LDPC-BASED ITERATIVE JOINT SOURCE/CHANNEL DECODING SCHEME FOR JPEG2000

Lingling Pu, Zhenyu Wu, Ali Bilgin, Michael W. Marcellin, and Bane Vasic
Dept. of Electrical and Computer Engineering
The University of Arizona, Tucson, AZ 85721

ABSTRACT

This paper presents a joint source-channel decoding scheme based on a JPEG2000 source coder and an LDPC channel coder. At the encoder, JPEG2000 is used to perform source coding with certain error resilience (ER) modes, and LDPC codes are used to perform channel coding. At the decoder, after one iteration of LDPC decoding, the output codestream is then decoded by JPEG2000. With the error resilience mode switches on, the source decoder detects the position of the first error within each codeblock of the JPEG2000 codestream. This information is fed back to the channel decoder, and incorporated into the calculation of likelihood values of variable nodes for the next iteration of LDPC decoding. Our results indicate that the proposed method has significant gains over conventional separate channel and source decoding.

KEYWORDS

Joint source channel coding, JPEG2000, iterative decoding, LDPC codes.

INTRODUCTION

Arithmetic codes are gaining attention in source coding schemes such as JPEG2000, which is the latest international image compression standard. Synchronization losses cause arithmetic codes to be sensitive to channel noise. Studies have been ongoing to design systems providing better error protection. It is known that combining channel and source coding can improve overall error control performance [1]. In [2], Turbo codes are applied to compressed images/video coded by different source coding schemes, such as vector quantization, JPEG and MPEG. Redundant source information, or some unique structure in these source codes, are used by the channel decoder. In [3], a soft decoding algorithm is proposed to provide good error-resilience of arithmetic codes. An iterative source-channel decoding structure is also proposed in the spirit of serial turbo codes. Pruning is used to limit the complexity of the soft decoding algorithm. However, it also limits the benefits of iterative joint decoding significantly.

This paper presents a joint source-channel decoding scheme based on a JPEG2000 [4] source coder and an LDPC channel coder. JPEG2000 offers a number of functionalities, including error resilience

tools. These tools combat error propagation in arithmetic coded JPEG2000 codestreams during transmission over noisy channels. As we will demonstrate, they can also provide effective feedback information to a channel decoder. Efficient decoding algorithms for LDPC codes and convenient error resilience tools of JPEG2000 make the complexity of the joint decoding scheme low, yet also provide large gains over separate decoding. Experimental results show significant improvement in PSNR of reconstructed images, and reduction of residual errors under different channel conditions.

DECODING OF LDPC CODES

LDPC codes were invented by Gallagher in 1960 [5], and have good block error correcting performance. These codes did not gain much attention until the mid-1990's. The iterative decoding algorithm provided in [5] was rediscovered as the belief propagation or sum-product algorithm in [6], [7]. To visualize the decoding algorithm for LDPC codes, the parity-check matrix is represented as a bipartite graph (also called a Tanner graph [8]) with two types of nodes. The first subset of nodes is comprised of code bits, and the second subset of nodes is comprised of parity-check equations. An edge between a bit and an equation exists if the bit is involved in the check. For example the Tanner graph representation of a parity check matrix for an LDPC code is shown in Figure 1.

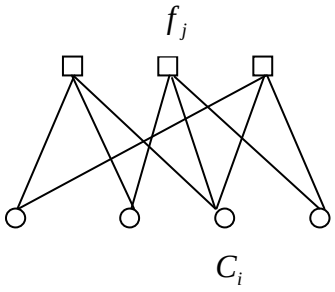

Figure 1. Tanner graph of an LDPC code.

