JOINT JPEG2000/LDPC CODE SYSTEM DESIGN FOR IMAGE TELEMETRY

Undergraduate Students:
Kristin Jagiello, Mahmut Zafer Aydin and Wei-Ren Ng
Faculty Advisors:
William E. Ryan, Michael W. Marcellin and Ali Bilgin
ECE Dept., University of Arizona, Tucson, AZ 85721

ABSTRACT

This paper considers the joint selection of the source code rate and channel code rate in an image telemetry system. Specifically considered is the JPEG2000 image coder and an LDPC code family. The goal is to determine the optimum apportioning of bits between the source and channel codes for a given channel signal-to-noise ratio and total bit rate, R_{total}. Optimality is in the sense of maximum peak image SNR and the tradeoff is between the JPEG2000 bit rate R_{source} and the LDPC code rate $R_{channel}$. For comparison, results are included for the industry standard rate-1/2, memory-6 convolutional code.

Keywords: JPEG2000, LDPC codes, joint source/channel coding.

1. INTRODUCTION

Compression of an image followed by transmission over a channel suffers from two sources of distortion: distortion due to quantization error at the source coder and distortion due to bit errors caused by channel noise. The first type of distortion can be reduced by increasing the source code rate R_{source} and the second type of distortion can be reduced by decreasing the channel code rate $R_{channel}$. Both increasing R_{source} and decreasing $R_{channel}$ increases the channel bit rate R_{total} (channel bits/pixel). Thus, for a fixed channel bit rate R_{total}, there is a tradeoff between R_{source} and $R_{channel}$ and we seek the optimum tradeoff, where optimality is in the sense of maximizing PSNR, defined below.

We consider a simple telemetry system on a binary-input additive white Gaussian noise (AWGN) channel. JPEG2000 compression is considered in conjunction with an LDPC channel code as well as a convolutional channel code. Results generated by the simulation of this telemetry model reveal the optimal apportioning of bits between the channel and source coders.

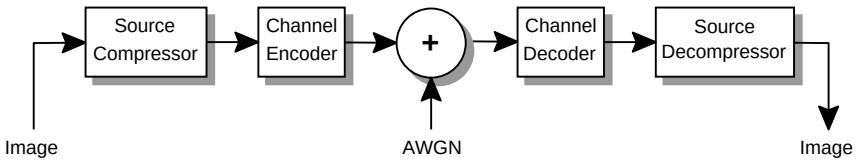


Figure 1. A block diagram of the system model.

Section 2 describes the system models used in the simulations. Section 3 provides details of the source and channel codes used in this study. Section 4 presents simulation results for the code system design and Section 5 concludes the paper.

2. SYSTEM MODELS

The image used in the simulations was a downsampled and cropped version of the eight-bit grey scale JPEG2000 Test Image "Aerial1." The image was input to the system depicted in Figure 1. The source compressor and decompressor was JPEG2000 [1]. The two channel codes used were LDPC and convolutional. The binary-input AWGN channel was assumed here although the approach is easily extendable to other channel models.

A variety of source code and channel code rate combinations were considered for several total rates, R_{total}, given by

$$R_{total} = R_{source}/R_{channel}. \tag{1}$$

R_{source} is in units of source code bits/pixel and $R_{channel}$ is in units of source code bits/channel code bit, so that R_{total} is in units of channel code bits/pixel. The source code rate, R_{source}, was adjusted via the inherent rate scalability of JPEG2000. The channel code rate was adjusted by puncturing a rate-1/2 "mother code" for both the LDPC code and the convolutional code.

The goal of the simulations is to find the optimum tradeoff between R_{source} and $R_{channel}$ for a fixed R_{total} and channel signal-to-noise ratio (SNR), where optimality is in the sense of maximizing the peak SNR (PSNR), defined as

$$PSNR = 10\log_{10}\left(\frac{255^2}{MSE}\right) \tag{2}$$

assuming an eight-bit grey scale image where pixel values range between 0 and 255. MSE is the mean-squared error and is computed as

$$MSE = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=i}^{M}(x_{ij} - \hat{x}_{ij})^2 \tag{3}$$

where x_{ij} represents the original pixel values and $\hat{x}_{ij}$ represents the decompressed pixel values. N and M are the respective width and height of the image in pixels.

