introduce 1-D and 2-D models which can leverage the spatial structure of images. We show how to exploit increased model complexity for improved performance. We apply these models to both binary and gray scale images, and we demonstrate that employing these models achieves greater compression than the i.i.d. source model.

This chapter is organized as follows. In Section 7.2 we describe our framework for compressing encrypted sources. Then, in Section 7.3 we develop statistical models and provide experimental results. Finally, we provide concluding notes and open problems in Section 7.4.

7.2 General Framework

In this section we present the framework of our design. We then specialize it to the scenario of interest (stream-cipher encryption and compression using linear codes). We use factor graphs [42] (discussed in greater detail in Section 2.3) to represent the statistical relationships between quantities in our problem.

As a brief summary of the material in Section 2.3, we give a quick summary. Factor graphs are bipartite graphs, consisting of two types of nodes: variable nodes (represented by circles) and factor nodes (represented by rectangles or squares). Variable nodes are either hidden quantities we wish to estimate (such as the plain-text) or observed quantities (such as the key). Factor nodes represent constraints between subsets of variable nodes. The set of variable nodes constrained by a particular factor are connected to the factor node by edges. Factor graphs not only concisely represent the statistical structure of a statistical inference problem, but also provide the basis for efficient inference algorithms.

The high-level factor graph for our problems is depicted in Figure 7.3. The left half of the factor graph depicts the generative model (the mapping from source and key to the compressed bits), while the right half is used by the decoder to estimate the source. The generative model contains the mapping from the source[2] x^n and key k^n through encryption

[2]We exclusively consider binary sources herein. This is not as restrictive an assumption as it may first

into the cipher-text y^n. The "encryption function" factor node constrains x^n, k^n and y^n to be compatible inputs and outputs of the encryption function. The generative model also contains the code used to map the cipher-text y^n into the syndrome[3] s^m. The compression rate is m/n. The "code constraint" factor node constrains y^n and s^m to be compatible inputs and outputs of the encoding function. The right-half of the factor graph is used by the decoder to estimate the plain-text. It has as inputs the key k^n and syndrome s^m and as internal variables the source and cipher-text estimates, $\hat{x}^n$ and $\hat{y}^n$, that are the output of the decoder. We now discuss how some of the high-level factor nodes of Figure 7.3 can be further decomposed into more tractable low-level functions. We describe these factorizations in terms of the variables on the right-hand decoding half of the graph, as it is in decoding that these simplifications will be exploited.

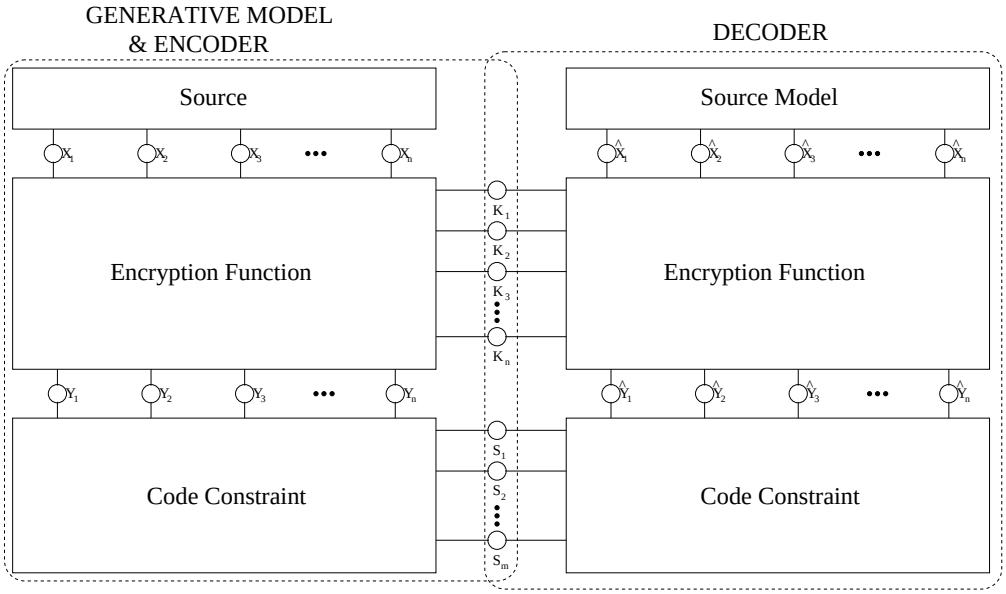


Figure 7.3. The abstracted model for compressing encrypted data. This model consists of two parts. The first part, the generative model and encoder, describes how the raw source data is first encrypted and then compressed. This part operates on the true source bits, x_i, the key bits, k_i, the encrypted bits, y_i, and the compressed bits, s_i. The second part, the decoder, shows how the various components of the source are modeled at the decoder for recovering an estimate of the original source. This part uses the same compressed bits, s_i, and key bits k_i, but works on an estimate of the encrypted bits, $\hat{y}_i$, and the source bits, $\hat{x}_i$.

appear. For instance, gray-scale image can be broken down into bit planes with the source variables indicating the bit-plane pixel values.

[3]We refer to the compressed bits as a "syndrome" to make explicit the connection to distributed source coding and to Chapter 3.

7.2.1 Encryption Function

In this dissertation we consider stream-cipher encryption exclusively. Such encryption is used in a variety of symmetric key encryption standards, and (under our assumption of an i.i.d. Bernoulli-0.5 key) are Shannon-sense secure [77]. The structure of stream-ciphers allows us to factor the larger encryption process into a set of bit-by-bit processes. The cipher-text is computed as $\hat{y}_i = \hat{x}_i \oplus k_i$, where $\hat{y}_i$ is the ith bit of the cipher-text estimate, k_i is the ith bit of the key, and $\oplus$ indicates addition mod-2 (XOR).

At decoding a parallel constraint must be enforced on $\hat{y}_i$, $\hat{x}_i$, and k_i for every i. In particular, these variables must sum to zero. We therefore split the large encryption factor node into n low-level factor nodes. As depicted in Figure 7.4, the square node labeled f_i enforces the even parity.

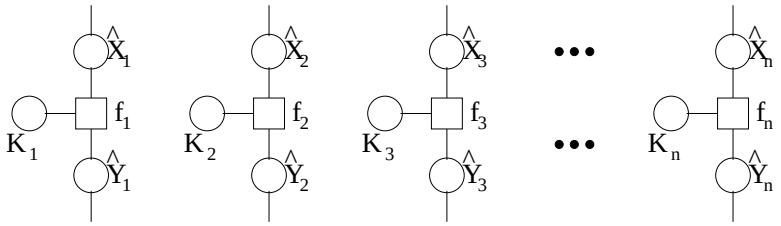


Figure 7.4. The graphical constraints imposed by stream-cipher encryption. The source estimate $\hat{x}_i$, the key bit k_i, and the cipher-text estimate $\hat{y}_i$ must have even parity.

7.2.2 Code Constraint

We use a linear transform to compress the cipher-text. The syndrome is calculated as

$$s^m = \mathbf{H} y^n,$$

where $\mathbf{H}$ is an $m \times n$ binary matrix, and addition is performed modulo two. This is the same technique used for compression in Section 3.2.

We choose a low-density parity-check (LDPC) code [26] as the linear transform. Further discussion of the LDPC codes we use and how we modify then is provided in Section 2.3 and in our experimental descriptions in Section 7.3.4 respectively. See Chapters 3 and 5 for a more extensive discussion of code design and the merits of using LDPC codes as a base.

The $\mathbf{H}$ matrix corresponding to a LDPC code is a sparse matrix, i.e., it has few ones in any column or row. As detailed later, this choice makes efficient application of iterative

message-passing inference possible. As with the encryption function, the "code constraint" function node of Figure 7.3 can also be factored into local functional constraints, as shown in Figure 7.5. In particular, $\mathbf{H}$ can be described by m local constraints, each corresponding to a row of the $\mathbf{H}$ matrix. For example, say that there is a non-zero (i.e., unity) entry in the ith column and jth row of $\mathbf{H}$. Then in the factor graph we draw an edge connecting the ith variable $\hat{y}_i$ to the jth parity constraint g_j. The sum of all $\hat{y}_i$ connected to g_j must equal s_j.

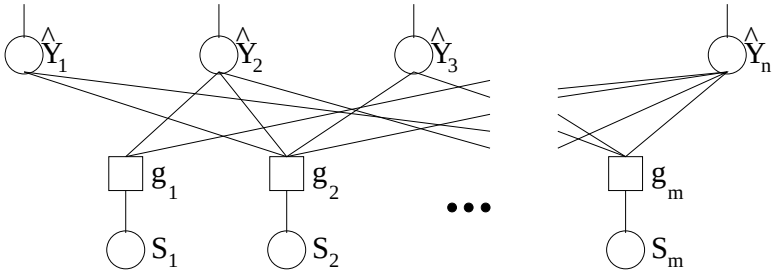


Figure 7.5. The graphical model of the constraints imposed by using a LDPC code for compression. All cipher-text estimate $\hat{y}_i$ connected to the same parity-check g_j must sum to the syndrome value s_j.

Here, we implicitly assume that the code provides sufficiently many compressed bits for the decoder to recover the source. In practice the encoder needs to learn how many bits it must supply. We discuss a relaxation to this assumption in Chapter 6.

7.2.3 Source and Source Model

Unlike the encryption and compression functions, where exact description of the imposed constraints is possible, we cannot have exact knowledge of the underlying characteristics of the general source. Instead, the decoder must use a model of the source. Thus, in Figure 7.3 the left-hand side function is labeled "source" (as determined by nature) while the corresponding function on the right-hand side is labeled "source model" (chosen by the system designer). The compression rates our system can achieve depend strongly on the quality of the model.

We first discuss the simple n-bit i.i.d. source model of Johnson et al. [38] for the "source model" constraint. This is the same model discussed in Chapter 3. We defer the discussion of more sophisticated models to Section 7.3 and Chapter 8. The i.i.d. model is represented

in Figure 7.6. This graph consists of the n variable nodes corresponding to source bits estimates, labeled $\hat{x}_i$, and the n function nodes corresponding to the source priors, labeled P_i. The source prior P_i is the marginal probability that each bit equals one, denoted $p = \Pr(\hat{x}_i = 1)$. For now, we assume the decoder is supplied with knowledge of p. We show how to relax this assumption in Section 7.3.4.

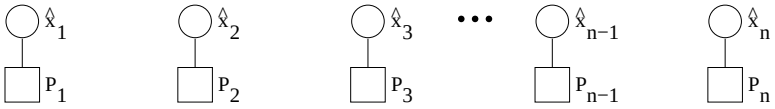


Figure 7.6. The simple i.i.d. model considered in [38] and in Chapter 3. This model consists of the source estimates $\hat{x}_i$ and their priors.

7.2.4 Decoding Algorithm

We recover the plain-text source using the sum-product message passing decoding algorithm. We give a brief summary of the algorithm. For greater depth, see Section 2.3.1. The sum-product algorithm is an inference algorithm that works on the factor graph. When the factor graph is a tree, the algorithm produces the correct variable estimates. It is not exact when there are loops in the factor graph (such as in the code graph of Figure 7.5). Empirically, however, its performance is quite good on average with reasonable computational complexity (the number of calculations grows polynomially with the block length). This is due both to code sparsity (thus its loops are typically quite long) and the slowly varying[4] nature of the source.

The algorithm works by iteratively sending messages from node to node along the edges of the factor graph. Each message pertains to a single variable node and represents the algorithm's current belief of whether its value is zero or one. Recall that all variables in our system are binary. We thus use log-likelihood ratios $\log[(P(x_i = 0))/P(x = 1)]$ to represent beliefs. By using this parameterization we only need to propagate scalar messages. Incoming messages are fused together to produce updated beliefs. The sum-product algorithm operates over the factor graph of Figure 7.3, with the simplifications developed in Section 7.2.1–Section 7.2.3. As message passing iterations progress, statistical information is disseminated across the graph. The algorithm continues until either a stopping condition

[4]For most models considered here, there is strong dependency between adjacent bits (with the exception of the i.i.d. model). See Section 7.3 and Chapter 8.

is met (e.g. if we compress our current best guess of $\hat{y}^n$ and the result is s^m) or a maximum number of iterations is reached.

We now describe the rules used to fuse together incoming messages into refined beliefs at each node. Messages coming into a variable node are added together to produce an updated log-likelihood estimate of the value of the node. The rule for fusing messages incoming to the ith parity constraint (g_i) is nearly identical to the update rule for LDPC codes [70], altering the sign as dictated by s_j. In addition g_i's fusion rule takes into account the value of the ith syndrome. See Section 3.4.1 for examples of the specific form of the update rules. The update rule for encryption function nodes (f_i) is simple: flip the sign of the log likelihood ratio if the key bit is 1,

$$\mu_{f_i,\hat{x}_i} = (-1)^{k_i} \nu_{\hat{y}_i,f_i}.$$

The message update rules for the specific source models we implement are discussed in Section 7.3 and Chapter 8.

In this section we have made specific choices for the code and encryption constraints of Figure 7.3. We chose these particular structures for their practical strengths, since they interact well together in sum-product algorithm. Other choices could be made for these constraints, but may result in significantly increased computational complexity. In the following section, we discuss various aspects of the source constraint.

7.3 From Compression of Encrypted Simple Sources to Encrypted Images

We now focus on the issue of source modeling. In this section, we consider source models of complexity greater than the i.i.d. source model from Section 7.2.3. Each source model is intended for use with the factor graph of Figure 7.3, is described in terms of the decoder half (right-hand side) of Figure 7.3, and is of increasing complexity with better compression performance. Since we focus on bit-wise stream ciphers in Section 7.2.1, each of our models is also bit based. As we describe each model, we illustrate them using factor graphs. We omit discussion of blind transmission for clarity.

7.3.1 1-D Markov Model

The first extension of the i.i.d. model we consider is Markov memory between successive bits. The factor graph of our 1-D Markov model is shown in Figure 7.7. In this figure, we again consider the n bits, labeled $\hat{x}_i$, and the source priors, labeled P_i. Additionally, we now consider the correlation between consecutive bits, shown as the constraints labeled M_i. The correlation constraints represent the Markov state transitions. In addition to parameterizing the source priors with the marginal probability $p = \Pr(\hat{x}_i = 1)$, we now also parameterize the Markov correlation constraints. Assuming these correlations to be equivalent both forwards and backwards, we denote these correlations $h_0 = \Pr(\hat{x}_i = 1|\hat{x}_{i-1} = 0) = \Pr(\hat{x}_i = 1|\hat{x}_{i+1} = 0)$ and $h_1 = \Pr(\hat{x}_i = 1|\hat{x}_{i-1} = 1) = \Pr(\hat{x}_i = 1|\hat{x}_{i+1} = 1)$.

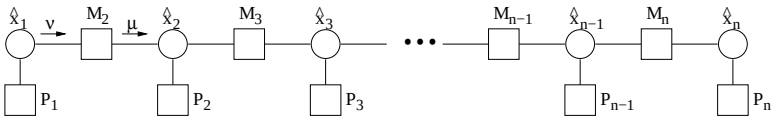


Figure 7.7. The 1-D source model. This model consists of bits (circles), source priors (squares below the bits) and correlation constraints (squares between the bits). In addition, message labels, μ and ν, are presented.