In this figure, the C_i represent variable nodes in a codeword, and the f_j represent check nodes. Variable nodes C_i are connected to f_j according to the rows of a parity check matrix *H* of an LDPC code. Each such row specifies one check equation. The corresponding *H* matrix of Figure 1 is:

$$\mathrm{H} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}. \qquad (1)$$

We are interested in finding the probability $\Pr(C_i = 1 \mid Y, S_i)$, where $\boldsymbol{C}$ is the transmitted codeword, $\boldsymbol{Y}$ is the received word, and S_i is the event that the bits in codeword $\boldsymbol{C}$ satisfy all the parity check equations involving C_i. Gallagher stated the following theorem in [5]:

$$\frac{\Pr(C_i = 0 \mid Y, S_i)}{\Pr(C_i = 1 \mid Y, S_i)} = \frac{1 - P_i}{P_i} \frac{\prod_{j \in N\{i\}} (1 + \prod_{i' \in N\{j\} \backslash i} (1 - 2P_{i'j}))}{\prod_{j \in N\{i\}} (1 - \prod_{i' \in N\{j\} \backslash i} (1 - 2P_{i'j}))}. \qquad (2)$$

Here, P_i is the probability that C_i is 1 given the received digit Y_i, and P_{ij} is the probability that the *i*th bit in the *j*th check equation is 1 given the received digit of that bit. The assumption is that the bits involved in one check equation are statistically independent of each other. Notation $N\{\cdot\}$ denotes the neighborhood of one node in the graph model, i.e., all the other nodes that are connected to that node. Notation $N\{j\}\setminus i$ means the neighborhood of node *j* except node *i*.

The message from variable node *i* to check node *j* is denoted by $q_{ij}(b)$, which is the probability that $C_i = b$ given the extrinsic information from all check nodes other than *j* and the received digit Y_i. The message from check node *j* to variable node *i* is denoted by $r_{ji}(b)$, representing the probability that the *j*th check equation is satisfied given $C_i = b$ and the information from all other variable nodes connected to *j*. Specifically, the check nodes gather information from the variable nodes to compute

$$r_{ji}(0) = \frac{1}{2} + \frac{1}{2}\prod_{i' \in N\{j\}\setminus i}(1 - 2q_{i'j}(1)), \qquad (3)$$

$$r_{ji}(1) = \frac{1}{2} - \frac{1}{2}\prod_{i' \in N\{j\}\setminus i}(1 - 2q_{i'j}(1)). \qquad (4)$$

The variable nodes then gather information from the check nodes to compute

$$q_{ij}(0) = \alpha_{ij}(1 - P_i)\prod_{j' \in N\{i\}\setminus j} r_{j'i}(0), \qquad (5)$$

$$q_{ij}(1) = \alpha_{ij} P_i \prod_{j' \in N\{i\}\setminus j} r_{j'i}(1). \qquad (6)$$

The constant α_{ij} assures $q_{ij}(0) + q_{ij}(1) = 1$. Now the "pseudo-posterior probabilities [7]" $Q_i(b)$, are computed as

$$Q_i(0) = \alpha_i(1 - P_i)\prod_{j' \in N\{i\}} r_{j'i}(0), \qquad (7)$$

$$Q_i(1) = \alpha_i P_i \prod_{j' \in N\{i\}} r_{j'i}(1). \qquad (8)$$

Again, the constant α_i assures $Q_i(0) + Q_i(1) = 1$. The decision $\hat{C}_i = 1$ is made if $Q_i(1) \geq 0.5$. The entire process is repeated in an iterative fashion.

To avoid the many multiplications in the algorithm, log domain computation is adopted. Then $L(C_i)$ denotes the log-APP ratio of variable node C_i, also called the log-likelihood ratio (LLR). Similarly, $L(q_{ij})$ is the LLR of the message q_{ij}, and $L(r_{ji})$ is the LLR of the message r_{ji}.

The decoding procedure starts with initialization. For a binary-input AWGN channel, symbol x_i takes values on {+1, -1}, corresponding to C_i being {0, 1}. The initial LLR of C_i is

$$L(C_i) = \log\frac{\Pr(x_i = +1 \mid y_i)}{\Pr(x_i = -1 \mid y_i)}. \qquad (9)$$

Assume AWGN channel and equal a priori probability,

$$L(C_i) = \log \frac{\Pr(y_i \mid x_i = +1)}{\Pr(y_i \mid x_i = -1)} = 2y_i / \sigma^2 . \qquad (10)$$

Setting the initial $L(q_{ij})$ to be the same as $L(C_i)$, the computation can be obtained straightforwardly by substituting the LLR definitions into Equations (3) – (8):

$$L(r_{ji}) = 2\tanh^{-1}\{\prod_{i' \in N\{j\}\backslash i} \tanh(\frac{1}{2} L(q_{i'j}))\}, \qquad (11)$$

$$L(q_{ij}) = L(C_i) + \sum_{j' \in N\{i\}\backslash j} L(r_{j'i}), \qquad (12)$$

$$L(Q_i) = L(C_i) + \sum_{j \in N\{i\}} L(r_{ji}) . \qquad (13)$$

If $L(Q_i) \geq 0$, the estimated bit $\hat{C}_i = 0$, otherwise $\hat{C}_i = 1$. $L(Q_i)$ can be viewed as the soft value of the binary random variable C_i. The above procedure is repeated until some maximum desired iteration number is reached, or a legal codeword is detected.