3. SIMULATION OVER THE AWGN CHANNEL

3.1. JPEG2000 Setup

The JPEG2000 code-stream begins with a main header which contains vital global information necessary for proper decompression of the entire code-stream. After the main header, the code-stream is divided into a sequence of tile-streams. Each tile-stream has layers that contain packets which hold compressed data from a precinct at a resolution level. Precincts consist of code-blocks from all sub-bands at a single resolution level and each code-block contributes a certain number of coding passes to each packet [2].

JPEG2000 implements an arithmetic coder known as the MQ coder. Both encoder and decoder need to maintain synchronization to properly decode a code-stream. JPEG2000 includes error resilience (ER) tools that promote robustness to bit-errors. These tools have two purposes: hierarchical data partitioning and resynchronization, and error detection and isolation [1], [3].

JPEG2000 ER tools stop bit errors from propagating from one code-block to another as long as the code-block synchronization is maintained [1]. The smallest coding unit is code-block data which are grouped together consisting of packet headers and code-block bytes. The packet headers allow the JPEG2000 decoder to determine the correct number of code-block bytes. This means that even though the code-block may be damaged, the decoder will be able to resynchronize the undamaged code-blocks.

In this paper, Kakadu software version 6 and ER parameters "Cmodes=RESTART|ERTERM," "Cprecincts={128,128}," "Cuse_sop=yes," and "Cuse_eph=yes" were used [4]. In RESTART mode, the MQ coder restarts at the beginning of each coding pass. The MQ codeword segment is properly terminated and the coder is reinitialized. The ERTERM (Error Resilient Termination) mode is a predictable termination policy that is employed by the encoder to detect errors introduced into either the bit-stream or the length values in packet headers. The decoder can identify the raw codeword segment in which the error first occurs and discards the codeword segment to conceal the visual artifact.

"Cuse_sop=yes" and "Cuse_eph=yes" are marker segments for ER decoding and for locating packets and/or packet headers. "Cuse_sop=yes" activates the START OF PACKET (SOP) marker segment and "Cuse_eph=yes," activates the END OF PACKET HEADER (EPH). These ER modes add a SOP marker segment before each packet in the code-stream and add an isolated EPH marker after every packet header. These markers are extra ER tools in JPEG2000 that are useful for code-stream parsing and ER decoding while also enabling resynchronization during detection of errors.

"Cprecincts={128,128}" sets the precinct dimensions, organizing compressed data within the code-stream. This ER tool allows the image to be divided into smaller rectangular regions which are treated independently for compression purposes, in this case a rectangular 128-by-128 region. The partitioning is useful so that a single error will damage only a limited region within an image. It does not do anything for small images, but can be useful for larger images.

3.2. LDPC CODE FAMILY

The JPEG2000 compressed images served as the input to the channel encoder. The code that was used was a rate 1/2, (4096, 2048), structured irregular repeat-accumulate (S-IRA) LDPC code [5]. The encoder takes 2048 bits of data and encodes them by adding 2048 parity bits. The channel encoder is a module that performs encoding by iteratively solving for 2048 parity bits from the parity check matrix $\mathbf{H}$, where

$$\mathbf{H} = [\mathbf{H_1 H_2}]. \tag{4}$$

$\mathbf{H_1}$ is a 2048 $\times$ 2048 matrix having row-weight 5 and column-weight 5. $\mathbf{H_2}$ is a 2048 $\times$ 2048 "dual-diagonal" matrix arranged as

$$\mathbf{H_2} = \begin{bmatrix} 1 & & & & & \\ 1 & 1 & & & & \\ & \ddots & & & & \\ & & & 1 & 1 & \\ & & & & 1 & 1 \end{bmatrix}. \tag{5}$$