7.3.2 2-D Markov Model

In order to incorporate the spatial correlations of sources such as images, we introduce a 2-D Markov model[5]. We consider the n bits as arranged according to a grid, with N_v rows and N_h columns ($N_v \times N_h = n$). In addition to the source prior, we consider the correlation between each pixel $\hat{x}_{i,j}$ and its 4 nearest neighbors; up & down, left & right. A section of the corresponding factor graph is illustrated in Figure 7.8. Besides the circles labeled $\hat{x}_{i,j}$ ($i \in \{1, \ldots, N_h\}$ and $j \in \{1, \ldots, N_v\}$) representing the bits and the squares labeled $P_{i,j}$ representing the source priors, the constraints labeled $M^h_{i,j}$ represent the horizontal correlations while the those labeled $M^v_{i,j}$ represent the vertical correlations.

As with the other models, the prior probability on each bit is denoted $p = \Pr(\hat{x}_{i,j} = 1)$. For the 2-D model, we denote the horizontal correlation parameters as $h_0 = \Pr(\hat{x}_{i,j} = 1|\hat{x}_{i,j-1} = 0) = \Pr(\hat{x}_{i,j} = 1|\hat{x}_{i,j+1} = 0)$ and $h_1 = \Pr(\hat{x}_{i,j} = 1|\hat{x}_{i,j-1} = 1) = \Pr(\hat{x}_{i,j} =$

[5]The constructions of [82, 65], developed concurrently, are related but consider an image and a noisy version of the same image. Since neither the key nor the cipher-text are images here, their constructions do not directly apply.

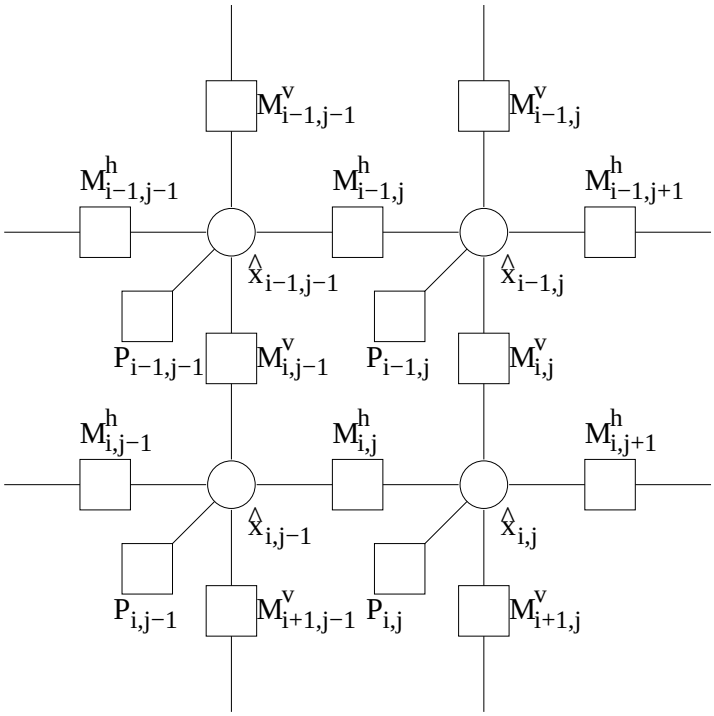


Figure 7.8. A section of the 2-D source model. This model includes the source bits (circles) and both horizontal correlations, $M^h_{i,j}$, and vertical correlations, $M^v_{i,j}$.

$1|\hat{x}_{i,j+1} = 1)$. We further denote the vertical correlation parameters as $v_0 = \Pr(\hat{x}_{i,j} = 1|\hat{x}_{i-1,j} = 0) = \Pr(\hat{x}_{i,j} = 1|\hat{x}_{i+1,j} = 0)$ and $v_1 = \Pr(\hat{x}_{i,j} = 1|\hat{x}_{i-1,j} = 1) = \Pr(\hat{x}_{i,j} = 1|\hat{x}_{i+1,j} = 1)$ for the 2-D model.

7.3.3 Message Passing Rules

Incorporating the models above into the general framework of Figure 7.3 is straightforward. In this section, we give the form of the additional message update rules necessary that were not covered in either Section 7.2.4 or Section 2.3.1. There are two additional types of nodes considered here; the source prior and the correlation constraints. Since each source prior is a terminal node, its messages are constant across iterations and is the log-likelihood ratio of the bit according to the prior, p.

As a representative example, we write the update rule for the 1-D message (Figure 7.7) from M_2 to $\hat{x}_2$ (labeled μ) based on the message from $\hat{x}_1$ to M_2 (labeled ν). All other correlation constraint message updates (messages in the alternate direction as well the

messages update rules for $M^h_{i,j}$ and $M^v_{i,j}$) are identical in form. Recall the parameters for M_2 here; $h_0 = Pr(\hat{x}_i = 1|\hat{x}_{i-1} = 0)$ and $h_1 = Pr(\hat{x}_i = 1|\hat{x}_{i-1} = 1)$. The update equation is given below,

$$\mu = \log\left(\frac{(2 - h_0 - h_1) + (h_1 - h_0)\tanh(\nu/2)}{(h_0 + h_1) + (h_0 - h_1)\tanh(\nu/2)}\right). \tag{7.1}$$

M_2 thus converts the estimate of $\hat{x}_1$ to an estimate of $\hat{x}_2$, based on the correlation between $\hat{x}_1$ and $\hat{x}_2$.

7.3.4 Doping

As mentioned in Section 7.2.2, the codes used to compress these encrypted sources are modified forms of LDPC codes. The code consists of bits calculated using a sparse linear transformation (based on an LDPC matrix) and a portion of the encrypted bits transmitted uncompressed. We refer to these uncompressed bits as "doped" bits. In practice, the doped bits range from 30% to 50% of the total output compressed bits (see Section 7.3.5 for sample empirical results). We use these bits in two ways[6].

First, since doped bits are known unambiguously at the decoder they anchor the iterative decoder and initiate the decoding process. Without these bits as a catalyst, decoding fails. Decoding fails since marginally each source bit is roughly uniform, and thus the log likelihood ratios lack bias. This aspect of their use is similar to their use in the distributed source coding scheme of Markov sources in García-Frías & Zhong [34] and fountain codes [51].

Second, they additionally provide a mechanism to estimate the statistics of the encrypted data. As mentioned in Section 7.2.4, we assume that the decoder knows the source statistics. Using doped bits, we can eliminate this assumption. By selecting an appropriate subset of the source to send as doped bits, the decoder develops estimates of the source parameters (p, h_0, *etc.*). For example, with Markov source models we dope adjacent bits for parameter estimation. In this work, we dope bits uniformly at random. Study of more structured doping schemes is an open problem. For instance, it may be that doping bits according to variable node degree of the compression code results in better sum-product algorithm performance.

Though we refer to these bits specifically, they could be incorporated into the code's degree distribution. Each bit transmitted doped is equivalent to adding a degree 1 check

[6]Though not a part of this discussion, one additional use of these "doped" bits would be in initialization of the blind transmission protocol of Chapter 6.

node to the sparse linear transform. A study of code design techniques [70] incorporating doped bits is an open problem.

7.3.5 Experimental Results

We use an example to demonstrate the compression results. Consider the image in Figure 7.9, reprinted from Figure 7.2. We encrypt this binary world map image ($10,000$ bits) and decompress it using the various idealized source models. Note that for the 1-D Markov model, we use a raster scan (north to south, than east to west) of the image. Through a sequence of trials, we determine the minimum rate we can compress the encrypted image to and still recover the original source exactly and then present that rate here.

For these experiments, we generate a finite set of LDPC codes from a selection of LDPC degree distributions [70]. We use 38 different degree distributions (each both check and variable irregular), with compression rates ranging from 0.025 to 0.95, with a spacing of approximately 0.025 between codes. Most degree distributions are obtained from the binary symmetric channel design section of the LTHC website [4]. The rest of the degree distributions we use are designed using EXIT (extrinsic information transfer) charts [7], though similar performance is seen by selecting codes from LTHC designed for other channel models (such as the binary additive white Gaussian noise channel). Using these degree distributions, the transform matrices are generated randomly, and then pruned to remove short loops (of length 4 or less). For greater detail, see Appendix A.

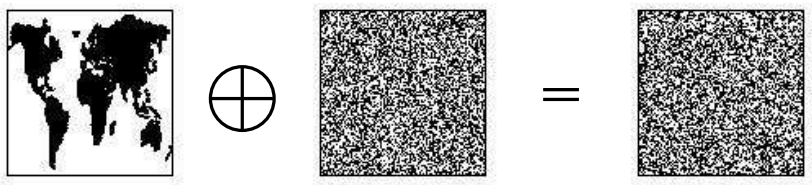


Figure 7.9. A sample 100×100 world map image (also shown in Figure 7.2) is on the left ($10,000$ bits total). To encrypt this image, the $10,000$ bit random key in the center is added to the unencrypted image on the left (using a bit-wise exclusive-OR).

Our first attempt to decompress the world map uses the i.i.d. model. Based on the doped bits we measure this image, under an i.i.d. source model, to have parameter (p) = (0.38) and thus entropy of about 0.9580 bits. In practice, we are unable recover the

source no matter what code rate we use to compress the encrypted image. Using the 1-D model, we estimate $(p, h_0, h_1) = (0.3738, 0.0399, 0.9331)$, and the encrypted image could be compressed to $7,710$ bits. In that simulation, the image was reconstructed in 27 iterations[7]. Finally, we compress the encrypted world map image to $4,299$ bits using the 2-D model, of which $2,019$ are doped bits. The decoder empirically estimates $(p, h_0, h_1, v_0, v_1) = (0.3935, 0.0594, 0.9132, 0.0420, 0.9295)$ and then reconstructs the original image using 81 iterations. The compressed bits and the reconstruction are presented in Figure 7.10.

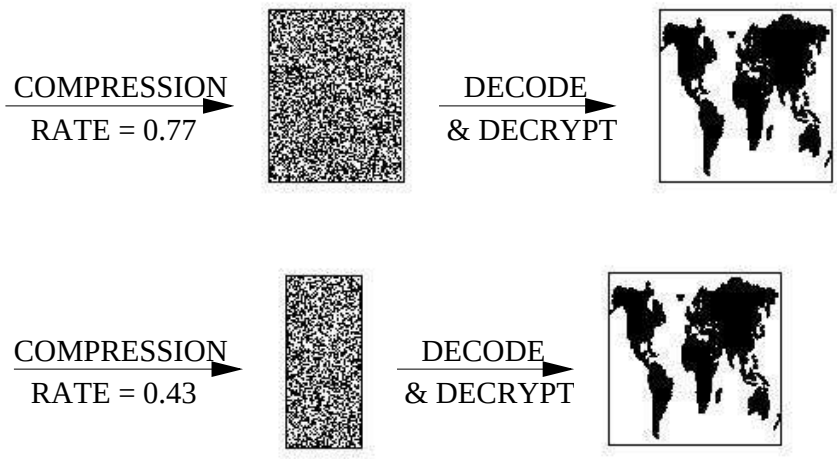


Figure 7.10. A comparison of the compressed bits and reconstructed image using the 1-D Markov model and the 2-D Markov model. The 1-D model compressed the encrypted data to $7,710$ bits. This is $3,411$ bits more than the $4,299$ bits used for the 2-D model. The i.i.d model could not compress the encrypted image at all.

To demonstrate the effects of the source model on decoding, we present the estimates of the 1-D and 2-D decoders at three iterations in Figure 7.11. The influence of the 1-D source model on decoder estimates in the estimates convergence along north-south lines. When the 2-D source model is used in contrast, the estimates converge in "clumped" regions seeded by the doped bits that grow from iteration to iteration.

[7]Note that we did not study the effect of the model on the number of iterations required to decode. Empirical results tended to show that a minimal number of iterations are required unless the code rate is close to the minimum.

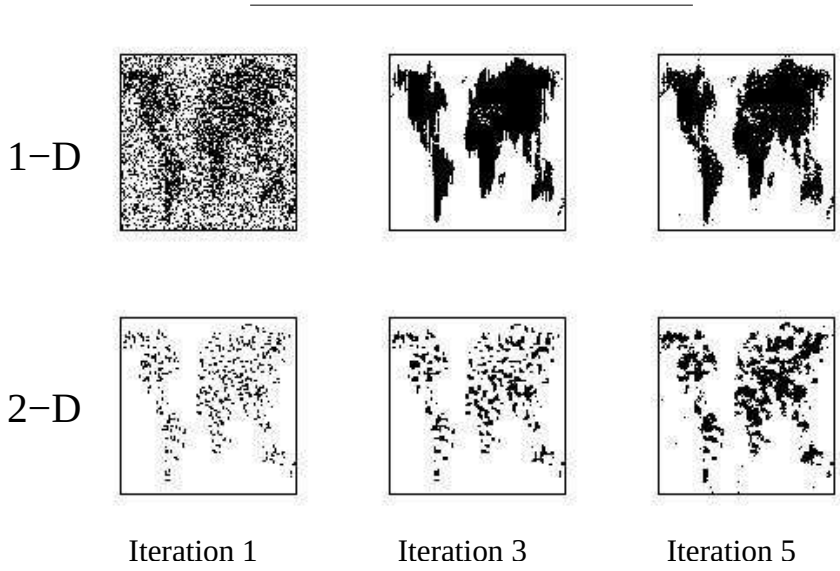


Figure 7.11. A comparison of the intermediate estimates at the decoder. The estimates from the 1-D Markov model are on the top row, while estimates from the 2-D model are on the bottom row. The estimates from the 1-D model exhibit convergence along north-south lines. In contrast, the 2-D model decoder exhibits a localized clumping nature.

7.3.6 Gray Scale Images

We discuss the challenges posed by gray-scale images whose pixel values range from 0 to 255. As with the 2-D model, pixels are assumed to lie on a grid with N_v rows and N_h columns. We use a bit plane decomposition of each pixel, with 8 bits per pixel. The bit planes range from most to least significant. Due to the bitwise focus of this model, we find it convenient to consider each bit plane separately, modeling each with the factor graph of Figure 7.8.

In contrast to the bit plane model, we could use a pixel based model. As an advantage, this would allow us to consider every bit plane. Unfortunately, this would also result in a drastic increase in decoding complexity since the computational complexity of the sum-product algorithm is proportional to the variable alphabet size. Developing the algorithm to operate efficiently with pixel based operations is an open problem.

The 1-D and 2-D models provide significant improvements over the i.i.d. model when considering binary images. These models are very simple in contrast with what is used in unencrypted gray-scale image coding. Unfortunately, this extension of the idealized

models from above for "real world" gray-scale images proves difficult. In practice we find only significant exploitable correlation gains for the two most significant bit planes. Even exploiting inter-bit plane correlations (as discussed in Chapter 8) provides no significant additional gain.