The nature of the algorithm implies that more reliable bits have higher soft values, i.e., further away from the threshold (zero). Bit errors occur around the threshold with high probabilities. If a bit is known to be correct, increasing its soft value can help correct errors via the positive message sent from this bit. As mentioned in the introduction, JPEG2000 is able to provide such source information to help the channel decoder. The following subsection provides a brief overview of JPEG2000, including the relevant error resilience tools.

BRIEF REVIEW OF THE JPEG2000 STANDARD

In JPEG2000, an image is optionally divided into non-overlapping rectangular regions, called tiles. The array of samples from one component (if the image has multiple components) which are in the area of a tile is called a tile-component. The wavelet transform is performed on each tile-component, generating subbands of different resolutions depending on the number of levels of wavelet transform. The resulting wavelet subbands are partitioned into a number of different geometric structures. The smallest structure is called a codeblock. Codeblocks are formed by partitioning the subbands. The wavelet coefficients in each codeblock are then quantized. The quantized coefficients in a given codeblock form a sequence of binary arrays by filling each binary array with one bit from each coefficient (from most significant bit to least significant bit). These binary arrays are called bitplanes. Each bitplane is encoded in three passes, referred to as coding passes.

The JPEG2000 codestream is formed by combining the coding passes from different codeblocks. Arithmetic coding is incorporated in the JPEG2000 bit-plane compression process. JPEG2000 provides several error resilience tools, including the arithmetic coder switches RESTART and ERTERM. RESTART causes the arithmetic coder to be restarted at the beginning of each coding pass. In this case, each coding pass has a separate arithmetic codeword segment. When the ERTERM switch is turned on, the source decoder is able to reliably detect when an arithmetic codeword segment is corrupted. If the JPEG2000 codestream is generated using these two mode switches, the decoder can identify that an error has occurred in a given coding pass. When an error occurs in a coding pass, common practice is to discard the current and all future coding passes of the

current codeblock. The decoder then starts decoding the first coding pass in the next codeblock. In this way, bit errors do not propagate from one codeblock to the next. For more detailed description of the standard, readers are referred to [4].

ITERATIVE JOINT SOURCE-CHANNEL DECODING OF JPEG2000 CODESTREAMS

In our work, information on which coding passes in each codeblock are decodable is fed back to the LDPC decoder. To see how this feedback source information can help the channel decoder, examination of the LDPC decoding algorithm is needed. In equation (10), the a priori probabilities of x_i are assumed equal. After source decoding, information on where the first bit error occurs is available to the channel decoder. This actually changes the a priori probabilities of x_i. Modifications are made to re-calculate the $L(C_i)$. The details are described in the following paragraphs.

After JPEG2000 encoding of an image, the resulting codestream is sequentially divided into channel codewords. These channel codewords are mapped into channel symbols as $x_i = (-1)^{C_i}$. The noisy channel will introduce errors into these channel symbols. If noise *n* is AWGN, the received words have symbols $Y_i = x_i + n$. One iteration of LDPC decoding is performed and the output codestream is then decoded by JPEG2000. With the error resilience mode switches on, the correct coding passes are detected in each channel codeword. This source information is passed to the channel decoder.

First suppose that the source decoder performs perfect error detection. For those bits in correct coding passes, $\Pr(x_i = +1) = 1$, if $L(Q_i) \geq 0$ and $\Pr(x_i = -1) = 1$ if $L(Q_i) < 0$. These bits are now known to the channel decoder, and they can help decoding other undetermined bits involved in the same check equations they are in, by sending positive messages. In other words, these known bits are assigned LLRs of $L(C_i) = \infty$, if $L(Q_i) \geq 0$ and $L(C_i) = -\infty$, if $L(Q_i) < 0$. In practice, large values are used instead of infinity.