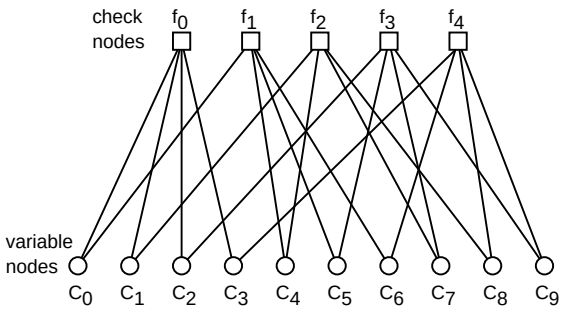
Let $\mathbf{h}_u$ represent the u^{th} row of $\mathbf{H}$ where $u \in \{0, \ldots, 2047\}$. Each row of $\mathbf{H}$ may be used to set up an equation to solve for a single parity bit in the codeword. This is because $\mathbf{c} \cdot \mathbf{h}_u$=0 must hold for each codeword $\mathbf{c}$ and each row $\mathbf{h}_u$. Thus, the locations of the ones in $\mathbf{h}_u$ indicate which information bits should be used in solving for the unknown parity bit. The unknown bit is equal to the result of the binary addition of the bits indicated by $\mathbf{h}_u$. The dual-diagonal organization of $\mathbf{H_1}$ facilitates encoding in that it results in a single, unknown parity bit in $\mathbf{h}_0$. The remaining equations for $u > 0$ depend upon the parity bit solved in the previous row. That is, $\mathbf{h}_u$ depends upon the parity bit solved for in $\mathbf{h}_{u-1}$. Thus it is necessary to solve for the parity bits in increasing order beginning with the one located in $\mathbf{h}_0$.

After encoding using the rate-1/2 encoder just described, depending on the code rate $R_{channel}$ desired, parity bits can be punctured (deleted) prior to transmission. For example, for $R_{channel} = 2/3$, every other parity bit is transmitted. The eventual transmitted channel code bits are transmitted in the bipolar form +1/-1 to model BPSK modulation. AWGN samples are added to each transmitted code bit.

The corrupted bits are then decoded using the on-off attenuated min-sum algorithm [6]. The decoding algorithm can be visualized with the help of a Tanner graph: a bipartite graph

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

a)



b)

Figure 2. An example of a Tanner graph and its matrix representation.

whose adjacency matrix is the code's parity check matrix **H**. That is, there are two node types, the variable nodes (VNs) and the check nodes (CNs), and only nodes of differing types may be connected. Further, the VNs correspond to the columns of **H** and the CNs correspond to the rows of **H**. Figure 2 shows a matrix **H** and its Tanner graph representation. For example, CN f_0 is connected to variable nodes c_0, c_1, c_2 and c_3 indicating that row zero of **H** has ones in columns zero, one, two and three. Thus, bits zero, one, two and three of the codeword generated using **H** must have the binary sum equal to zero, i.e., $\mathbf{c} \cdot \mathbf{h}_0 = 0$.

The nodes can be likened to processors where messages are computed, stored and then passed along the edges of the graph. Messages are passed back and forth in an iterative fashion until a codeword is found or the maximum number of iterations has been reached. The upward message from VN i to CN j is given by

$$q_{ij} = s_i + \sum_{j' \neq j} r_{j'i} \tag{6}$$

where s_i is the channel sample for code bit c_i, $r_{j'i}$ is the downward message from CN j' to VN i, and the summation is over all CNs connected to VN i, excluding CN j. All such messages are computed and transmitted upward to the CNs, after which the CN messages are computed. The downward message from CN j to VN i is given by

$$r_{ji} = \prod_{i' \neq i} sign(q_{i'j}) \cdot \min_{i' \neq i} \{|q_{i'j}|\}, \tag{7}$$

where $sign(x) = +1$ for $x \geq 0$ and $sign(x) = -1$ for $x < 0$. In order to improve the decoding algorithm, a scale factor of 0.5 is applied to each message r_{ji} at even iterations, hence the name "on-off attenuated min sum algorithm" [6].

