Iterative Joint Source/Channel Decoding 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*— We propose a framework for iterative joint source-channel decoding of JPEG2000 codestreams. 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. During decoding, the source decoder uses the ER modes to identify corrupt sections of the codestream and provides this information to the channel decoder. The decoding is carried out jointly in an iterative fashion. Our results indicate that the proposed method improves the convergence rate as well as the overall system performance.**

I. Introduction

It has been observed 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 utilized by the channel decoder.

This paper presents a joint source-channel decoding scheme similar to those in [2], but based on a JPEG2000 [3] source coder and an LDPC channel coder. JPEG2000 is the latest international image compression standard, and offers a number of functionalities, including error resilience tools. These tools combat error propagation in the JPEG2000 codestreams during transmission over noisy channels. As we will demonstrate, they can also provide effective feedback information to the channel decoder. Experimental results show significant improvement in PSNR of reconstructed images and in reduction of residual errors under different channel conditions.

The paper is organized as follows: Section II presents the proposed joint source-channel decoding scheme. Section III includes experimental results. Section IV concludes the paper.

II. Iterative Joint Source-Channel Decoding of JPEG2000 Codestreams

LDPC codes were invented by Gallager in 1960 [4], 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 [4] was rediscovered as the belief propagation or sum-product algorithm in [5], [6]. Before we describe our joint decoding scheme, we first present a high level description of this LDPC decoding algorithm.

The Tanner graph [7] representation of a parity check matrix for an LDPC code is shown in Figure 1. In Fig. 1, the C_i represent variable nodes in a codeword $\mathbf{C}$, and the f_j represent check nodes. Variable nodes C_i are connected to f_j according

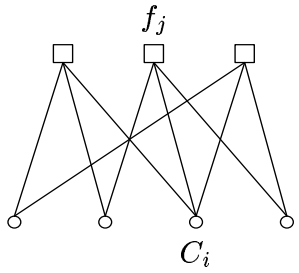

Fig. 1. Tanner graph of an LDPC code

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 Fig. 1 is: $\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$. We are interested in finding the probability $\Pr(C_i = 1|\mathbf{Y}, \mathbf{S_i})$, where $\mathbf{C}$ is the transmitted codeword, $\mathbf{Y}$ is the received word, and $\mathbf{S_i}$ is the event that the bits in codeword $\mathbf{C}$ satisfy all the parity check equations involving C_i. Gallager stated the following theorem in [4]:

$$\frac{\Pr(C_i = 0|\mathbf{Y}, \mathbf{S_i})}{\Pr(C_i = 1|\mathbf{Y}, \mathbf{S_i})} = \frac{1 - P_i}{P_i} \frac{\prod_{j \in \mathcal{N}\{i\}} (1 + \prod_{i' \in \mathcal{N}\{j\} \backslash i} (1 - 2P_{i'j}))}{\prod_{j \in \mathcal{N}\{i\}} (1 - \prod_{i' \in \mathcal{N}\{j\} \backslash i} (1 - 2P_{i'j}))} \quad (1)$$

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 ith bit in the jth 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 $\mathcal{N}\{\cdot\}$ denotes the neighborhood of one node in the graph model, *i.e.*, all the other nodes that are connected to that node. Notation $\mathcal{N}\{j\}\backslash i$ 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 jth 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

$$r_{ji}(0) = \frac{1}{2} + \frac{1}{2} \prod_{i' \in \mathcal{N}\{j\}\backslash i} (1 - 2q_{i'j}(1)), \quad (2)$$

$$r_{ji}(1) = \frac{1}{2} - \frac{1}{2} \prod_{i' \in \mathcal{N}\{j\}\backslash i} (1 - 2q_{i'j}(1)). \quad (3)$$

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

$$q_{ij}(0) = \alpha_{ij}(1 - P_i) \prod_{j' \in \mathcal{N}\{i\}\backslash j} r_{j'i}(0), \quad (4)$$

$$q_{ij}(1) = \alpha_{ij} P_i \prod_{j' \in \mathcal{N}\{i\}\backslash j} r_{j'i}(1). \quad (5)$$

The constant α_{ij} assures $q_{ij}(0) + q_{ij}(1) = 1$. Now we can compute the "pseudoposterior probabilities" [6] $Q_i(b)$.