Two of the most common techniques used for compression of unencrypted gray scale images are inapplicable in compressing encrypted images. The first method is the application of transforms, such as the DCT (Discrete Cosine Transform). As can be seen in Figure 7.12 though, bitwise encryption of gray scale images is a non-linear operation. Therefore the transform of the encrypted image lacks the exploitable structure of the transform of the unencrypted image. The second method is the use of localized predictors. Without access to the local unencrypted context available to compressors of unencrypted images though, localized predictors fail. In contrast, due to the high temporal correlation that is present, video offers greater opportunity for compression. We consider the compression of encrypted video in Chapter 8.

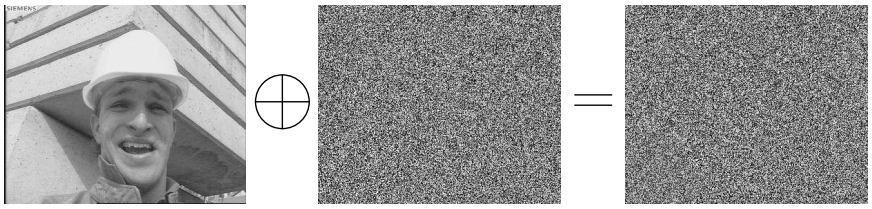


Figure 7.12. A sample image, "Foreman," is shown on the left (811,008 bits total). To encrypt this image, the 811,008 bit random key in the center is added to the unencrypted image (using a bitwise exclusive-OR). Many classical methods exist to compress the "Foreman" image. These methods use techniques such as transform based methods and localized predictors. Unfortunately, due to the nonlinear effects of bitwise encryption, none of the classical methods or techniques will successfully compress the encrypted image.

7.4 Conclusions & Open Problems

In this chapter we presented a practical scheme for compressing encrypted data. We first describe a general framework, and develop this framework for LDPC based codes and stream ciphers. We propose idealized i.i.d., 1-D, and 2-D Markov models for compressing encrypted images, and give an example of their benefit.

The work presented herein is a proof of concept. It suggests a number of areas for further

study. First, the study of other models for data could provide even stronger compression performance. For example, in nature correlation between image pixels is not limited simply to adjacent pixels. Incorporation of a greater number of relationships in the model could improve system performance, though these gains would be balanced by increased model complexity.

Another area of study is in the structure of the "doped" bits. We would like to study a systematic method for selecting which bits to dope. Some questions related to bit doping immediately arise from the work presented in this chapter. The first question is to consider what proportion of bits should be doped. Second, we would like to study structured bits doping schemes. It seems likely that doping based on the code structure could result in better performance. Finally, we would like to incorporate the open problem of code design presented in Section 4.5 into this study.

A final area of future work suggested by this chapter is in the study of the compression of other encrypted "real world" sources. Sources such as text, audio, and video. In the following chapter, we present such a scheme for compressing encrypted video sequences.

Chapter 8

Compression of Encrypted Video

In this chapter, we build on the framework presented in Chapter 7 to develop a scheme for compressing encrypted video. Encryption masks the source, thereby rendering traditional compression algorithms ineffective. However, by conceiving of the problem as one of distributed source coding it is shown in (Johnson et al., 2004) that encrypted data is as compressible as unencrypted data. However, as discussed throughout this thesis, there are a number of challenges to implementing these theoretical results. In this chapter we tackle some of the major impediments.

To achieve compression, it is crucial that our models are well-matched to the underlying source *and* are compatible with our compression techniques. In this chapter, we develop models for video. For video, we compare our performance to a state-of-the-art motion-compensated lossless video encoder that requires unencrypted video as input. It compresses each unencrypted frame of the "Foreman" test video sequence by about 59% on average. In comparison, our proof-of-concept implementation that works on *encrypted* data compresses the same sequence by about 33%. Because the source is masked, the encoder cannot know the target rate *a priori*. We incorporate the rate adaptive protocol of Chapter 6 into our compression of encrypted video solution to address this problem.

8.1 Introduction

Encryption masks digital content so that it appears completely random. This renders traditional compression algorithms ineffective. Best practices therefore dictate that content

must be compressed before it is encrypted. Unfortunately best practices in compression and security cannot always be assumed. There are prominent examples where raw digital content is encrypted without first being compressed. For example, this is the case in the High-Bandwidth Digital Content Protection (HDCP) protocol [86]. This motivates the search for novel compression routines that operate on uncompressed, encrypted data.

As a motivating application, consider the delivery of high-definition video content over home wireless networks. On one hand are content providers who have strict security and privacy requirements. To meet these requirements while maintaining the highest quality video, content providers often encrypt their raw (uncompressed) content prior to distribution. For example, the afore mentioned HDCP standard requires the encryption of raw video content. On the other hand are wireless infrastructure companies. In wireless networks, transport capacity is limited, and the data rates of raw high-definition video will overwhelm many communication links. This limits the potential for such systems. By building a prototype system, this chapter illuminates a solution that can enable the successful development of these systems.

8.1.1 High-Level System Description and Prior Work

To illustrate the system architecture, consider the encryption process depicted in Figure 8.1. In the left-hand plot is a frame from the standard "Foreman" test video sequence. This "plain-text" image is unencrypted. Each of the 288×352 pixels in this frame takes an integer value in the range 0 to 255. The resulting size of each raw frame is $811,008$ bits. The middle plot of the figure shows an image of equal dimension to the first, but consisting of bits selected uniformly at random (an independent and identically distributed (i.i.d.) Bernoulli-0.5 sequence). This is an example of a "stream-cipher." We encrypt the image by applying a bitwise exclusive-OR (XOR) between each key bit and the corresponding plain-text bit. Under this assumption such a "one-time pad" system offers perfect security [77]. The resulting encrypted "cipher-text" is shown in the right-hand plot. Compression of the highly structured plain-text image has been studied for decades. However, as discussed in Chapter 7, none of these techniques can be used to successfully compress the *marginally random* cipher-text. The cipher-text, independent of the source, is a Bernoulli-0.5 i.i.d. image and is therefore not individually compressible.

As discussed in Chapter 7, while the plain-text and cipher-text are independent the cipher-text and stream-cipher are not. This is the crucial insight. Since the stream-cipher

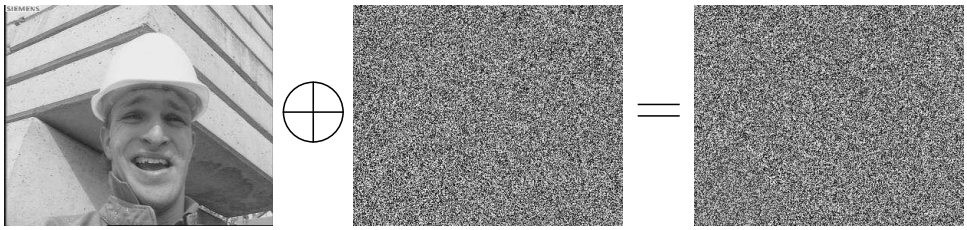


Figure 8.1. A sample frame from the standard "Foreman" video test sequence is shown on the left (811,008 bits total). To encrypt this frame, the 811,008 bit random key in the center is added to the unencrypted image (using a bitwise exclusive-OR). Although many classical methods exist to compress the "Foreman" video sequence, none will successfully compress the encrypted video.

is known to the decoder, compression is possible by leveraging the dependence between cipher-text and stream-cipher. The compression and reconstruction/decryption process can be understood by viewing the cipher-text as a noisy version of the stream-cipher.

We compress the cipher-text by calculating a number of parity constraints of the cipher-text. These are mod-2 sums of various subsets of the cipher-text bits. The resulting sums are the compressed bits, which we refer to as the "syndrome" of the cipher-text (as discussed in Chapter 7). The joint decompresser and decrypter combines the key sequence and the syndrome to find the source sequence closest to the key sequence that also satisfies the parity constraints. This search is identical to the decoding process of an error-correcting code. By choosing the parity-constraints appropriately the cipher-text can be recovered with high probability. Adding the key sequence to the reconstructed cipher-text yields the plain-text, assuming decoding success.

A simplified block-diagram of the system is depicted in Figure 8.2 (a repeat of the block diagram introduced in Figure 7.1). The length n plain-text source x^n is encrypted with the stream-cipher k^n to produce the cipher-text y^n. The cipher-text is compressed into the nR ($R < 1$) bits of the syndrome. The decoder makes its source estimate $\hat{x}^n$ based on the syndrome and the stream-cipher k^n. In standard Slepian-Wolf terminology the key sequence is the "side-information" available to the decoder.

In the previous chapter, we discussed a framework for compressing encrypted sources, ranging from simple source to images. However, we foresee the main applications of these ideas to lie in sources such as video. The HDCP motivating example above underscores

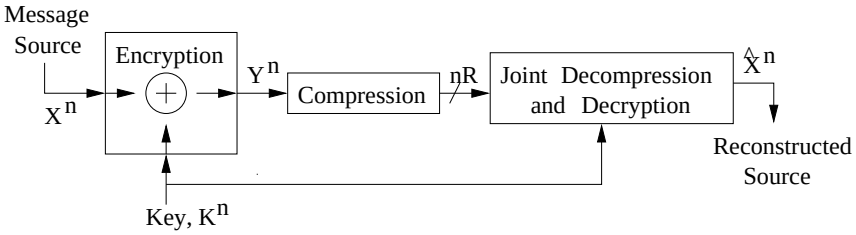


Figure 8.2. The source is first encrypted and *then* compressed. The compressor does not have access to the key used in the encryption step. The decoder jointly decompresses and decrypts. The key serves as the decoder side information.

video as an important focus. Unfortunately, for video sources, the models discussed in Chapter 7 poorly capture the source characteristics.

8.1.2 Main Results & Outline

Although we demonstrate improvements on images in Chapter 7, when applying the lessons learned to video content our techniques really gain traction. In this chapter, we show that in contrast to images, encrypted video is better matched to our techniques and is an application in much greater need of such a solution. Operating on a frame by frame basis, video offers the decoder the opportunity to combine intra-frame spatial statistical models with predictive temporal models. We present a proof-of-concept implementation and evaluate its performance on standard video sequences. Finally, we incorporate the feedback ideas to give a fully operational system for blind compression of encrypted video.

This chapter is organized as follows. Section 8.2 provides a model to incorporate in the framework of Chapter 7 for compressing encrypted video. In Section 8.3 we discuss the decoding algorithm for encrypted video. Section 8.4 provides the results of compressing encrypted video, and compares the results to other leading lossless video codecs. In Section 8.5, we discuss how to incorporate the blind protocol of Chapter 6 and provide compression results. Finally, we provide concluding notes and open problems in Section 8.6.

8.2 Source Model

In this section, we consider a model for video sequences. Encrypted video supports a larger opportunity for compression than still images, due to the availability of temporal correlations in addition to spatial correlations. Presuming frame by frame compression and decompression in our scheme, the decoder has access to temporal (as opposed to just spatial) predictors. In this section, we consider videos composed of sequences of gray scale images. Since we consider bitwise stream ciphers Section 7.2, we develop a bitwise source model for video sources. We then proceed to describe adaptations made to the decoder and finally present practical results (including blind simulations). Note that we describe our model in terms of the joint decoder from the framework presented in Section 7.2.

We base our model for video on a succession of the 2-D Markov model considered in Section 7.3.2, one for each bit plane. We begin by relabeling each bit from Figure 8.3 (reprinting Figure 7.8) as $\hat{x}_{i,j}^t[l]$ to represent the lth most significant bit of the pixel at row i and column j of frame t. A dropped index implies that we are considering the entire range over that index. For example, $\hat{x}^t[l]$ represents the entire lth bit plane of the frame t, while $\hat{x}^t$ represents the entire frame at time t with elements ranging from 0 to 255 (since we include every bit plane).

We begin description of our temporal prediction scheme with a high level enumeration of the steps involved.

1. Generate frame predictors $\tilde{x}^t$ and $\tilde{x}^{t-1}$ based on motion extrapolation from previously decoded frames $\hat{x}^{t-1}$, $\hat{x}^{t-2}$, and $\hat{x}^{t-3}$. Using motion extrapolation function $g(\cdot,\cdot)$, we denote this as follows

$$\tilde{x}^t = g(\hat{x}^{t-1}, \hat{x}^{t-2}), \text{ and } \tilde{x}^{t-1} = g(\hat{x}^{t-2}, \hat{x}^{t-3}).$$

2. Having frame predictors, the next step is to estimate their quality. We generate an estimated distribution, $\tilde{P}(\hat{x}^t, \tilde{x}^t)$, based on the empirical distribution of the prior decoded frame and it's predictor. We denote this as follows

$$\tilde{P}(\hat{x}^t, \tilde{x}^t) \approx P(\hat{x}^{t-1}, \tilde{x}^{t-1}).$$

3. Finally, we condition the current frame bit plane model, $x^t[l]$, using the estimated distribution, $\tilde{P}(\hat{x}^t, \tilde{x}^t)$, the predictor, $\tilde{x}^t$, and the prior decoded bit planes, $\hat{x}^t[1]$ through $\hat{x}^t[l-1]$.

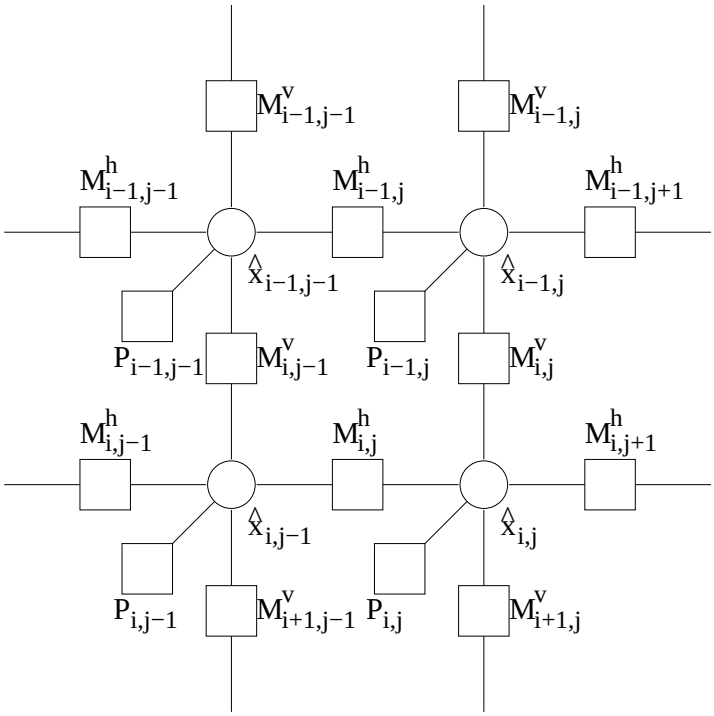


Figure 8.3. A reprint of Figure 7.8. A section of the 2-D source model. This model includes the source bits (circles) and both horizontal correlations, $M^h_{i,j}$, and vertical correlations, $M^v_{i,j}$. We apply this model to each bit plane of each frame of the video sequences.