Code bit decisions are given by

$$\hat{c}_i = sign(Q_i) \tag{8}$$

where

$$Q_i = s_i + \sum_{j} r_{ji} \tag{9}$$

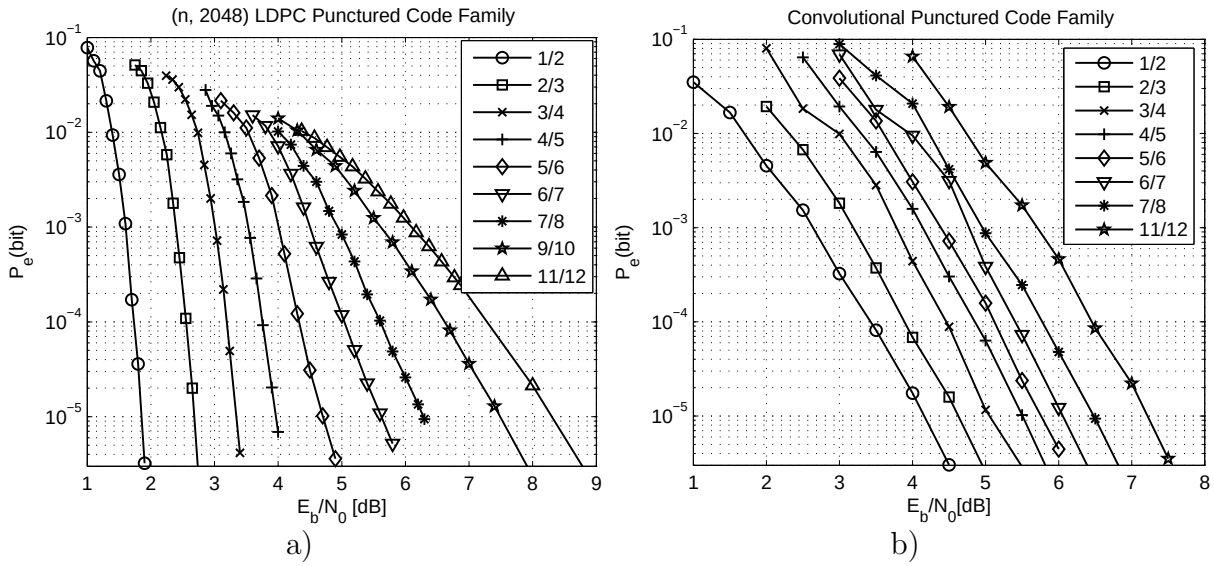


Figure 3. Plots of the simulated probability of error curves.

and where the summation is over all CNs j connected to VN i. Decoding stops when $\mathbf{cH}^T$=$\mathbf{0}$ or 50 iterations have been completed. The performance of the LDPC punctured-code family with the rate-1/2 mother code is illustrated in Figure 3(a).

3.3. CONVOLUTIONAL CODE FAMILY

For comparison to the LDPC code results, the same simulations were run using the industry standard rate-1/2 convolutional channel code together with its punctured-code descendants. A convolutional code maps k_0 information bits into n_0 code bits in a stream-oriented fashion, where $n_0 > k_0$. The transformation from information bits to code words is accomplished by a convolution of the information bits with two generator vectors, $\mathbf{g_0}$ and $\mathbf{g_1}$, to produce two code sequences C_0 and C_1.

The length of the finite memory of the convolutional generating polynomials is the "constraint length" of the code. Thus it is a constraint length $K = 7$ code. The generating vectors are $\mathbf{g_0} = 1011011$ and $\mathbf{g_1} = 1111001$, which are those used in the industry standard.

While the implementation of the convolutional encoder is straightforward, the decoding of such a coded data stream at the receiving node is more complex. In the late 1960s, A. J. Viterbi described a maximum likelihood (ML) decoding technique which greatly reduced the complexity of ML decoding. Because the Viterbi decoding algorithm can be found in many textbooks, we do not discuss it here. The performance of the convolutional punctured-code family with the rate-1/2 mother code is shown in Figure 3(b).