$$Q_i(0) = \alpha_i(1 - P_i) \prod_{j' \in \mathcal{N}\{i\}} r_{j'i}(0), \quad (6)$$

$$Q_i(1) = \alpha_i P_i \prod_{j' \in \mathcal{N}\{i\}} r_{j'i}(1). \quad (7)$$

Again, 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:

$$L(C_i) = \log \frac{\Pr(x_i = +1|y_i)}{\Pr(x_i = -1|y_i)}. \quad (8)$$

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 2 – 7.

$$L(r_{ji}) = 2 \tanh^{-1}\{ \prod_{i' \in N\{j\}\backslash i} \tanh(\frac{1}{2} L(q_{i'j})\}. \quad (9)$$

$$L(q_{ij}) = L(c_i) + \sum_{j' \in N\{i\}\backslash j} L(r_{j'i}). \quad (10)$$

$$L(Q_i) = L(c_i) + \sum_{j \in N\{i\}} L(r_{ji}). \quad (11)$$

If $L(Q_i) > 0$, 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 paragraph provides a brief overview of JPEG2000, including the relevant error resilience tools.

In JPEG2000, an image is 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 then 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 [3]. 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.

In our work, information on which coding passes in each codeblock are decodable is fed back to the LDPC decoder. Aided by such feedback source information, increasing the soft values of the variable nodes representing those correct coding passes can reduce the bit error rate and accelerate the iterative decoding procedure. The details of our scheme are described as follows:

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.

The source information is passed to the channel decoder and used to modify the soft values of the variable nodes involved in correct coding passes as:

$$L(c_i) = \begin{cases} L(c_i) - t, & \text{if} \quad L(Q_i) < 0; \\ L(c_i) + t, & \text{if} \quad L(Q_i) \geq 0. \end{cases} \tag{12}$$

In the expression above, t is a large positive weighting factor. Due to the LDPC decoding algorithm, extremely large values of t yield comparable results to those obtained with moderate values of t. In the experiments, t is chosen to be 5.

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 soft values for all code blocks, the next iteration of channel decoding is performed. This iterative decoding procedure is repeated until some stopping criterion is met.

III. Experimental Results

An important factor that affects overall performance is the codeblock size used during the creation of the JPEG2000 codestream. We have used the KakaduV3.3 implementation of JPEG2000 [8]. By default, Kakadu uses 64×64 codeblocks. In addition to this size, we have also tested our scheme using codeblocks size 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. From the point view of joint iterative decoding, shorter coding passes are preferable, since they provide better localization. Shorter coding passes result in more coding passes in one channel codeword. These short error-free coding passes can participate in accelerating bit error corrections in their channel codewords.

In our experiments, a (3648, 3135) LDPC code is selected. In each case where a noisy channel is employed, 1000 simulations are performed. The 512×512 Lenna image compressed at 1.0 bits/pixel is used as the test image. As mentioned above, different codeblock sizes are employed to form the JPEG2000 codestream. For a codeblock size of 64×64, error-free decoding provides a reconstructed image with a PSNR of 40.24 dB; for codeblock sizes of 32×32, 16×16 and 8×8, error-free decoding yields PSNR values of 39.80 dB, 39.07 dB, and 37.87 dB, respectively.

Tables I – III list the average PSNR results of our joint decoding method versus the usual separate decoding method under different channel conditions, *i.e.*, when the channel SNR equals 4.2, 4.3 and 4.4 dB, respectively. A maximum number of 40 iterations are carried out for each method. From these tables, we can see the effect of the codeblock size. As expected, ΔPSNR increases when codeblock size decreases. It can also be observed that under lower channel SNR conditions, ΔPSNR is generally larger. However eventually, in the very low channel SNR case, the PSNR results from both methods are both extremely low, close to zero, and the gain between the two methods disappears. In Table I, the gain between the two methods for a codeblock size of 32×32 (1.8701 dB) is close to that in Table II (1.8423 dB). Decreasing the channel SNR further results in immediate and significant damage to the performance under both methods for this codeblock size. For the smaller codeblock sizes, ΔPSNR continues to increase somewhat as channel SNR is decreased, before eventually falls precipitously.