Now suppose that the source decoder performs correct error-detection with some probability $p_{\text{det}} < 1$. For those bits in correct (believed by the source decoder) coding passes, Equation (10) now becomes

$$\begin{aligned} \tilde{L}(C_i) &= \log \frac{\Pr(y_i \mid x_i = +1)}{\Pr(y_i \mid x_i = -1)} + \log \frac{\Pr(x_i = +1)}{\Pr(x_i = -1)} \\ &= L(C_i) + \operatorname{sgn}\{L(Q_i)\} \cdot \log \frac{p_{\text{det}}}{1 - p_{\text{det}}} \end{aligned}, \qquad (14)$$

where sgn is the sign function.

In summary, we modify the soft values of the variable nodes involved in correct coding passes as:

$$\tilde{L}(C_i) = \begin{cases} L(C_i) - t, & \text{if } L(Q_i) < 0; \\ L(C_i) + t, & \text{if } L(Q_i) \geq 0. \end{cases} \qquad (15)$$

The weighting factor *t* is defined as:

$$t = \left| \log \frac{\Pr(x_i = +1)}{\Pr(x_i = -1)} \right|. \qquad (16)$$

It is important to note that coding passes must be treated sequentially. Due to the context dependent arithmetic coding employed in JPEG2000, a coding pass can only be decoded when all the previous coding passes in the same codeblock are correct. Thus the soft values can be modified for all bits within a codeblock, up to but not including those in the first coding pass containing incorrectly decoded bits. After modification of the appropriate soft values for all code blocks, the next iteration of channel decoding is performed, followed by another round of source decoding. This iterative decoding procedure is repeated until some stopping criterion is met.

EXPERIMENTAL RESULTS

We have used the Kakadu V3.3 implementation of JPEG2000 [9]. In our experiments, a (3648, 3135) LDPC code is selected. In each case where a noisy channel is employed, 1000 simulations are performed. MSE (Mean Square Error) values of decoded images are averaged over the simulations, then the average MSE is converted to PSNR. The 512×512 Lenna image, compressed at 1.0 bits/pixel, is used as the test image.

In previous work [10], we assumed exact information on which coding pass contains the first error within each code block, i.e., no incorrect or mis-detection of corrupt coding passes. The weighting factor is then $t = \log 1/0 = \infty$. Due to the LDPC decoding algorithm, extremely large values of t yield comparable results to those obtained with moderate value of t. In the experiments, t was chosen to be five. We also studied the effect of different codeblock sizes in [10]. By default, Kakadu uses 64×64 codeblocks. In addition to this size, we have also tested our scheme using codeblock sizes of 32×32, 16×16 and 8×8. Different codeblock sizes affect the compression performance as well as the lengths of the coding passes. Smaller codeblock sizes result in shorter coding passes and some reduction in compression efficiency. However, shorter coding passes have better performance in our scheme, since they provide better error localization.

We now consider a more realistic use of the proposed joint decoding method. In what follows, Kakadu is tasked to find the location of the first corrupt coding pass of each codeblock. We begin by studying the reliability of Kakadu in performing this job. To this end, we corrupt Kakadu codestreams with a BSC having error probability ε. During decoding of the compressed image, Kakadu generates an error report about the position of the first error it detects within each code block. By comparing the error report with the truth, the statistical results of error-detection performance are found. The value of ε is selected as the average residual BER after some number of iterations for the channel code used in the joint decoding scheme. Tables 1 and 2 list the results under channel SNRs of 4.2 dB and 4.5 dB, and differing numbers of iterations. The entry “success” in the table is the percentage of cases when the first error in a codeblock is detected as occurring in the correct coding pass. “Delay 1” is the percentage of the errors which are detected one coding pass later than their occurrences. “Delay 2” is the same as “Delay 1” but two coding passes later. Further delays are negligible. “Miss” is the percentage of cases when an existing error goes undetected. More than 90% of the “Miss” cases occur when the error is in the last coding pass of a code block.

From these statistics a suitable weighting factor t can be calculated. Assume Kakadu reports an occurrence of the first error in the jth coding pass. Then from Equation (16), the weighting factor for the (j-1)th coding pass should be calculated as:

Table 1. Statistics for channel SNR = 4.2 dB.