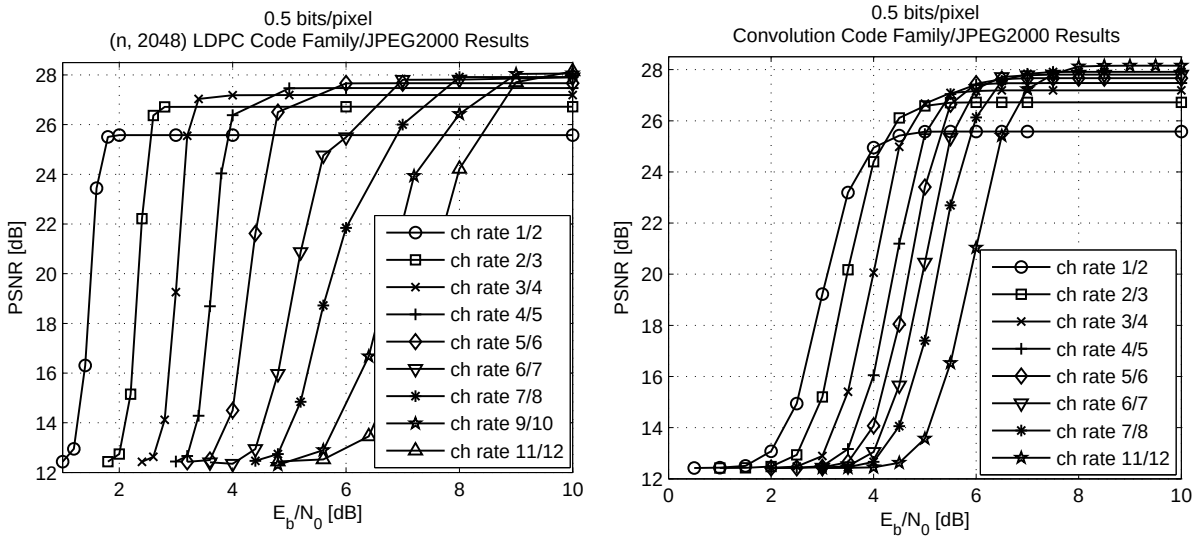


Figure 4. Plots of the average PSNR versus channel SNR for $R_{total} = 0.5$ bits/pixel. LDPC results are on the left and convolutional results are on the right.

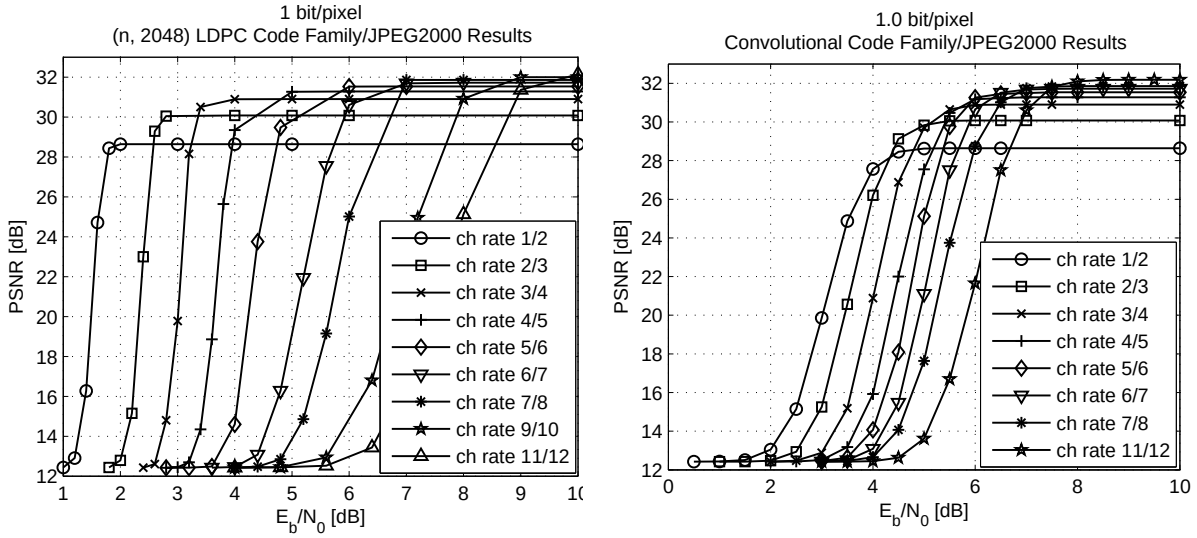


Figure 5. Plots of the average PSNR versus channel SNR for $R_{total} = 1$ bit/pixel. LDPC code results are on the left and convolutional code results are on the right.