We now discuss our temporal prediction approach in detail, diagramed in Figure 8.4 and Figure 8.5. Figure 8.4 shows what information is fed to the decoder for the frame at time t. We assume our system works on one frame at a time, and that the previous frames were decoded exactly. This allows our decoder to have a full copy of the exact previous frames available, and thus it is able to exploit temporal correlations. We operate by having a predictor generated of each frame based on the two previous frames. That is, given that the previous two frames, $\hat{x}^{t-2}$ and $\hat{x}^{t-1}$, are decoded successfully, we generate a predictor, $\tilde{x}^t = g(\hat{x}^{t-1}, \hat{x}^{t-2})$, such that this function is represented by the "Predictor" boxes in Figure 8.4.

In this work, we implement a simple predictor operating across the bit planes that uses motion compensation together with motion extrapolation to generate $\tilde{x}^t$, illustrated in Figure 8.5. In our scheme, we use the exact motion relationship between frames $\hat{x}^{t-2}$ and $\hat{x}^{t-1}$ to extrapolate the motion between frames $\hat{x}^{t-1}$ and $\hat{x}^t$. We divide $\hat{x}^{t-1}$ into sub-blocks of $N \times N$ pixels ($N = 8$ in this work), and for each block, we perform motion

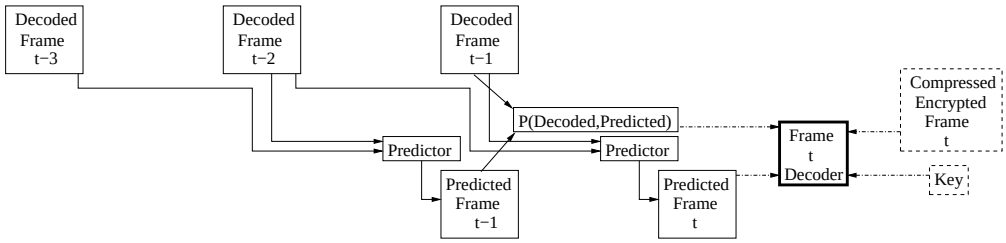


Figure 8.4. A diagram of the way by which information is generated and fed to the decoder for each frame. Here, predicted frames are developed from the two previous frames. A measure of the predictor's accuracy is generated by measuring the accuracy of the previous frame's predictor. This gives the decoder sufficient assistance to recover the frame with the standard joint decompresser and decrypter discussed in Chapter 7.

estimation to find the best matching block in $\hat{x}^{t-2}$. In other words, for the (a, b) block, with $a \in 1, 2, ..., N_v/N$ and $b \in 1, 2, ..., N_h/N$, we find $(v_x(a, b), v_y(a, b))$ such that:

$$(v_x(a,b), v_y(a,b)) = \arg \min_{(v_x,v_y)\in\mathcal{N}} \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} d\left(\hat{x}^{t-1}_{aN+i,bN+j}, \hat{x}^{t-2}_{aN+i+v_y,bN+j+v_x}\right)$$

where $\mathcal{N}$ denotes the allowable search range, and $d(\cdot, \cdot)$ is a metric that measures the difference between two pixel intensities. Here, we simply use the square difference, i.e. $d(\alpha, \beta) = |\alpha - \beta|^2$. Assuming that there is little change in the motion fields over one frame instance, we estimate the motion vector for each block in $\hat{x}^t$ as the motion vector of the co-located block in $\hat{x}^{t-1}$. Therefore, if (v_x, v_y) is the motion vector of the (a, b) block of $\hat{x}^{t-1}$, the (a, b) block in $\tilde{x}^t$ is then given by:

$$\tilde{x}^t_{aN+i,bN+j} = \hat{x}^{t-1}_{aN+i+v_y,bN+j+v_x} \ , \forall \quad 0 \le i, j \le N-1.$$

A more sophisticated predictor could lead to a better $\tilde{x}^t$ and hence a better model performance, but this implementation suffices here.

It is clear that the predicted frame $\tilde{x}^t$ can be useful for understanding the actual frame $\hat{x}^t$. To make use of the predictor though, we must model how accurate of a predictor it is. We do this by considering the empirical distribution $P(\hat{x}^{t-1}, \tilde{x}^{t-1})$ of the previous frames and their predictors, as done in Chapter 6. These distributions are calculated on a pixel level since the distribution for individual bit planes can be determined as a function of the pixel level distribution. That is, we estimate the joint distribution of our current frame and its predictor as $\tilde{P}(\hat{x}^t, \tilde{x}^t) \approx P(\hat{x}^{t-1}, \tilde{x}^{t-1})$. In Figure 8.4, we represent this function with the box labeled "P(Decoded,Predicted)." In practice, when calculating the histogram for

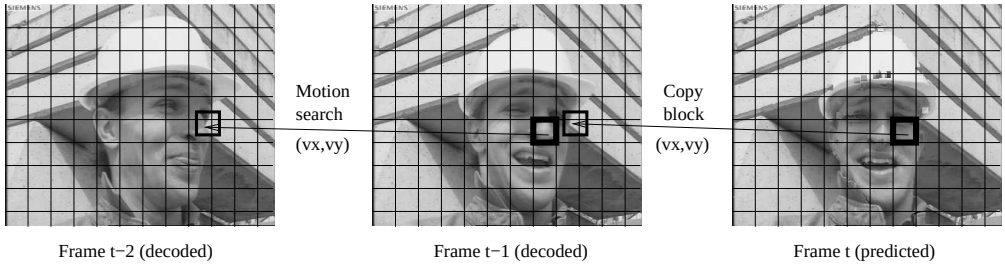


Figure 8.5. Predictor generation using motion compensation and motion extrapolation. Frames $\hat{x}^{t-2}$ and $\hat{x}^{t-1}$ are available after having been previously decoded. For each block in frame $\hat{x}^{t-1}$, motion estimation is performed to find the best matching block in frame $\hat{x}^{t-2}$. In the above example, the best predictor for the block with thickened edges in $\hat{x}^{t-1}$ is the lightly thickened box in $\hat{x}^{t-2}$, shown with its corresponding motion vector (v_x, v_y). We estimate the motion vector for each block in frame $\hat{x}^t$ with that of its co-located block in frame $\hat{x}^{t-1}$. Its predictor is then indicated by the estimated motion vector. Here, the predictor for the block with thickened edges in $\hat{x}^t$ is the lightly thickened box in $\hat{x}^{t-1}$, as pointed to by the motion vector (v_x, v_y).

the previous frame and its predictor, we add it to the histogram (though with its counts divided by 2 to act as a forgetting factor) used for decoding the prior frame. This results in greater stability in the estimate for the distribution between the predicted and actual frames. We omit this from Figure 8.4 for clarity.

We consider one final source of side information for the decoder (as mentioned in Section 7.3.6). Since we process the frame with a single bit plane at a time, we further assume that they are processed in order of most significant to least significant. Consider adjacent pixels $x^t_{i,j} = 127$ and $x^t_{i,j+1} = 128$. If we consider only the bitwise expansion of these two pixels, we see that they differ at every bit plane. Looking at each bit plane in isolation, the strong similarity between these two pixels is missed. Conversely, by considering each bit plane in context of the bit planes of greater significance when using $\tilde{P}(\hat{x}^t, \tilde{x}^t)$, we are able to exploit more relevant correlations between bits.

The overall model consists of a predicted frame $\tilde{x}^t$ based on the two previous frames and an empirical distribution $\tilde{P}(\hat{x}^t, \tilde{x}^t)$ based on the previous frame and its predictor. We use all this information available to alter the parameters (p, h_0, h_1, v_0, v_1). We no longer assume

them to be stationary; instead we make them dependent upon their context as follows

$$
\begin{aligned}
&p(x_{i,j}^t[l]|\hat{x}_{i,j}^t, x_{i,j}^t[1], \ldots, x_{i,j}^t[l-1]),\\
&h_0(x_{i,j}^t[l]|\hat{x}_{i,j}^t, x_{i,j}^t[1], \ldots, x_{i,j}^t[l-1]),\\
&h_1(x_{i,j}^t[l]|\hat{x}_{i,j}^t, x_{i,j}^t[1], \ldots, x_{i,j}^t[l-1]),\\
&v_0(x_{i,j}^t[l]|\hat{x}_{i,j}^t, x_{i,j}^t[1], \ldots, x_{i,j}^t[l-1]),\\
&v_1(x_{i,j}^t[l]|\hat{x}_{i,j}^t, x_{i,j}^t[1], \ldots, x_{i,j}^t[l-1]).
\end{aligned}
$$

Specifically, we calculate each of these parameters using the marginalized empirical distribution $\tilde{P}(\hat{x}^t, \tilde{x}^t)$ conditioned as above.

8.3 Message Passing Rules

When belief propagation is run for each bit plane over the graph model considered here, the operation of the constraints is similar to the operation discussed in Section 7.3.3. Since each source constraint, labeled $P_{i,j}^t[l]$, is a terminal node, it transmits its log likelihood ratio calculated using the marginal probability $p(x_{i,j}^t[l]|\hat{x}_{i,j}^t,\ x_{i,j}^t[1],\ \ldots,\ x_{i,j}^t[l-1])$. For the correlation nodes, labeled $M_{i,j}^{h,t}[l]$ and $M_{i,j}^{v,t}[l]$, message update rules are of the form in Equation (8.1) (reprinting Equation (7.1)) but modified for the correlation parameters above

$$\mu = \log\left(\frac{(2-h_0-h_1)+(h_1-h_0)\tanh(\nu/2)}{(h_0+h_1)+(h_0-h_1)\tanh(\nu/2)}\right). \tag{8.1}$$

The rest of the messages passed throughout the compression of encrypted data framework are identical to those discussed in Chapter 7.

8.4 Experimental Results

We evaluate our technique on the standard "Foreman," "Garden," and "Football" video test sequences[1]. Respectively, these are low-motion, high-motion, and high-motion sequences. We compress 12 encrypted frames (i.e., a group of pictures (GOP) size of 12) of these test sequences. In this group, we only compress the last nine frames (frames 4 through 12). As mentioned above, the first three frames are used to initialize the predictors. With a larger GOP size, the rate effect of the three uncompressed frames diminishes. Further, for ease of implementation, we divide each frame into 9 regions (arranged 3×3)

[1]These sequences are used to establish benchmarks for video coding schemes throughout the video coding community.

as a grid. Each region, having 1/9 of the pixels, has on the order of $\sim 10^4$ pixels (and each bit plane has approximately $\sim 10^4$ bits). Better performance is likely by considering each frame as a whole.

For these experiments, we generate a finite set of LDPC codes from a selection of LDPC degree distributions [70]. We use 38 different degree distributions (each both check and variable irregular), with compression rates ranging from 0.025 to 0.95, with a spacing of approximately 0.025 between codes. Most degree distributions are obtained from the binary symmetric channel design section of the LTHC website [4]. The rest of the degree distributions we use are designed using EXIT (extrinsic information transfer) charts [7], though similar performance is seen by selecting codes from LTHC designed for other channel models (such as the binary additive white Gaussian noise channel). Using these degree distributions, the transform matrices are generated randomly, and then pruned to remove short loops (of length 4 or less). For greater detail, see Appendix A.

As with the image results from Chapter 7, through a sequence of trials we determine the minimum rate we can compress the encrypted sections to and still recover the original source exactly. We aggregate the total number of "transmitted bits" and present the resulting rate here. The results of an automated system are presented in Section 8.5.

As in Section 7.3, we use doping to initiate the decoding process. In contrast to the models of Section 7.3 though, only 5% to 10% of the total output compressed bits are doped bits here. The reduced need for doped bits is due to the strength of the predictors used. Since the prior frames provide significant bias for the decoder, additional doped bits are unnecessary.

Our overall results are presented in Table 8.4. The numbers in this table represent the average compression ratio in terms of the number of output bits per source bit. For example, the number 0.6700 in the upper right most cell of the table indicates that on average, only 0.6700 output bits were necessary to represent each bit of the source video. As a reminder, this number applies only to the last 9 frames in our group of pictures, ignoring the first 3 frames of the video for initialization as in Figure 8.4. As can be seen in this chart, although we cannot perform as well as we would have on unencrypted data, significant compression gains can still be achieved. Further, we can see the effect of the predictor on performance, as performance is far better for the low motion "Foreman" sequence. The "Foreman" sequence has better predictor frames than the other sequences, which have large deviations between predicted and actual frames.

Frames 4 - 12	JPEG-LS	MSU lossless	Encrypted
Foreman	0.4904	0.4113	0.6700
Garden	0.7320	0.5908	0.8236
Football	0.6700	0.5956	0.9283

Table 8.1. Comparison of results for the compression of three encrypted sequences compared to results of traditional encoders compressing the unencrypted sequences. Despite operating on encrypted data, the proposed approach performs well.

For purposes of comparison, we also compress the videos losslessly using publicly available software. The first system uses JPEG-LS [84] to perform pure intra-frame video compression. JPEG-LS not only has low encoding complexity, but also demonstrates exceptional compression performance relative to other lossless image coding standards [72]. In our study, we compress each video frame with the publicly available JPEG-LS coder [37]. The second system, MSU lossless video codec [57], is a publicly available lossless inter-frame video codec, which is claimed by its authors to have the highest lossless video compression performance [58]. Due to its proprietary nature, the details of their video codec are not known. However, its performance does seem to be comparable to past results in the literature that used either fixed spatio-temporal predictor [9] or motion compensation [53].

As an example of the operations performed for a particular frame, consider Figure 8.6. This figure demonstrates everything that goes into the decompression of frame 4 (except the encryption key, omitted for clarity). Here we see that the encoder has access to only the encrypted version of frame 3. In contrast, the decoder is given the predicted frame 4, the probability model estimating the reliability of the predicted frame, and the compressed bits. In this example, the frame is recovered without error.

In Figure 8.7 and Figure 8.8, we present more detailed results from compressing the encrypted "Foreman" sequence. The vertical axis of these plots represent the compression ratio presented earlier (output bits per source bit). In the plot on the left (Figure 8.7), we plot the rate as a function of the frame number. Note that we made no attempt to compress the first three frames. This plot shows that the overall performance varies as the video progresses but each frame is compressed to at least 70% of the source rate. In the plot on the right (Figure 8.8), we present the rate used for each bit plane (across the 9 compressed frames). The horizontal axis ranges from the most significant bit plane (1) at left to the least significant bit plane (8) at right. For reference, we are able to compress the most significant bit plane by 78% on average. Due to variations in the magnitude of the motion, for the most significant bit plane frame 12 is most compressible (81%) while

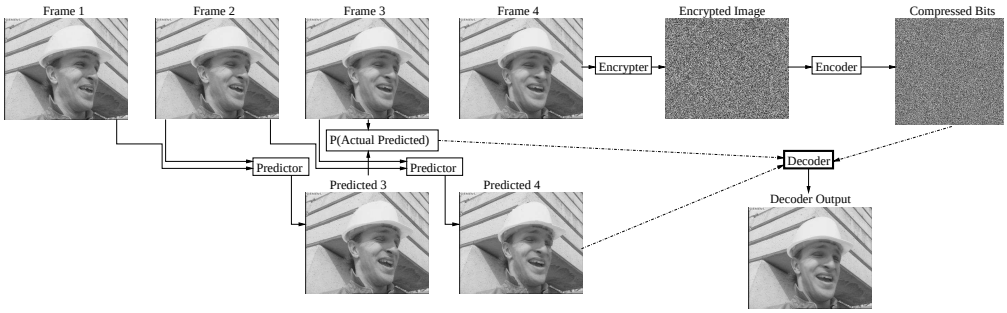


Figure 8.6. A sample compression of the fourth frame of the "Foreman" sequence. This block diagram demonstrates the generation of each of the elements available to the encoder and to the decoder. Here, the decoder is able to reconstruct frame 4 from the predicted frame 4, the probability model, and the compressed bits. The encryption key has been omitted from this figure for clarity.

frame 5 is least compressible (76%). This gives a good indication of how much correlation our system can exploit in each bit plane. As can be seen in this plot, we are able to obtain no compression gains in the two least significant bit planes of this sequence.