Iteration	$\varepsilon\,(10^{-2})$	Success (%)	Delay 1 (%)	Delay 2 (%)	Miss (%)
5	7.09	98.75	0.59	0.45	0.20
10	4.86	98.64	0.62	0.49	0.23
20	3.53	98.50	0.67	0.52	0.28
40	3.02	98.44	0.66	0.56	0.30

Table 2. Statistics for channel SNR = 4.5 dB.

Iteration	$\varepsilon\,(10^{-4})$	Success (%)	Delay 1 (%)	Delay 2 (%)	Miss (%)
5	9.1	98.21	0.67	0.61	0.40
10	4.0	98.11	0.62	0.60	0.52
20	1.7	97.99	0.60	0.58	0.65
40	1.5	98.05	0.60	0.60	0.59

$$t_{j-1} = \log \frac{\text{Success}}{1 - \text{Success}}. \qquad (17)$$

Because of the fairly high reliability of detection, this weighting factor is large. This value of *t* is computed based only on probability of incorrect decoding. Unfortunately however, if a coding pass is marked as error-free while it actually contains one or more errors, dramatic increases in image distortion can result. Thus, in practice, the value of *t* as computed above results in poor performance. It is desirable to derive an improved expression for *t* that incorporates MSE of the decoded imagery. However such derivation is left for future work.

It is noted above that in most cases, the delayed detections occur within two coding passes of the true error location. In our experiments, we have found that values for *t* of 0.5 and 0.2 work well for the (*j-2*)th and (*j-1*)th coding passes, respectively. As in the error free case, we use *t* = 5 for the *(j-i)*th coding pass for all *i*>2. Results of experiments employing these values of *t* and a codeblock size of 32×32, are listed in Table 3. For this codeblock size, error-free decoding provides a reconstructed image with a PSNR of 39.80 dB.

Table 3. Average PSNR for codeblock size 32×32 after 40 iterations.

Channel SNR	With FB	With FB Ideal	Without FB	ΔPSNR	ΔPSNR Ideal
4.2 (dB)	23.64	24.00	22.75	0.89	1.25
4.3 (dB)	27.57	28.38	26.40	1.17	1.98
4.4 (dB)	31.93	32.65	30.98	0.95	1.57
4.5 (dB)	34.76	35.23	34.01	0.75	1.22

The second column lists the average PSNR using the feedback information. The third column lists the ideal results while we assumed perfect detection as in [10]. The fourth column shows the results

without feedback information, which is the conventional separate decoding method. The last two columns list the performance gains. The gains obtained have the same tendency as in previous work. But, as expected, gains are not as large as those in the ideal case, when perfect error detection was assumed. The ideal case provides an upper bound for the performance of a practical joint decoding scheme. With more complex weighting schemes, as hinted at above, the performance of practical joint decoding may be closer to the ideal case.

The convergence rates of the two schemes are illustrated in Figure 2 and Figure 3 for two different channel SNRs. These figures clearly show how the joint decoding method accelerates convergence of correct decoding.

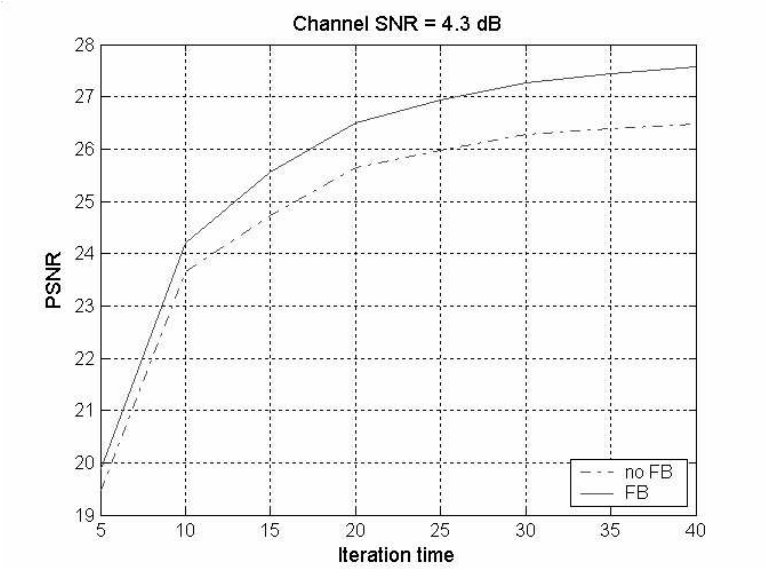

Figure 2. PSNR vs. iteration number for channel SNR = 4.3 dB.