The distribution of the gains (*i.e.*, ΔPSNRs) between the two methods can also be illustrative. Fig. 2 – 4 show the

TABLE I
AVERAGE PSNR FOR LENNA AFTER 40 ITERATIONS FOR A 4.2 DB AWGN CHANNEL

Channel SNR	4.2 dB		
	With FB	No FB	Δ PSNR
CB 32×32	30.1941	28.3240	1.8701
CB 16×16	32.0528	28.6444	3.4083
CB 8×8	36.5015	28.8784	7.6231

TABLE II
AVERAGE PSNR FOR LENNA AFTER 40 ITERATIONS FOR A 4.3 DB AWGN CHANNEL

Channel SNR	4.3 dB		
	With FB	No FB	Δ PSNR
CB 32×32	35.0525	33.2102	1.8423
CB 16×16	36.5079	33.3112	3.1968
CB 8×8	37.5510	33.0091	4.5419

TABLE III
AVERAGE PSNR FOR LENNA AFTER 40 ITERATIONS FOR A 4.4 DB AWGN CHANNEL

Channel SNR	4.4 dB		
	With FB	No FB	Δ PSNR
CB 32×32	38.0060	37.0637	0.9422
CB 16×16	38.4179	36.9323	1.4856
CB 8×8	37.8090	36.0044	1.8046

TABLE IV
RESIDUAL BER COMPARISON BETWEEN TWO METHODS

Channel SNR (dB)	4.2	4.3	4.4
With FB 5	4.8290×10^{-3}	2.6189×10^{-3}	1.2654×10^{-3}
No FB 5	5.9573×10^{-3}	3.4468×10^{-3}	1.7482×10^{-3}
With FB 40	6.6024×10^{-4}	1.7622×10^{-4}	4.9798×10^{-5}
No FB 40	2.3361×10^{-3}	9.3588×10^{-4}	3.1665×10^{-4}

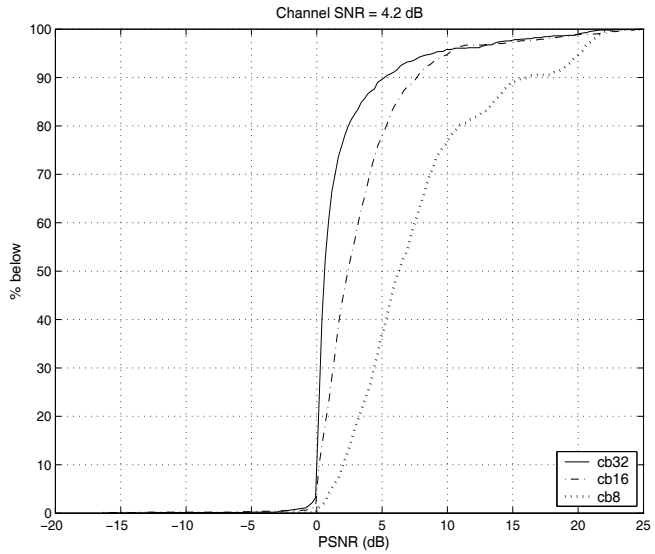

Fig. 2. CDF of ΔPSNR for channel SNR = 4.2 dB.

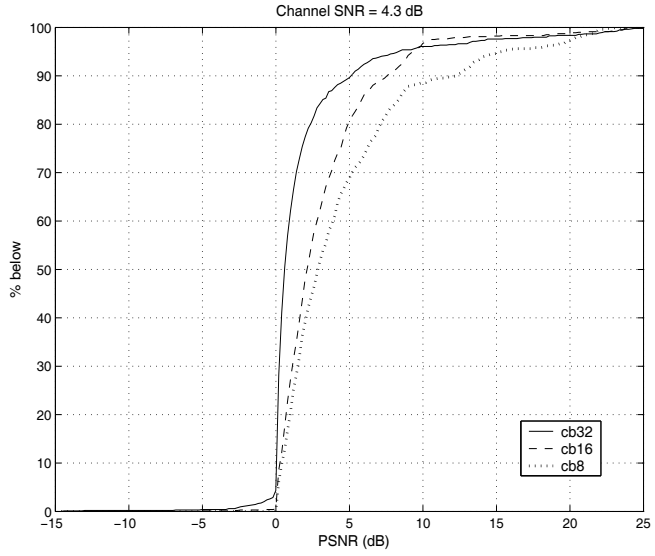

Fig. 3. CDF of ΔPSNR for channel SNR = 4.3 dB.

cumulative distribution function (CDF) of ΔPSNR under different channel conditions. For example, it can be seen in Fig. 3 that the proposed method results in PSNR gains larger than 5 dB in roughly 10% of the simulations for a codeblock size of 32×32. However for codeblock sizes of 16×16 and 8×8, PSNR gains larger than 5 dB occur in roughly 20% and 30% of the simulations, respectively.