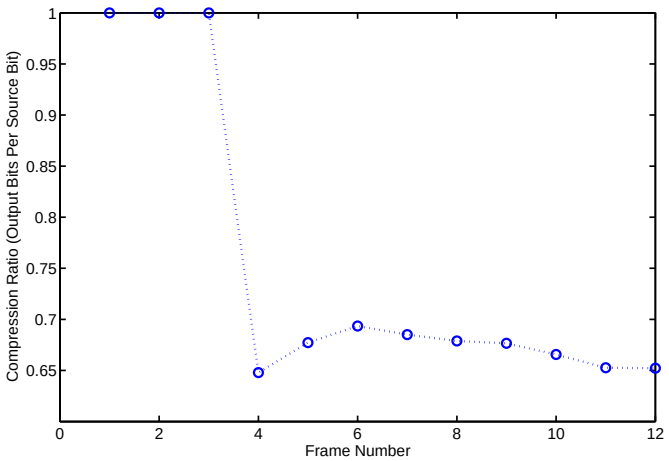


Figure 8.7. Compression results for the 9 frames considered by our system. The average rate (in source bits per output bit) used on each frame. This plot demonstrates consistent rate savings from frame to frame.

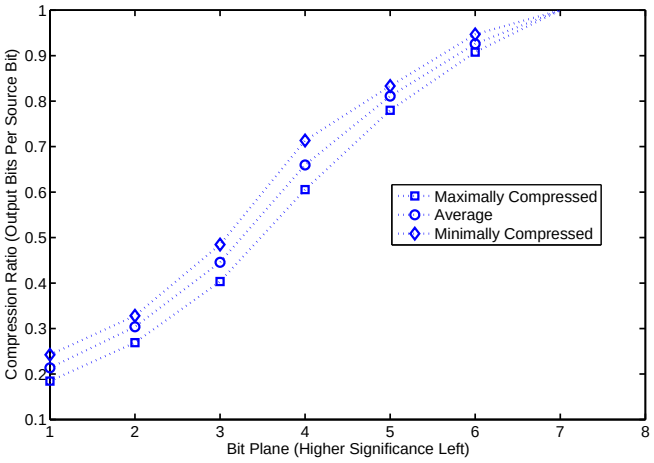


Figure 8.8. Compression results for the 9 frames considered by our system. This plot shows the rate used on each bit plane, ranging from most significant (left) to least significant (right). Maximum and minimum are taken across all 9 frames. This demonstrates the consistent compressibility of each bit plane from frame to frame. It further demonstrates that successively less significant bit planes are successively less compressible.

8.5 Blind Video Compression

In this section we extend the algorithm of Chapter 6 to compressing encrypted video. In particular, the decoder feeds rate requests (as in Section 6.4) for each portion of each bit plane of each frame. When a decoding error is detected, the decoder requests the data be retransmitted uncompressed. As mentioned in Section 6.2, performance can be improved with a hybrid-ARQ-type retransmission strategy.

For clarity, the following discussion presumes each bit plane of each frame is compressed whole, even though in Section 8.4 we break each bit plane into sections. We label the rate used for lth most significant bit plane of frame t as $R^t[l]$. We label the minimum value of $R^t[l]$ for which decoding is successful as $R^t[l]^\star$.

In Section 8.4, we assume that the system knows the values of $R^t[l]^\star$ for each frame and plane. In blind operation, while $R^t[l]^\star$ cannot be achieved, we note that $R^t[l]^\star$ can be determined by the decoder after successful transmission is complete. That is, after successfully decoding a portion of the video sequence, the decoder can then, through trial and error, determine the value of $R^t[l]^\star$. In practice, we found that the decoder could determine this number with fewer than 20 encoding trials.

Frames 4 - 12	Foreman	Garden	Football
Encrypted	0.7918	0.9039	0.9917

Table 8.2. Results for the blind compression of the three encrypted sequences. The proposed approach achieves significant compression gains. This demonstrates the value of the approach presented here.

Our goal then when compressing plane $x^t[l]$ is to determine a rate, $R^t[l]$, based on the best rate for the previous bit plane, $R^t[l-1]^\star$, and on the best rate for the previous frame, $R^{t-1}[l]^\star$. Our goal is to select $R^t[l]$ such that it balances the rate used in transmission against the probability of a decoding error and a retransmission, as in Section 6.3. Fortunately, there is significant predictability between the value $R^t[l]^\star$ and the values $R^t[l-1]^\star$ and $R^{t-1}[l]^\star$ as can be seen in Figure 8.8.

In practice we used a function of the form $\phi_1 R^{t-1}[l]^\star + \phi_2 R^t[l-1]^\star + \phi_3$ to select $x^t[l]$. We generate this function by using the "Foreman" sequence as training data and then apply this function to all three sequences. We select $\phi_1 = 0.8350$, $\phi_2 = 0.2518$, and $\phi_3 = 0.0426$. The blind results are given in Table 8.5. As can be seen here, significant compression gains are still achievable.

8.6 Conclusions & Open Problems

In this chapter we have presented a practical scheme for compressing encrypted video sequences. This chapter describes how to adapt the models of Chapter 7 to gray scale video, and demonstrate the practical compression performance on the standard "Foreman," "Garden," and "Football" sequences. Finally, we present algorithms for blindly compressing encrypted video sequences, and demonstrate its performance.

The work presented herein is more a proof of concept. It suggests a number of areas for further study. As a first area of study, we would like to extend the techniques developed here to other "real world" sources, such as text and audio. Next, we would like to develop techniques to compress the initial frames in order to reduce their influence on the overall results.

Another important area of further study is in the influence of the quality of predictors on compression performance. Prediction resulting from the motion extrapolation offers significant potential performance improvement. A better motion extrapolation technique

will result in better prediction frames and hence better performance. This is demonstrated by our algorithms relative performance on the high motion sequences versus the low motion sequences. Along the lines of prediction, a technique to better evaluate the relationship of the predictor and the actual frames would also improve performance. A detailed study could better bound the error in the predicted frames compared to the actual frames. As a final issue of prediction, a further study of how to perform rate requests could result in vastly improved blind video compression.

Though we consider lossless video coding here, our inability to compress the least significant bit planes suggests a way to apply our scheme to lossy video coding as a final area of future study. Namely, by simply dropping the less significant bit planes, we will improve system performance while providing good signal quality. Since these bit planes play little significance in the prediction process, their loss should not strongly inhibit algorithm performance.

Chapter 9

Conclusions & Future Directions

In this thesis, we have described algorithms and protocols for practical distributed source coding. We have presented a solution flexible enough to be applied to arbitrarily structured distributed source coding problems. Throughout, we have demonstrated the incorporation of Low-Density Parity-Check (LDPC) codes, and how their properties can be leveraged for strong practical performance.

In the first half of this dissertation, we presented code construction and theoretical analysis for the generalized distributed source coding problem. The developments in this half of this dissertation can be classified as follows:

- A practical construction based on linear block codes for the source coding with side information problem was presented in Chapter 3. We selected LDPC codes to incorporate into this construction, and demonstrated the strength of these source codes.
- An adaptation of the construction in Chapter 3 to distributed source codes for parallel sources. The analysis presented in Chapter 4 demonstrates the value of attempting to leverage the non-stationary nature of parallel sources into a single coding framework. We further identified the resulting LDPC code design problem, and presented a powerful solution.
- A further adaptation of the construction in Chapter 3 for the arbitrarily structured distributed source coding problem is presented in Chapter 5. This linear block code based construction is flexible enough to perform over arbitrary correlation structures, for arbitrary rate points, and for an arbitrary number of sources. This construction is

also flexible enough to naturally degrade to the solution of Chapter 3. We demonstrate the application of LDPC codes to the block code construction of Chapter 5, and show that they approach the Slepian-Wolf limit.

- A protocol for blind distributed source coding to fully automate the codes presented in this portion of this thesis is introduced in Chapter 6. Such a protocol is necessary to eliminate the *a priori* assumptions made by most distributed source coding solutions. We present a theoretical analyze of the protocol and show that it quickly approaches the scenario with *a priori* knowledge of the source statistics. We further present results demonstrating the success of this protocol in practice.

In this second half of this dissertation, we have focused on the application of our distributed source coding solution to the problem of compressing encrypted data. The developments in this half of this dissertation can be classified as follows:

- A practical framework for compressing encrypted data in Chapter 7. We demonstrate the flexibility of this framework to compress a source under various source models. We conclude by demonstrating the application of this framework to the compression of encrypted binary images.

- A method to compress encrypted video, extending the compression of encrypted data framework of Chapter 7, is applied to gray scale video sequences in Chapter 8. We demonstrate the significant compression gains achievable when compressing encrypted video without access to the encryption key. Finally, we incorporate the blind transmission protocol of Chapter 6 and demonstrate the results of a completely automatic blind compression of encrypted video system.

The work in this thesis suggests several avenues of further study. We itemize some of these directions here.

- The study of additional code design techniques as the apply to the parallel source problem considered in Chapter 4, the arbitrary distributed source coding problem considered in Chapter 5, and the source models for compressing encrypted data in Chapters 7, and 8 is a significant open problem. In particular, study of the application of density evolution [70] techniques to the code design problem offers significant potential gains.

- The study of structured doping is an interesting open problem. Though unstructured doping results in strong performance, as in Chapter 7, it seems probable that structured doping could perform better still. Doping based on either the structure of the previously decoded data or based on the structure of the compression code used promises strong performance improvements. The density evolution tool of Richardson & Urbanke [70] potentially can provide answers to several of these questions. Incorporating this study into the study of code design offers significant promise as a field of study.

- Application of our distributed source coding solution of Chapter 5 to larger alphabets and to a larger number of sources is another area of further study. Though our construction naturally adapts to these scenarios, it does so at a cost of increased decoder complexity. A full characterization of the system performance in these scenarios is an important next step.

- Since Chapter 6 presented only an upper bound for our protocols performance, we would also like to analytically develop a lower bound. The development of a lower bound on protocol performance would help to establish the optimality of the protocol presented.

- In addition, the incorporation of other types of linear codes or other types of decoders into the blind protocol of Chapter 6 is a promising open problem. Rate adaptable codes and codes offering decoding guarantees promises to greatly improve convergence performance. Further, the LDPC Decoder presented in Feldman, Wainwright, & Karger [24] could improve system performance through decoding quality guarantees.

- The study of improved source models for images and video is another area of further study. Though the models presented in Chapters 7 and 8 result in strong performance, it is clear that better modeling of the data would result in better performance.

- The application of the techniques for practical distributed source coding and practical compression of encrypted data to other "real world" sources is another field of further interest. Significant gains have been achieved through the study of video sequences. Similar gains are likely achievable through the study of text and audio sources.

- Throughout this dissertation, the focus has been on lossless data compression. Yet each problem considered here could also be considered in terms of lossy data compression. Extension of these techniques to lossy data compression is a significant open problem.

To conclude, we hope that this work will spur greater research into applied distributed source codes. More importantly, we hope that it will spur greater use of distributed source codes in a variety of applications.

Bibliography

[1] *Digital Compression and Coding of Continuous-tone Still Images: Requirements and Guidelines*, 1st ed. ISO/IEC JTC 1/SC 29 10918-1, 1994.

[2] A. Aaron and B. Girod, "Compression with side information using turbo codes," in *Proc. Data Compression Conf.*, Snowbird, UT, Mar. 2002, pp. 252–261.

[3] N. Alon and A. Orlitsky, "Source coding and graph entropies," *IEEE Trans. IT*, vol. 42, no. 5, pp. 1329–1339, Sep. 1996.

[4] A. Amraoui and R. Urbanke, "Ldpcopt," World Wide Web, http://lthcwww.epfl.ch/research/ldpcopt/bschelp.php, 2003.

[5] T. Ancheta, "Syndrome source-coding and its universal generalization," *IEEE Trans. IT*, vol. 22, pp. 432–436, Jul. 1976.

[6] ——, "Bounds and techniques for linear source coding (Ph.D. Thesis abstract)," *IEEE Trans. IT*, vol. 24, p. 276, Mar. 1978.

[7] A. Ashikhmin, G. Kramer, and S. ten Brink, "Extrinsic information transfer functions: model and erasure channel properties," in *IEEE Trans. IT*, vol. 50, no. 11, Nov. 2004, pp. 2657–2673.

[8] R. B. Blizard, "Convolutional coding for data compression," in *Martin Marietta Corp., Denver Div.*, vol. Report R-69-17, 1969.

[9] D. Brunello, G. Calvagno, G. Mian, and R. Rinaldo, "Lossless compression of video using temporal information," *IEEE Trans. Image Processing*, vol. 12, no. 2, pp. 132–139, 2003.

[10] G. Caire, S. Shamai, and S. Verdú, "Lossless data compression with error correcting codes," in *Proc. Int. Symp. Inform. Theory*, Yokohama, Japan, Jul. 2003.

[11] ——, "Lossless data compression with low-density parity-check codes," in *Multiantenna Channels: Capacity, Coding, and Signal Processing*. Providence, RI: DIMACS, American Mathematical Society, 2003.

[12] ——, "A new data compression algorithm for sources with memory based on error correcting codes," in *Proc. Inform. Theory Workshop*, Paris, France, Apr. 2003, pp. 291–295.

[13] ——, "Universal data compression with LDPC codes," in *Proc. Intl. Symp. Turbo Codes and Related Topics*, Brest, France, Sep. 2003.

[14] J. Chou, S. S. Pradhan, and K. Ramchandran, "Turbo and trellis-based constructions for source coding with side information," in *Proc. Data Compression Conf.*, Snowbird, UT, Mar. 2003.

[15] S. Y. Chung, T. J. Richardson, and R. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a gaussian approximation," *IEEE Trans. IT*, vol. 47, pp. 657–670, Feb. 2001.

[16] T. P. Coleman, A. H. Lee, M. Médard, and M. Effros, "On some new approaches to practical Slepian-Wolf compression inspired by channel coding," in *Proc. Data Compression Conf.*, Mar. 2004.

[17] G. Cote, B. Erol, M. Gallant, and F. Kossentini, "H.263+: Video Coding at Low Bit Rates," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 7, pp. 849–66, November 1998.

[18] T. Cover, "A proof of the data compression theorem of Slepian and Wolf for ergodic sources," in *IEEE Trans. IT*, vol. 22, Mar. 1975, pp. 226–228.

[19] T. M. Cover and J. A. Thomas, *Elements of Information Theory.* New York: John Wiley and Sons, 1991.