3.4. PSNR CALCULATION

After the data have been decoded by either the LDPC code decoder or the convolutional code decoder, they are then input to the JPEG2000 decompressor. A PSNR calculation is performed for the resulting decompressed image relative to the original image per equations (2) and (3). Because of the large variance that exists in the PSNR calculation, 1000 simulations were run for each pair of R_{source} and $R_{channel}$, from which the average PSNR was computed.

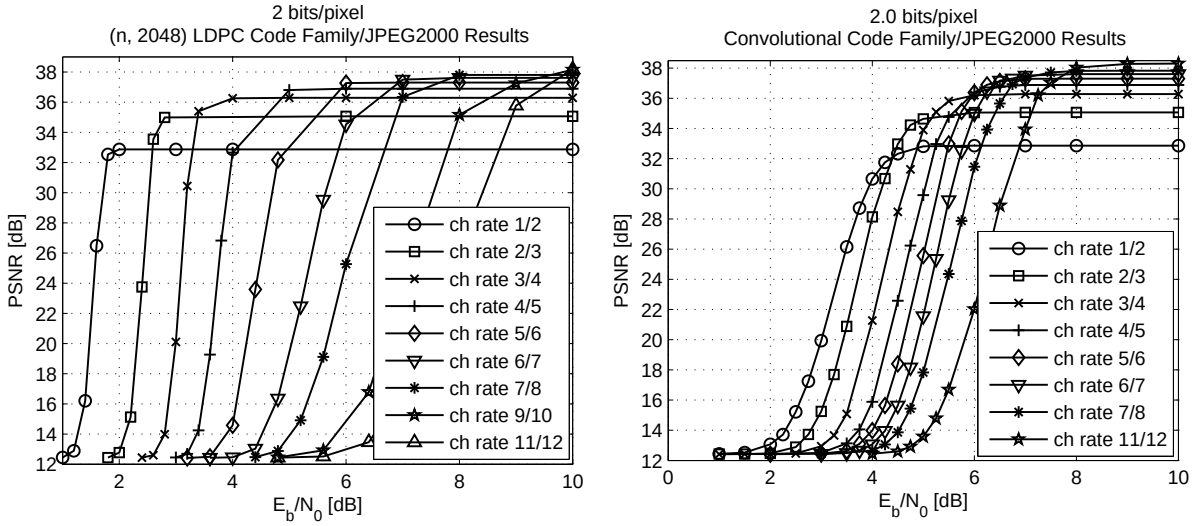


Figure 6. Plots of the average PSNR versus channel SNR for $R_{total} = 2$ bits/pixel. LDPC code results are on the left and convolutional code results are on the right.

4. RESULTS

For each pair of R_{source} and $R_{channel}$ values examined, average PSNR values were plotted as a function of SNR. Figures 4, 5, and 6 contain the LDPC/JPEG2000 and convolutional/JPEG2000 intermediate results for the three values of R_{total} considered, 0.5, 1 and 2 bits/pixel. An examination of the plots reveals that in each, the PSNR increases as the channel SNR increases until saturation occurs at a level corresponding to R_{source}. Clearly these saturation levels increase with increasing R_{total}. Further, for a given R_{total}, at high SNR, the saturation levels increase with increasing $R_{channel}$ because this corresponds to increasing R_{source}.

By fixing the SNR and plotting the PSNR as a function of $R_{channel}$, a second set of plots was produced. Figures 7, 8, and 9 contain the results generated for the LDPC/JPEG2000 and convolutional/JPEG2000 simulations. These plots demonstrate that, for a given SNR, there exists an optimal rate allocation between the source and channel codes. Although the PSNR is plotted against $R_{channel}$, it is a simple matter to determine the corresponding R_{source} using (1). It is clear from the two sets of plots that the LDPC code offers an advantage over the convolutional code for low SNR. For example, in Figure 8 at 1 bit/pixel and Eb/No = 4 dB, the LDPC code yields a maximum PSNR of 30.8 dB at $R_{channel} = 0.75$ whereas the convolutional code yields a maximum PSNR of only 27.8 dB at $R_{channel} = 0.5$. It should be noted, however, that for large SNR, the LDPC code ceases to outperform the convolutional code.