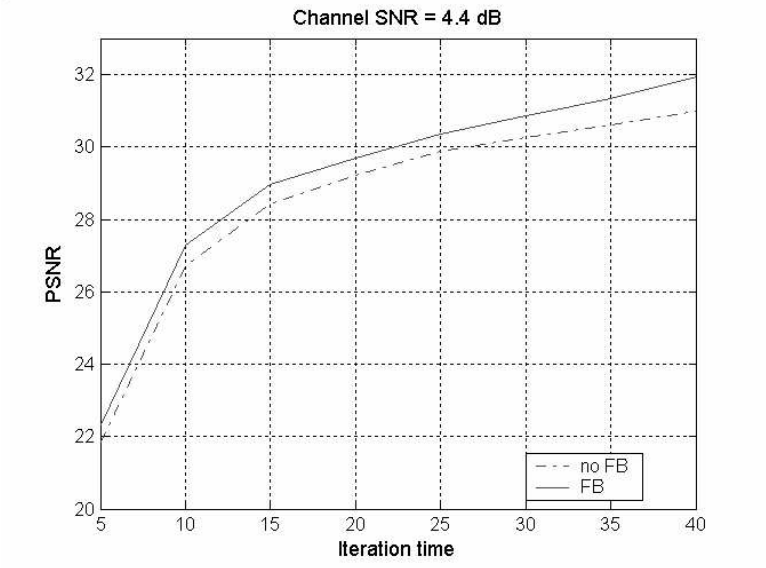

Figure 3. PSNR vs. iteration number for channel SNR = 4.4 dB.

CONCLUSION

This work extends our previous work to a more practical scenario. From the experimental results, it is clear that the proposed joint iterative decoding method can improve overall system performance. For a given PSNR requirement, the joint iterative decoding method requires less iterations than the separate decoding method. For a given channel SNR condition, the joint iterative decoding method can improve the quality of the reconstructed images. The use of our joint decoding scheme increases the operational range of the communication system.

REFERENCE

[1] Hagenauer, J., "Source-controlled channel decoding", IEEE Transactions on Communications, Vol. 43, September 1995, pp. 2449—2457.

[2] Peng, Z., Huang, Y. and Costello, D.J., "Turbo codes for image transmission—A joint channel and source decoding approach", IEEE Journal on Selected Areas in Communications, Vol. 18, No. 6, June 2000, pp. 868—879.

[3] Guionnet ,T. and Guillemot , C., " Soft decoding and synchronization of arithmetic codes: Application to image transmission over noisy channels", IEEE Transaction on Image Processing, Vol. 12, No. 12, December 2003, pp. 1599—1609.

[4] Taubman, D.S. and Marcellin, M.W., JPEG2000: Image compression fundamentals, standards and practice, Kluwer Academic Publishers, 2002.

[5] Gallagher, R. G., "Low-density parity-check codes", IRE Transactions on Information Theory, Vol. IT-8, January 1962, pp. 21—28.

[6] MacKay, D.J.C. and Neal, R., "Good codes based on very sparse matrices", 5th IMA Conference Cryptography and Coding, Berlin, Germany, 1995.

[7] MacKay, D.J.C., "Good Error-Correcting Codes Based on Very Sparse Matrices", IEEE Transactions on Information Theory, Vol. 45, March 1999, pp. 399—431.

[8] Tanner, R. M., "A recursive approach to low-complexity codes", IEEE Transactions on Information Theory, Vol. 27, 1981, pp. 533—547.

[9] Available online: www.kakadusoftware.com.

[10] Pu, L., Wu, Z., Bilgin, A., Marcellin, M.W. and Vasic, B., "Iterative Joint Source-channel Decoding for JPEG2000", (invited paper), in Proceedings of 2003 Asilomar Conference on Signals, Systems, and Computers, November 2003.