[20] I. Csiszár, "Linear codes for sources and source networks: Error exponents, universal coding," *IEEE Trans. IT*, vol. 28, no. 4, pp. 585–592, Jul. 1982.

[21] I. Csiszár and J. Körner, "Towards a general theory of source networks," *IEEE Trans. IT*, vol. 26, pp. 155–165, Mar. 1980.

[22] ——, *Information Theory: Coding Theorems for Discrete Memoryless Systems.* Academic Press, New York, 1981.

[23] S. C. Draper, "Universal incremental Slepian-Wolf coding," in *42nd Annual Allerton Conf.*, Oct. 2004.

[24] J. Feldman, M. J. Wainwright, and D. R. Karger, "Using linear programming to decode linear codes," in *IEEE Trans. IT*, vol. 51, no. 3, Mar. 2005, pp. 954–972.

[25] K. C. Fung, S. Tavares, and J. M. Stein, "A comparison of data compression schemes using block codes," *Proc. IEEE Int. Electrical and Electronics Conf.*, pp. 60–61, Oct. 1973.

[26] R. G. Gallager, "Low density parity check codes," Ph.D. dissertation, MIT, Cambridge, MA, 1963.

[27] ——, *Information Theory and Reliable Communication.* New York: John Wiley and Sons, 1968.

[28] ——, "Source coding with side information and universal coding," Mass. Instit. Tech., Tech. Rep. LIDS-P-937, 1976.

[29] A. E. Gamal and T. M. Cover, "Achievable rates for multiple descriptions," *IEEE Trans. IT*, vol. 28, pp. 851–857, Nov. 1982.

[30] J. García-Frías, "Joint source-channel decoding of correlated sources over noisy channels," in *Proc. Data Compression Conf.*, Snowbird, UT, Mar. 2001, pp. 283–292.

[31] J. García-Frías and Y. Zhao, "Compression of correlated binary sources using turbo codes," *IEEE Comm. Letters*, vol. 5, pp. 417–419, Oct. 2001.

[32] ——, "Data compression of unknown single and correlated binary sources using punctured turbo codes," in *39th Annual Allerton Conf.*, Allerton, IL, Oct. 2001.

[33] ——, "Compression of binary memoryless sources using punctured turbo codes," in *IEEE Comm. Letters*, vol. 6, no. 9, Sep. 2002, pp. 394–396.

[34] J. García-Frías and W. Zhong, "LDPC codes for compression of multiterminal sources with hidden Markov correlation," in *IEEE Comm. Letters*, Mar. 2003, pp. 115–117.

[35] N. Gehrig and P. L. Dragotti, "Symmetric and asymmetric Slepian-Wolf codes with systematic and non-systematic linear codes," in *IEEE Comm. Letters*, vol. 9, no. 1, Jan. 2005, pp. 61–63.

[36] M. E. Hellman, "Convolutional source encoding," *IEEE Trans. IT*, vol. 21, no. 6, pp. 651–656, Nov. 1975.

[37] HP Labs, "HP Labs LOCO-I/JPEG-LS," http://www.hpl.hp.com/loco/.

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

[39] V. N. Koshelev, "Direct sequential encoding and decoding for discrete sources," *IEEE Trans. IT*, vol. 19, no. 3, pp. 340–343, May 1973.

[40] P. Koulgi, E. Tuncel, S. L. Ragunathan, and K. Rose, "On zero-error source coding with decoder side information," *IEEE Trans. IT*, vol. 49, pp. 99–111, Jan. 2003.

[41] P. Koulgi, E. Tuncel, S. Regunathan, and K. Rose, "On zero-error coding of correlated sources," in *IEEE Trans. IT*, vol. 49, Nov. 2003, pp. 2856–2873.

[42] F. Kschischang, B. Frey, and H. Loeliger, "Factor graphs and the sum-product algorithm," in *IEEE Trans. IT*, vol. 47, no. 2, Feb. 2001, pp. 498–519.

[43] C. Lan, A. D. Liveris, K. Narayanan, Z. Xiong, and C. N. Georghiades, "Slepian-Wolf coding of three binary sources using LDPC codes," in *Proc. Data Compression Conf.*, Snowbird, UT, Mar. 2004.

[44] J. Li, Z. Tu, and R. S. Blum, "Slepian-Wolf coding for nonuniform sources using turbo codes," in *Proc. Data Compression Conf.*, Mar. 2004.

[45] A. D. Liveris, C. Lan, K. Narayanan, Z. Xiong, and C. N. Georghiades, "Slepian-Wolf coding of three binary sources using LDPC codes," in *Proc. Intl. Symp. Turbo Codes and Related Topics*, Brest, France, Sep. 2003.

[46] A. D. Liveris, Z. Xiong, and C. N. Georghiades, "Compression of binary sources with side information at the decoder using LDPC codes," in *Proc. IEEE Global Comm. Symposium*, Taipei, Taiwan, Nov. 2002.

[47] ——, "Compression of binary sources with side information at the decoder using low-density parity-check codes," *IEEE Comm. Letters*, vol. 6, pp. 440–442, 2002.

[48] ——, "A distributed source coding technique for highly correlated images using turbo-codes," in *Proc. Int. Conf. Acoust. Speech, Signal Processing*, May 2002, pp. 3261–3264.

[49] ——, "Joint source-channel coding of binary sources with side information at the decoder using IRA codes," in *Proc. Multimedia Signal Processing Workshop*, St. Thomas, US Virgin Islands, Dec. 2002.

[50] ——, "Distributed compression of binary sources using conventional parallel and serial concatenated convolutional codes," in *Proc. Data Compression Conf.*, Snowbird, UT, Mar. 2003.

[51] M. Luby, "LT codes," in *IEEE Symposium on Foundations of Computer Science*, citeseer.nj.nec.com/luby02lt.html, 2002.

[52] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, pp. 457–458, Oct. 1996.

[53] I. Matsuda, T. Shiodera, and S. Itoh, "Lossless video coding using variable block-size MC and 3D prediction optimized for each frame," *European Signal Processing Conf.*, pp. 1967–1970, 2004.

[54] P. Mitran and J. Bajcsy, "Coding for the Wyner-Ziv problem with turbo-like codes," in *Proc. Int. Symp. Inform. Theory*, Lausanne, Switzerland, Jun. 2002, p. 91.

[55] ——, "Near shannon-limit coding for the Slepian-Wolf problem," in *21st Biennial Symposium on Communications*, Jun. 2002.

[56] ——, "Turbo source coding: A noise-robust approach to data compression," in *Proc. Data Compression Conf.*, Apr. 2002, p. 465.

[57] MSU Graphics & Media Lab Video Group, "MSU lossless video codec," http://www.compression.ru/video/ls-codec/index_en.html.

[58] ——, "MSU lossless video codecs comparison," http://www.compression.ru/video/codec_comparison/lossless_codecs_en.html.

[59] H. Ohnsorge, "Data compression system for the transmission of digitized signals," *Proc. Int. Conf. Comm.*, vol. II, pp. 485–488, Jun. 1973.

[60] A. Orlitsky, "Interactive communication of balanced distributions and correlated files," *SIAM Journal on Discrete Mathematics*, vol. 6, pp. 548–564, 1993.

[61] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 1988.

[62] S. S. Pradhan, "On Rate-Distortion of Gaussian Sources with memory in the Presence of Side Iinformation at the Decoder," *Project Report, ECE 480, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign*, December 1998.

[63] S. S. Pradhan, R. Puri, and K. Ramchandran, "n-channel symmetric multiple descriptions - part I: (n,k) source-channel erasure codes," *IEEE Trans. IT*, vol. 50, no. 1, pp. 47 – 61, Jan. 2004.

[64] S. S. Pradhan and K. Ramchandran, "Distributed source coding: Symmetric rates and applications to sensor networks," in *Proc. Data Compression Conf.*, Mar. 2000.

[65] ——, "Enhancing analog image transmission systems using digital side information: A new wavelet based image coding paradigm," in *Proc. Data Compression Conf.*, Mar. 2001.

[66] ——, "Distributed source coding using syndromes (DISCUS): design and construction," *IEEE Trans. IT*, vol. 49, no. 3, pp. 626–643, Mar. 2003.

[67] R. Puri, S. Pradhan, and K. Ramchandran, "n-channel symmetric multiple descriptions - part II: An achievable rate-distortion region," *IEEE Trans. IT*, vol. 51, no. 4, pp. 1377 – 1392, Apr. 2005.

[68] R. Puri and K. Ramchandran, "PRISM: A New Robust Video Coding Architecture Based on Distributed Compression Principles," *40th Allerton Conference on Communication, Control and Computing*, October 2002.

[69] T. J. Richardson, M. A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. IT*, vol. 47, pp. 619–637, Feb. 2001.

[70] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. IT*, vol. 47, pp. 599–618, Feb. 2001.

[71] B. Rimoldi and R. Urbanke, "Asynchronous Slepian-Wolf coding via source-splitting," *Proc. Int. Symp. Inform. Theory*, p. 271, Jul. 1997.

[72] D. Santa-Cruz, T. Ebrahimi, J. Askelof, M. Larsson, and C. Christopoulos, "JPEG 2000 still image coding versus other standards," *PROC SPIE INT SOC OPT ENG*, vol. 4115, pp. 446–454, 2000.

[73] M. Sartipi and F. Fekri, "Distributed source coding in wireless sensor networks using LDPC codes: A non-uniform framework," in *Proc. Data Compression Conf.*, Mar. 2005, p. 477.

[74] ——, "Distributed source coding in wireless sensor networks using LDPC codes: The entire Slepian-Wolf rate region," in *Wireless Comm. And Networking Conf.*, vol. 4, Mar. 2005, pp. 1939–1944.

[75] D. Schonberg, S. S. Pradhan, and K. Ramchandran, "Distributed code constructions for the entire Slepian-Wolf rate region for arbitrarily correlated sources," in *Proc. Data Compression Conf.*, Mar. 2004, pp. 292–301.

[76] D. Schonberg, K. Ramchandran, and S. S. Pradhan, "LDPC codes can approach the Slepian Wolf bound for general binary sources," in *40th Annual Allerton Conf.*, Oct. 2002, pp. 576–585.

[77] C. Shannon, "Communication theory of secrecy systems," in *Bell System Technical Journal*, vol. 28, Oct. 1949, pp. 656–715.

[78] D. Slepian and J. K. Wolf, "Noiseless coding of correlated information sources," *IEEE Trans. IT*, vol. 19, pp. 471–480, Jul. 1973.

[79] V. Stanković, A. D. Liveris, Z. Xiong, and C. N. Georghiades, "Design of Slepian-Wolf codes by channel code partitioning," in *Proc. Data Compression Conf.*, Mar. 2004.

[80] T. Uyematsu, "An algebraic construction of codes for Slepian-Wolf source networks," *IEEE Trans. IT*, vol. 47, no. 7, pp. 3082–3088, Nov. 2001.

[81] V. A. Vaishampayan, "Design of multiple description scalar quantizers," *IEEE Trans. IT*, vol. 3, pp. 821–834, May 1993.

[82] D. Varodayan, A. Aaron, and B. Girod, "Exploiting spatial correlation in pixel-domain distributed image compression," in *Proc. Picture Coding Symp.*, Beijing, China, April 2006.

[83] X. Wang and M. Orchard, "Design of trellis codes for source coding with side information at the decoder," in *Proc. Data Compression Conf.*, Snowbird, UT, Mar. 2001, pp. 361–370.

[84] M. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles andstandardization into JPEG-LS," *IEEE Trans. Image Processing*, vol. 9, no. 8, pp. 1309–1324, 2000.

[85] T. Wiegand, G. J. Sullivan, G. Bjntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003.

[86] Wikipedia, "High-Bandwidth Digital Content Protection," http://en.wikipedia.org/wiki/High-Bandwidth_Digital_Content_Protection, October 2006.

[87] F. M. J. Willems, "Totally asynchronous Slepian-Wolf data compression," *IEEE Trans. IT*, vol. IT-34, pp. 35–44, Jan. 1988.

[88] H. S. Witsenhausen and A. D. Wyner, "Interframe coder for video signals," U.S. Patent 4,191,970, May, 1978.

[89] A. Wyner and J. Ziv, "The rate-distortion function for source coding with side information at the decoder," *IEEE Trans. IT*, vol. 22, no. 1, pp. 1–10, Jan. 1976.

[90] A. D. Wyner, "Recent results in the Shannon theory," *IEEE Trans. IT*, pp. 2–9, Jan. 1974.

[91] Z. Xiong, A. D. Liveris, and S. Cheng, "Distributed source coding for sensor networks," *IEEE Signal Proc. Mag.*, vol. 21, pp. 80–94, Sep. 2004.

[92] R. Zamir, S. Shamai, and U. Erez, "Nested linear/lattice codes for structured multiterminal binning," *IEEE Trans. IT*, vol. 48, pp. 1250–1276, Jun. 2002.

[93] Q. Zhao and M. Effros, "Lossless and near-lossless source coding for multiple access networks," *IEEE Trans. IT*, vol. 49, pp. 112–128, Jan. 2003.

[94] Y. Zhao and J. García-Frías, "Data compression of correlated non-binary sources using punctured turbo codes," in *Proc. Data Compression Conf.*, Snowbird, UT, Apr. 2002, pp. 242–251.

[95] ——, "Joint estimation and data compression of correlated non-binary sources using punctured turbo codes," *Proc. Conf. on Information Sciences and Systems*, Mar. 2002.

[96] W. Zhong, H. Lou, and J. García-Frías, "LDGM codes for joint source-channel coding of correlated sources," in *Proc. Int. Conf. Image Processing*, Barcelona, Spain, Sep. 2003.

Appendix A

Codes and Degree Distributions Used to Generate LDPC Codes

In this section we list the degree distributions we used in our simulations. These degree distributions were either obtained from the LTHC [4] online database or were generated using the EXIT charts [7] based design technique. For a further description of LDPC code design techniques, and explanation of the terminology used in this section, see Section 2.3.2.

For convenience, the compression rates of the codes, to 4 significant digits, are as follows; 0.0300, 0.0500, 0.0800, 0.1000, 0.1300, 0.1500, 0.1800, 0.2000, 0.2281, 0.2500, 0.2780, 0.3000, 0.3277, 0.3500, 0.3765, 0.4010, 0.4257, 0.4508, 0.4754, 0.5000, 0.5261, 0.5515, 0.5765, 0.6006, 0.6257, 0.6500, 0.6800, 0.7000, 0.7300, 0.7500, 0.7800, 0.8000, 0.8200, 0.8500, 0.8800, 0.9000, 0.9300, 0.9500. Note that the source coding rate, R_s, and the channel coding rate, R_c, are related as $R_s = 1 - R_c$. Unless otherwise specified, rates are given as source coding rates throughout this thesis.

We give the detailed degree distributions for each of the codes below.