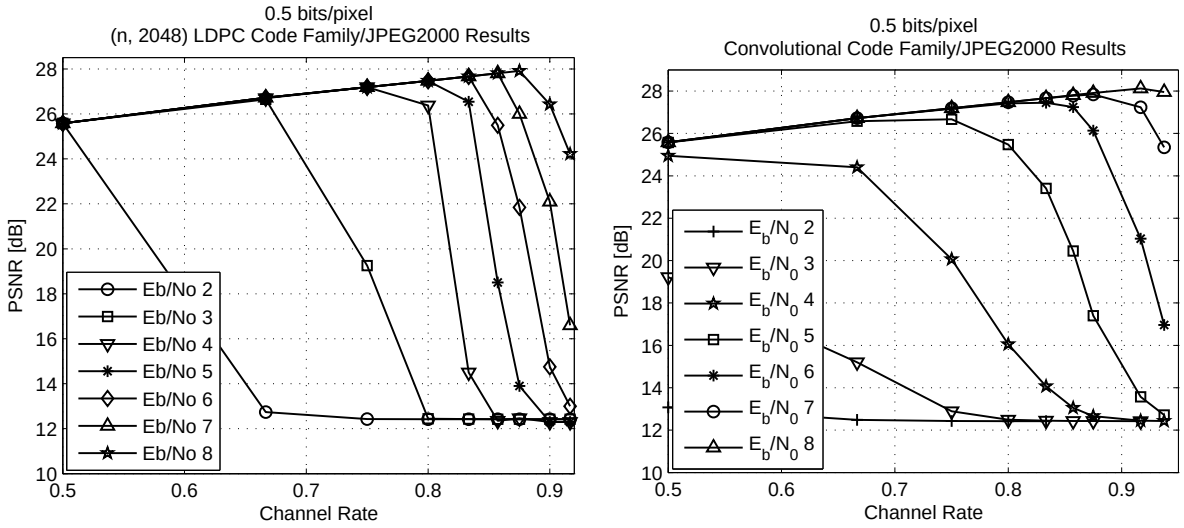


Figure 7. Plots of PSNR as a function of $R_{channel}$ for $R_{total} = 0.5$ bits/pixel. LDPC code results are on the left and convolutional code results are on the right.

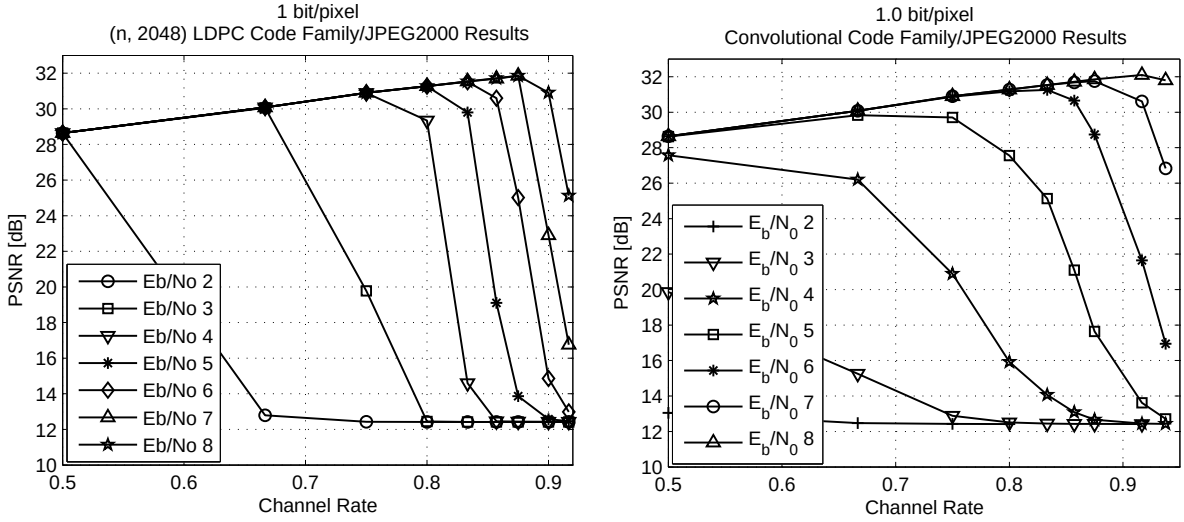


Figure 8. Plots of PSNR as a function of $R_{channel}$ for $R_{total} = 1$ bit/pixel. LDPC code results are on the left and convolutional results code are on the right.