1. Code compression rate $R = 0.0300$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.0976872000x + 0.2084250000x^2 + .0239832000x^4 \\ +0.0025877000x^5 + 0.0030756200x^6 + 0.2216020000x^7 \\ +0.0791919000x^{15} + 0.0178188000x^{17} + 0.0707131000x^{21} \\ +0.0032752600x^{30} + 0.0539584000x^{39} + 0.0959645000x^{40} \\ +0.0373115000x^{49} + 0.0555949000x^{64} + 0.0288105000x^{66} \end{array} \right.$$

$$\rho(x) = x^{199}$$

2. Code compression rate $R = 0.0500$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.1130550000x + 0.2223260000x^2 + 0.0217564000x^5 \\ +0.1437610000x^6 + 0.0077757700x^7 + 0.0978175000x^8 \\ +0.0282852000x^{13} + 0.0669393000x^{15} + 0.0730412000x^{26} \\ +0.0149455000x^{34} + 0.2102970000x^{35} \end{array} \right.$$

$$\rho(x) = x^{109}$$

3. Code compression rate $R = 0.0800$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.1172100000x + 0.2107930000x^2 + 0.1521100000x^6 \\ +0.0845317000x^7 + 0.0236898000x^8 + 0.0049341200x^{15} \\ +0.0493028000x^{16} + 0.1133230000x^{19} + 0.0147003000x^{23} \\ +0.1190500000x^{40} + 0.0175400000x^{42} + 0.0928161000x^{46} \end{array} \right.$$

$$\rho(x) = x^{69}$$

4. Code compression rate $R = 0.1000$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.1173140000x + 0.1997290000x^2 + 0.1935750000x^6 \\ +0.0179234000x^7 + 0.0322564000x^8 + 0.0133904000x^{16} \\ +0.1638470000x^{18} + 0.0222610000x^{20} + 0.1509760000x^{48} \\ +0.0445984000x^{50} + 0.0131310000x^{51} + 0.0168516000x^{56} \\ +0.0141470000x^{58} \end{array} \right.$$

$$\rho(x) = 0.5x^{56} + 0.5x^{57}$$

5. Code compression rate $R = 0.1300$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.1154540000x + 0.1846430000x^2 + 0.1872730000x^6 \\ +0.0107396000x^7 + 0.0107802000x^8 + 0.0298498000x^9 \\ +0.0676952000x^{17} + 0.0713005000x^{21} + 0.0311166000x^{22} \\ +0.0523218000x^{25} + 0.1940290000x^{65} + 0.0148834000x^{69} \\ +0.0192438000x^{72} + 0.0020586100x^{89} + 0.0086120200x^{99} \end{array} \right.$$

$$\rho(x) = 0.5x^{45} + 0.5x^{46}$$

6. Code compression rate $R = 0.1500$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.0903419000x + 0.1760760000x^2 + 0.3044350000x^6 \\ +0.1356970000x^{15} + 0.0127703000x^{16} + 0.0764734000x^{22} \\ +0.1680910000x^{27} + 0.0361156000x^{49} \end{array} \right.$$

$$\rho(x) = x^{39}$$

7. Code compression rate $R = 0.1800$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.1267310000x + 0.1851360000x^2 + 0.1895540000x^6 \\ +0.0406345000x^7 + 0.0170619000x^{15} + 0.0511614000x^{18} \\ +0.0888360000x^{19} + 0.0299488000x^{20} + 0.0217271000x^{21} \\ +0.0074947900x^{32} + 0.0092171200x^{39} + 0.0078997400x^{53} \\ +0.0871835000x^{60} + 0.0412600000x^{62} + 0.0890379000x^{63} \\ +0.0071160400x^{68} \end{array} \right.$$

$$\rho(x) = 0.7x^{31} + 0.3x^{32}$$

8. Code compression rate $R = 0.2000$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.0815474000x + 0.1982150000x^2 + 0.7202380000x^{19} \end{array} \right.$$

$$\rho(x) = x^{34}$$

9. Code compression rate $R = 0.2281$. Obtained from EXIT chart based design.

$$\lambda(x) = \left\{ \begin{array}{c} 0.0000696838x + 0.5645527721x^2 + 0.0006797545x^3 \\ +0.0298558824x^4 + 0.0010561819x^5 + 0.0162886198x^6 \\ +0.2852260980x^7 + 0.0945686623x^8 + 0.0003948341x^9 \\ +0.0017553939x^{10} + 0.0010631110x^{11} + 0.0006613925x^{12} \\ +0.0004469575x^{13} + 0.0003264797x^{14} + 0.0002535001x^{15} \\ +0.0002065416x^{16} + 0.0001745563x^{17} + 0.0001515564x^{18} \\ +0.0001343969x^{19} + 0.0001209905x^{20} + 0.0001102764x^{21} \\ +0.0001014784x^{22} + 0.0000940540x^{23} + 0.0000877841x^{24} \\ +0.0000822511x^{25} + 0.0000775660x^{26} + 0.0000733644x^{27} \\ +0.0000697260x^{28} + 0.0000664825x^{29} + 0.0000636670x^{30} \\ +0.0000611618x^{31} + 0.0000589626x^{32} + 0.0000570479x^{33} \\ +0.0000554059x^{34} + 0.0000539797x^{35} + 0.0000527396x^{36} \\ +0.0000517750x^{37} + 0.0000508221x^{38} + 0.0000500451x^{39} \\ +0.0000494506x^{40} + 0.0000489172x^{41} + 0.0000484412x^{42} \\ +0.0000481137x^{43} + 0.0000476051x^{44} + 0.0000472480x^{45} \\ +0.0000467789x^{46} + 0.0000463338x^{47} + 0.0000458079x^{48} \\ +0.0000453462x^{49} + 0.0000446644x^{50} + 0.0000441249x^{51} \\ +0.0000436608x^{52} + 0.0000435132x^{53} + 0.0000440380x^{54} \end{array} \right.$$

$$\rho(x) = x^{17}$$

10. Code compression rate $R = 0.2500$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.1118170000x + 0.1479280000x^2 + 0.0721407000x^5 \\ +0.2464250000x^6 + 0.0021321100x^8 + 0.4195580000x^{29} \end{array} \right.$$

$$\rho(x) = x^{23}$$

11. Code compression rate $R = 0.2780$. Obtained from EXIT chart based design.

$$\lambda(x) = \left\{ \begin{array}{c} 0.0365683268x + 0.4339804675x^2 + 0.0001299819x^3 \\ +0.0171330361x^4 + 0.0023872502x^5 + 0.1923109345x^6 \\ +0.0016539566x^7 + 0.0012354770x^8 + 0.0009357160x^9 \\ +0.0010813697x^{10} + 0.0013693568x^{11} + 0.0024491406x^{12} \\ +0.0001371963x^{13} + 0.2953988307x^{14} + 0.0072606502x^{15} \\ +0.0021649586x^{16} + 0.0009936879x^{17} + 0.0005717216x^{18} \\ +0.0003736405x^{19} + 0.0002657385x^{20} + 0.0002003367x^{21} \\ +0.0001578227x^{22} + 0.0001285540x^{23} + 0.0001075570x^{24} \\ +0.0000919092x^{25} + 0.0000799746x^{26} + 0.0000706532x^{27} \\ +0.0000632326x^{28} + 0.0000571490x^{29} + 0.0000521729x^{30} \\ +0.0000480070x^{31} + 0.0000444806x^{32} + 0.0000414675x^{33} \\ +0.0000389182x^{34} + 0.0000366747x^{35} + 0.0000347150x^{36} \\ +0.0000329633x^{37} + 0.0000314569x^{38} + 0.0000301197x^{39} \\ +0.0000288897x^{40} + 0.0000278226x^{41} + 0.0000268317x^{42} \\ +0.0000259364x^{43} + 0.0000251422x^{44} + 0.0000244178x^{45} \\ +0.0000237279x^{46} + 0.0000231042x^{47} + 0.0000225359x^{48} \\ +0.0000219885x^{49} \end{array} \right.$$

$$\rho(x) = 0.3x^{15} + 0.7x^{16}$$

12. Code compression rate $R = 0.3000$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.1392280000x + 0.2007590000x^2 + 0.2522010000x^6 \\ +0.0134136000x^{11} + 0.1710390000x^{17} + 0.0424794000x^{31} \\ +0.0855733000x^{41} + 0.0953074000x^{49} \end{array} \right.$$

$$\rho(x) = 0.3x^{16} + 0.7x^{17}$$

13. Code compression rate $R = 0.3277$. Obtained from EXIT chart based design.

$$\lambda(x) = \left\{ \begin{array}{c} 0.0530701457x + 0.3670818133x^2 + 0.0000467414x^3 \\ +0.0001693434x^4 + 0.0001839202x^5 + 0.0069791251x^6 \\ +0.2340869351x^7 + 0.0293543350x^8 + 0.0010512763x^9 \\ +0.0005884899x^{10} + 0.0004336224x^{11} + 0.0004421017x^{12} \\ +0.0003594770x^{13} + 0.0004299928x^{14} + 0.0004909475x^{15} \\ +0.0005842977x^{16} + 0.0006793042x^{17} + 0.0007968070x^{18} \\ +0.0012180480x^{19} + 0.0004888936x^{20} + 0.0233161791x^{21} \\ +0.1332294395x^{22} + 0.1290474222x^{23} + 0.0085915359x^{24} \\ +0.0029141588x^{25} + 0.0012946281x^{26} + 0.0007207962x^{27} \\ +0.0004649800x^{28} + 0.0003297155x^{29} + 0.0002492324x^{30} \\ +0.0001973730x^{31} + 0.0001619024x^{32} + 0.0001364856x^{33} \\ +0.0001174977x^{34} + 0.0001029730x^{35} + 0.0000914730x^{36} \\ +0.0000822134x^{37} + 0.0000744651x^{38} + 0.0000681241x^{39} \\ +0.0000626681x^{40} + 0.0000580978x^{41} + 0.0000541896x^{42} \\ +0.0000508900x^{43} + 0.0000479426x^{44} \end{array} \right.$$

$$\rho(x) = 0.4x^{14} + 0.6x^{15}$$

14. Code compression rate $R = 0.3500$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.1577030000x + 0.1991060000x^2 + 0.0324838000x^5 \\ +0.1985480000x^6 + 0.0807045000x^7 + 0.2997880000x^{21} \\ +0.0310813000x^{22} + 0.0005852400x^{33} \end{array} \right.$$

$$\rho(x) = x^{13}$$

15. Code compression rate $R = 0.3765$. Obtained from EXIT chart based design.

$$\lambda(x) = \left\{ \begin{array}{c} 0.0749746514x + 0.3276397454x^2 + 0.0000133230x^3 \\ +0.0000334288x^4 + 0.0000536346x^5 + 0.0153727931x^6 \\ +0.0815021268x^7 + 0.2102399279x^8 + 0.0002396780x^9 \\ +0.0001470809x^{10} + 0.0001101496x^{11} + 0.0000966061x^{12} \\ +0.0000903891x^{13} + 0.0000890635x^{14} + 0.0000914754x^{15} \\ +0.0000970680x^{16} + 0.0001067689x^{17} + 0.0001207840x^{18} \\ +0.0001414627x^{19} + 0.0001706132x^{20} + 0.0002138222x^{21} \\ +0.0002778502x^{22} + 0.0003801933x^{23} + 0.0005575559x^{24} \\ +0.0009942277x^{25} + 0.0038202879x^{26} + 0.1061438593x^{27} \\ +0.1680729114x^{28} + 0.0046663975x^{29} + 0.0013383072x^{30} \\ +0.0007207862x^{31} + 0.0004404507x^{32} + 0.0002851821x^{33} \\ +0.0002019419x^{34} + 0.0001557353x^{35} + 0.0001265960x^{36} \\ +0.0001058190x^{37} + 0.0000899191x^{38} + 0.0000773867x^{39} \end{array} \right.$$

$$\rho(x) = 0.2x^{12} + 0.8x^{13}$$

16. Code compression rate $R = 0.4010$. Obtained from EXIT chart based design.

$$\lambda(x) = \begin{cases} 0.0665052292x + 0.3059480919x^2 + 0.0001148099x^3 \\ +0.0008967976x^4 + 0.0004931189x^5 + 0.0885292799x^6 \\ +0.0028298017x^7 + 0.0933425249x^8 + 0.0270974961x^9 \\ +0.0288241934x^{10} + 0.0067485256x^{11} + 0.0000092266x^{12} \\ +0.0622890112x^{13} + 0.0120946708x^{14} + 0.0041578038x^{15} \\ +0.0023011096x^{16} + 0.0016328252x^{17} + 0.0015906365x^{18} \\ +0.0014174658x^{19} + 0.0012243279x^{20} + 0.0010393924x^{21} \\ +0.0008685978x^{22} + 0.0007279214x^{23} + 0.0006136080x^{24} \\ +0.0005260355x^{25} + 0.0004590425x^{26} + 0.0004104392x^{27} \\ +0.0003758855x^{28} + 0.0003539140x^{29} + 0.0003424701x^{30} \\ +0.0003416727x^{31} + 0.0003517762x^{32} + 0.0003755456x^{33} \\ +0.0004182788x^{34} + 0.0004931475x^{35} + 0.0006325812x^{36} \\ +0.0009463712x^{37} + 0.0020643746x^{38} + 0.2806119994x^{39} \end{cases}$$

$$\rho(x) = x^{13}$$

17. Code compression rate $R = 0.4257$. Obtained from EXIT chart based design.

$$\lambda(x) = \begin{cases} 0.1242596346x + 0.3029532976x^2 + 0.0000000503x^3 \\ +0.1130781949x^4 + 0.0000019471x^5 + 0.1030539620x^6 \\ +0.0000011974x^7 + 0.0000000440x^8 + 0.0000002074x^9 \\ +0.0000002709x^{10} + 0.0000003055x^{11} + 0.0000003467x^{12} \\ +0.0000004646x^{13} + 0.3566500769x^{14} \end{cases}$$

$$\rho(x) = 0.5x^9 + 0.5x^{10}$$

18. Code compression rate $R = 0.4508$. Obtained from EXIT chart based design.

$$\lambda(x) = \begin{cases} 0.0999701013x + 0.2884808570x^2 + 0.0000004467x^3 \\ +0.0000007193x^4 + 0.0000015285x^5 + 0.2323194203x^6 \\ +0.0053583039x^7 + 0.0000044615x^8 + 0.0000014946x^9 \\ +0.0000008772x^{10} + 0.0000006229x^{11} + 0.0000004958x^{12} \\ +0.0000004363x^{13} + 0.0000004124x^{14} + 0.0000003973x^{15} \\ +0.0000004041x^{16} + 0.0000004193x^{17} + 0.0000004649x^{18} \\ +0.0000005231x^{19} + 0.0000006215x^{20} + 0.0000008004x^{21} \\ +0.0000011595x^{22} + 0.0000022604x^{23} + 0.3738527719x^{24} \end{cases}$$

$$\rho(x) = 0.6x^{10} + 0.4x^{11}$$

19. Code compression rate $R = 0.4754$. Obtained from EXIT chart based design.

$$\lambda(x) = \left\{ \begin{array}{c} 0.1153122367x + 0.2911329163x^2 + 0.0000013866x^3 \\ +0.0000237308x^4 + 0.0000055074x^5 + 0.1563430715x^6 \\ +0.1185591176x^7 + 0.0000123470x^8 + 0.0000039402x^9 \\ +0.0000021293x^{10} + 0.0000016722x^{11} + 0.0000014129x^{12} \\ +0.0000012628x^{13} + 0.0000011759x^{14} + 0.0000011534x^{15} \\ +0.0000011760x^{16} + 0.0000012159x^{17} + 0.0000013357x^{18} \\ +0.0000015028x^{19} + 0.0000017858x^{20} + 0.0000022500x^{21} \\ +0.0000032496x^{22} + 0.0000062909x^{23} + 0.3185781326x^{24} \end{array} \right.$$

$$\rho(x) = 0.7x^9 + 0.3x^{10}$$

20. Code compression rate $R = 0.5000$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.1527930000x + 0.2823500000x^2 + 0.0062193000x^3 \\ +0.5586370000x^{19} \end{array} \right.$$

$$\rho(x) = x^9$$

21. Code compression rate $R = 0.5261$. Obtained from EXIT chart based design.

$$\lambda(x) = \left\{ \begin{array}{c} 0.1460827952x + 0.2706599775x^2 + 0.0375150106x^4 \\ +0.2419193856x^6 + 0.3038228311x^{21} \end{array} \right.$$

$$\rho(x) = 0.3x^7 + 0.7x^8$$

22. Code compression rate $R = 0.5515$. Obtained from EXIT chart based design.

$$\lambda(x) = \left\{ \begin{array}{c} 0.1483309314x + 0.2284719333x^2 + 0.0000030096x^3 \\ +0.0949413930x^4 + 0.0001039602x^5 + 0.1805360736x^6 \\ +0.0000004930x^7 + 0.0000044589x^8 + 0.0000001607x^9 \\ +0.0000056705x^{10} + 0.0000009369x^{11} + 0.0000017390x^{12} \\ +0.0000020428x^{13} + 0.0000021972x^{14} + 0.0000024381x^{15} \\ +0.0000027101x^{16} + 0.0000031452x^{17} + 0.0000038634x^{18} \\ +0.0000048678x^{19} + 0.0000067093x^{20} + 0.0000100212x^{21} \\ +0.0000167482x^{22} + 0.0000309110x^{23} + 0.3475135857x^{24} \end{array} \right.$$

$$\rho(x) = 0.3x^7 + 0.7x^8$$

23. Code compression rate $R = 0.5765$. Obtained from EXIT chart based design.

$$\lambda(x) = \left\{ \begin{array}{c} 0.1603735738x + 0.2493334680x^2 + 0.0000000001x^3 \\ +0.1398432549x^4 + 0.0009310009x^6 + 0.1919102356x^8 \\ +0.2576084667x^{24} \end{array} \right.$$

$$\rho(x) = 0.2x^6 + 0.8x^7$$

24. Code compression rate $R = 0.6006$. Obtained from EXIT chart based design.

$$\lambda(x) = \begin{cases} 0.2365103129x + 0.0000003166x^2 + 0.3137349967x^3 \\ +0.0714882030x^4 + 0.0000017984x^5 + 0.0000026206x^6 \\ +0.0000020295x^7 + 0.0000013959x^8 + 0.0000009058x^9 \\ +0.1706734148x^{10} + 0.0004347797x^{11} + 0.0000483848x^{12} \\ +0.0000733639x^{13} + 0.0000728850x^{14} + 0.0000680328x^{15} \\ +0.0000796862x^{16} + 0.0000905415x^{17} + 0.0001139106x^{18} \\ +0.0001440668x^{19} + 0.0002042075x^{20} + 0.0003112035x^{21} \\ +0.0006324201x^{22} + 0.0182989558x^{23} + 0.1854026561x^{24} \\ +0.0016072092x^{25} + 0.0000017023x^{26} \end{cases}$$

$$\rho(x) = 0.9x^6 + 0.1x^7$$

25. Code compression rate $R = 0.6257$. Obtained from EXIT chart based design.

$$\lambda(x) = \begin{cases} 0.2345281333x + 0.2494103165x^3 + 0.0868495874x^4 \\ +0.1048097509x^6 + 0.3244022119x^{23} \end{cases}$$

$$\rho(x) = 0.9x^6 + 0.1x^7$$

26. Code compression rate $R = 0.6500$. Obtained from LTHC database.

$$\lambda(x) = \begin{cases} 0.2454340000x + 0.1921240000x^2 + 0.1357320000x^5 \\ +0.0838990000x^6 + 0.1116600000x^{12} + 0.0029827600x^{14} \\ +0.0222593000x^{15} + 0.0742901000x^{28} + 0.1316190000x^{32} \end{cases}$$

$$\rho(x) = 0.5x^5 + 0.5x^6$$

27. Code compression rate $R = 0.6800$. Obtained from LTHC database.

$$\lambda(x) = \begin{cases} 0.2177260000x + 0.1634340000x^2 + 0.0001449710x^3 \\ +0.0000070738x^4 + 0.0980647000x^5 + 0.1018190000x^6 \\ +0.0537834000x^{13} + 0.0301359000x^{16} + 0.0566144000x^{20} \\ +0.0109644000x^{26} + 0.0808932000x^{30} + 0.0000059471x^{97} \\ +0.0001740400x^{98} + 0.1862340000x^{99} \end{cases}$$

$$\rho(x) = 0.9x^6 + 0.1x^7$$

28. Code compression rate $R = 0.7000$. Obtained from LTHC database.

$$\lambda(x) = \begin{cases} 0.2202400000x + 0.1604510000x^2 + 0.1219000000x^5 \\ +0.0669837000x^6 + 0.0728829000x^{12} + 0.0056090100x^{19} \\ +0.0223284000x^{21} + 0.0531729000x^{22} + 0.0496530000x^{25} \\ +0.0222808000x^{26} + 0.2044980000x^{99} \end{cases}$$

$$\rho(x) = 0.1x^5 + 0.9x^6$$

29. Code compression rate $R = 0.7300$. Obtained from LTHC database.

$$\lambda(x) = \begin{cases} 0.2374900000x + 0.1643770000x^2 + 0.1482710000x^5 \\ +0.0442728000x^6 + 0.0276623000x^{13} + 0.1174440000x^{15} \\ +0.0384218000x^{30} + 0.0368359000x^{35} + 0.0319294000x^{37} \\ +0.1532960000x^{99} \end{cases}$$

$$\rho(x) = 0.7x^5 + 0.3x^6$$

30. Code compression rate $R = 0.7500$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.2911570000x + 0.1891740000x^2 + 0.0408389000x^4 \\ +0.0873393000x^5 + 0.0074271800x^6 + 0.1125810000x^7 \\ +0.0925954000x^{15} + 0.0186572000x^{20} + 0.1240640000x^{32} \\ +0.0160020000x^{39} + 0.0201644000x^{44} \end{array} \right.$$

$$\rho(x) = 0.8x^4 + 0.2x^5$$

31. Code compression rate $R = 0.7800$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.2547740000x + 0.1634760000x^2 + 0.0032539300x^4 \\ +0.1524220000x^5 + 0.0331399000x^6 + 0.0038860400x^9 \\ +0.0189110000x^{12} + 0.0998195000x^{14} + 0.0151103000x^{27} \\ +0.0769337000x^{29} + 0.0218393000x^{32} + 0.1564350000x^{99} \end{array} \right.$$

$$\rho(x) = 0.3x^4 + 0.7x^5$$

32. Code compression rate $R = 0.8000$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.2920250000x + 0.1739820000x^2 + 0.0523131000x^4 \\ +0.0257749000x^5 + 0.1220460000x^6 + 0.0218315000x^8 \\ +0.0209295000x^{10} + 0.0322251000x^{14} + 0.1127710000x^{23} \\ +0.0001708020x^{25} + 0.0328124000x^{31} + 0.0274748000x^{44} \\ +0.0048302000x^{53} + 0.0126282000x^{59} + 0.0681855000x^{99} \end{array} \right.$$

$$\rho(x) = x^4$$

33. Code compression rate $R = 0.8200$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.3037920000x + 0.1731880000x^2 + 0.0671337000x^4 \\ +0.0123568000x^5 + 0.1341320000x^6 + 0.0314767000x^{12} \\ +0.0108393000x^{14} + 0.0256390000x^{16} + 0.0910351000x^{19} \\ +0.0400076000x^{39} + 0.0000240473x^{45} + 0.0117242000x^{51} \\ +0.0189157000x^{57} + 0.0112433000x^{62} + 0.0684922000x^{76} \end{array} \right.$$

$$\rho(x) = 0.2x^3 + 0.8x^4$$

34. Code compression rate $R = 0.8500$. Obtained from LTHC database.

$$\lambda(x) = \left\{ \begin{array}{c} 0.3151270000x + 0.1902840000x^2 + 0.0449124000x^4 \\ +0.1705930000x^6 + 0.1405970000x^{17} + 0.0081261000x^{37} \\ +0.0440236000x^{41} + 0.0863369000x^{66} \end{array} \right.$$

$$\rho(x) = 0.5x^3 + 0.5x^4$$

35. Code compression rate $R = 0.8800$. Obtained from LTHC database.

$$\lambda(x) = \begin{cases} 0.3424730000x + 0.1650060000x^2 + 0.1203830000x^4 \\ +0.0191956000x^5 + 0.0120714000x^6 + 0.1416920000x^{10} \\ +0.0211997000x^{25} + 0.0201976000x^{26} + 0.0185881000x^{34} \\ +0.0428897000x^{36} + 0.0133019000x^{38} + 0.0021735800x^{39} \\ +0.0104203000x^{40} + 0.0704081000x^{99} \end{cases}$$

$$\rho(x) = 0.8x^3 + 0.2x^4$$

36. Code compression rate $R = 0.9000$. Obtained from LTHC database.

$$\lambda(x) = \begin{cases} 0.3585670000x + 0.1663620000x^2 + 0.0000299853x^3 \\ +0.0487523000x^4 + 0.1205300000x^5 + 0.0004778820x^6 \\ +0.0000422043x^7 + 0.0409013000x^{10} + 0.0744850000x^{13} \\ +0.0339421000x^{25} + 0.0076194000x^{30} + 0.0564230000x^{34} \\ +0.0918683000x^{99} \end{cases}$$

$$\rho(x) = x^3$$

37. Code compression rate $R = 0.9300$. Obtained from LTHC database.

$$\lambda(x) = \begin{cases} 0.4050180000x + 0.1716200000x^2 + 0.0995717000x^4 \\ +0.0446767000x^5 + 0.0379776000x^6 + 0.0612300000x^{10} \\ +0.0188277000x^{14} + 0.0332702000x^{16} + 0.0026478100x^{17} \\ +0.0127722000x^{20} + 0.0435222000x^{28} + 0.0075207600x^{50} \\ +0.0123120000x^{52} + 0.0258378000x^{62} + 0.0065513300x^{63} \\ +0.0166443000x^{71} \end{cases}$$

$$\rho(x) = 0.4x^2 + 0.6x^3$$

38. Code compression rate $R = 0.9500$. Obtained from LTHC database.

$$\lambda(x) = \begin{cases} 0.4145410000x + 0.1667480000x^2 + 0.0971414000x^4 \\ +0.0737392000x^5 + 0.0007658270x^6 + 0.0022987300x^8 \\ +0.0118195000x^9 + 0.0751327000x^{11} + 0.0575786000x^{19} \\ +0.0063649900x^{26} + 0.0046459300x^{35} + 0.0171996000x^{43} \\ +0.0443262000x^{62} + 0.0111913000x^{82} + 0.0165064000x^{99} \end{cases}$$

$$\rho(x) = 0.5x^2 + 0.5x^3$$

Appendix B

Deriving the Slepian-Wolf Error Exponent From the Channel Coding Error Exponent

In this section we show that Slepian-Wolf source coding is equivalent to a channel coding problem by deriving the Slepian-Wolf error exponent from the random coding exponent. Because their forms are somewhat simpler to manipulate, in this appendix we use the forms of these exponents given by Csiszár and Körner [22]. Their form of the Slepian-Wolf exponent is [22, Exercise 3.1.5, p. 264]:

$$\min_{P_{\tilde{y}|\tilde{x}}, P_{\tilde{x}}} D(P_{\tilde{x},\tilde{y}} \| Q_{x,y}) + \max\{0, R_{sw} - H(\tilde{x}|\tilde{y})\}, \tag{B.1}$$

where R_{sw} is the encoding rate, $Q_{x,y} = Q_{y|x} Q_x$ is the underlying joint distribution defined by the side information channel $Q_{y|x}$ and source distribution Q_x, and $\tilde{x}$ and $\tilde{y}$ are random variables corresponding to the arbitrary distribution $P_{\tilde{x},\tilde{y}} = P_{\tilde{y}|\tilde{x}} P_{\tilde{x}}$.

The constant-composition random coding error exponent for channel $Q_{y|x}$ where codewords are selected uniformly from all vectors of type (empirical distribution) $P_{\tilde{x}}$ is [22, Thm. 2.5.2, p. 165]:

$$\inf_{P_{\tilde{y}|\tilde{x}}} D(P_{\tilde{y}|\tilde{x}} \| Q_{y|x} | P_{\tilde{x}}) + \max\{0, I(\tilde{x};\tilde{y}) - R_{cc}\}, \tag{B.2}$$

where R_{cc} is the code rate. Note, the Slepian-Wolf and random-coding exponents of Equation (B.1) and Equation (B.2) equal the maximum-likelihood exponents derived by Gallager in [28] and [27].

To show how to derive Equation (B.1) from Equation (B.2) we use the following Slepian-Wolf code. The encoder first calculates the type (empirical distribution) of the observed random source sequence $\mathbf{x}$ and communicates this type to the decoder. Call this type $P_{\tilde{x}}$, and note that this initial communication does not cost us any rate asymptotically since the number of types is only polynomial in the block-length. For each type, the encoder and decoder share a random partition of the sequences in that type class. The partition is made by uniformly randomly binning the sequences of the given type into $\exp\{NR_{sw}\}$ bins, so that each bin consists of roughly $\exp\{N[H(P_{\tilde{x}}) - R_{sw}]\}$ sequences uniformly selected from

the type-class specified by $P_{\tilde{x}}$. From the decoder's point of view, this bin of sequences is the same as a randomly-generated constant composition code of rate $H(P_{\tilde{x}}) - R_{sw}$. The decoder observes one of these sequences through the channel $Q_{y|x}$.

The probability that the source $\mathbf{x}$, which is distributed according to Q_x, has type $P_{\tilde{x}}$ is upper-bounded by $\exp\{-ND(P_{\tilde{x}}\|Q_x)\}$. Given that the source is of type $P_{\tilde{x}}$, the probability that we subsequently make a decoding error is bounded by Equation (B.2) where $R_{cc} = H(P_{\tilde{x}}) - R_{sw}$. The overall probability of error is therefore bounded by the probability of the jointly worst-case source type $P_{\tilde{x}}$ and channel law $P_{\tilde{y}|\tilde{x}}$ which, using the divergence expansion $D(P_{\tilde{x},\tilde{y}}\|Q_{x,y}) = D(P_{\tilde{y}|\tilde{x}}\|Q_{y|x}|P_{\tilde{x}}) + D(P_{\tilde{x}}\|Q_x)$, can be seen to equal Equation (B.1).