5. CONCLUSION

Our results show how R_{source} and $R_{channel}$ may be optimally (maximum PSNR) selected given a fixed R_{total} (channel bits/pixel) and channel SNR. We also showed the advantage gained by using LDPC codes instead of convolutional codes. Further work could include true joint source/channel decoding in which the two decoders work cooperatively to maximize PSNR. Also, optimal LDPC code design can be considered, as can video transmission.

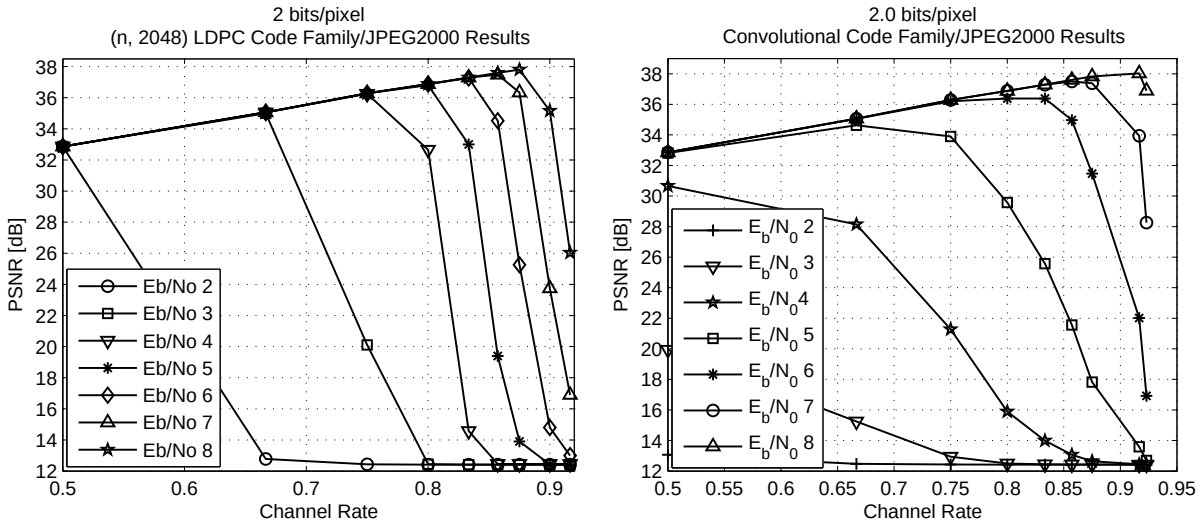


Figure 9. Plots of PSNR as a function of $R_{channel}$ for $R_{total} = 2$ bits/pixel. LDPC code results are on the left and convolutional code results are on the right.

REFERENCES

1. D. S. Taubman and M. W. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards and Practice.* Norwell, MA: Kluwer, 2002.
2. Z. Wu, A. Bilgin and W. M. Marcellin, "Joint Source/Channel Coding for Image Transmission With JPEG2000 Over Memoryless Channels," IEEE Trans. Image Process., vol. 14, no. 8, pp. 1020-1032, Aug. 2005.
3. I. Moccagatta, S. Soudagar, J. Liang and H. Chen, "Error-Resilience Coding in JPEG-2000 and MPEG-4," IEEE J. Sel. Areas Commun., vol. 18, no. 6, pp. 899-914, Jun. 2000.
4. D. Taubman, Kakadu SDK version 6.0. Available at: http://www.kakadusoftware.com.
5. Y. Zhang and W. E. Ryan, "Structured IRA Codes: Performance Analysis and Construction," *IEEE Trans. Commun.*, vol. 55, pp. 837-844, May 2007.
6. Y. Zhang, *Design of Low-Floor Quasi-Cyclic IRA Codes and Their FPGA Decoders*, Ph.D. dissertation, ECE Dept., University of Arizona, May 2007.