In Fig. 2 – 4, there are cases when ΔPSNR is negative. From Table IV, we can clearly see that the joint decoding method improves the overall error correction performance. However, the method does not determine which particular bits are corrected. The weighting applied to the variable nodes causes changes in the relationship between variable

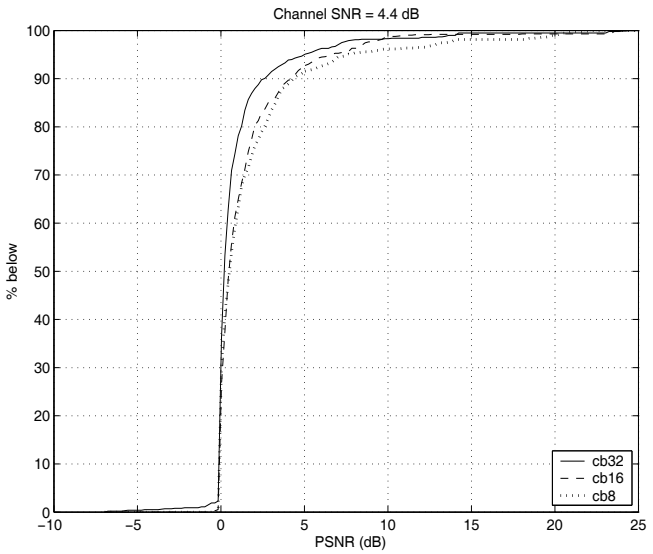

Fig. 4. CDF of ΔPSNR for channel SNR = 4.4 dB.

TABLE V
PERCENTAGES BELOW ZERO IN CDF OF ΔPSNR

Channel SNR	4.2 dB	4.3 dB	4.4 dB
CB 32×32	4.13%	4.52%	3.20%
CB 16×16	1.84%	0.50%	1.05%
CB 8×8	0.19%	0.00%	0.00%

nodes and check nodes. The overall tendency is to correct erroneous bits. But, locally, some unexpected miscorrections may occur. Thus, some bits may be decoded correctly after the final iteration by the separate decoding method, while decoded incorrectly by the joint decoding method. This is true, even though the number of the residual error bits by the latter method is significantly less than that by the former method. This, together with the fact that different source bits are of different importance, can explain the negative ΔPSNR values on the distribution figures. From the figures, we note that the percentage of negative ΔPSNRs is related to the codeblock size. Table V gives the percentage of negative cases for various parameter choices. The smaller the codeblock size, the smaller the percentage of negative ΔPSNRs. This result is straightforward from the analysis in previous sections due to the stronger error correction ability and less residual errors associated with smaller codeblock sizes.

When designing a system to transmit images, the channel conditions are often assumed to be known. In many practical situations, actual channel statistics are not known, or they may vary in time, such as Internet or wireless channels. Such channel mismatch causes performance loss, and a channel SNR decrease is more harmful than a channel SNR increase. From the data in previous tables, it can be seen that the channel code chosen in this paper is not strong enough on its own to protect the source codestream to a high level

TABLE VI
AVERAGE PSNR FOR LENA 512×512 AFTER 40 ITERATIONS

Channel SNR	4.5 dB		
	With FB	No FB	Δ PSNR
CB 32×32	39.3080	38.8790	0.4290
CB 16×16	38.9274	38.4678	0.4596
CB 8×8	37.8703	37.3792	0.4912

of reliability. Performance under these conditions can be considered as mismatched. The proposed iterative decoding method always outperforms the separate decoding method in the mismatched case. On the other hand, if the channel SNR is increased further to achieve almost error-free decoding, system performance can be considered as the matched case. Table VI lists the average PSNR results of the two methods after 40 iterations when the channel SNR is 4.5 dB. We note that in Table VI, for a codeblock size of 8×8, our joint decoding method is almost always able to decode the source codestream error-free (99.9%) while the separate decoding scheme is not (70.7%). The joint decoding scheme under that condition can be considered as the matched case. From this, it can be concluded that a weaker channel code can be adopted in the joint decoding scheme as compared to the separate decoding scheme, for a given channel SNR.

Finally, we examine the convergence rate of the two schemes. This can be shown in the plot of PSNR vs. iteration number, as in Fig. 5 – 6. From these figures, it is clear that the joint decoding method accelerates convergence. For example, in Fig. 5, when the channel SNR is 4.3 dB and the codeblock size is 32×32, the joint decoding method takes 16 iterations to reach a PSNR of 33 dB, which is 9 iterations less than the separate decoding method.

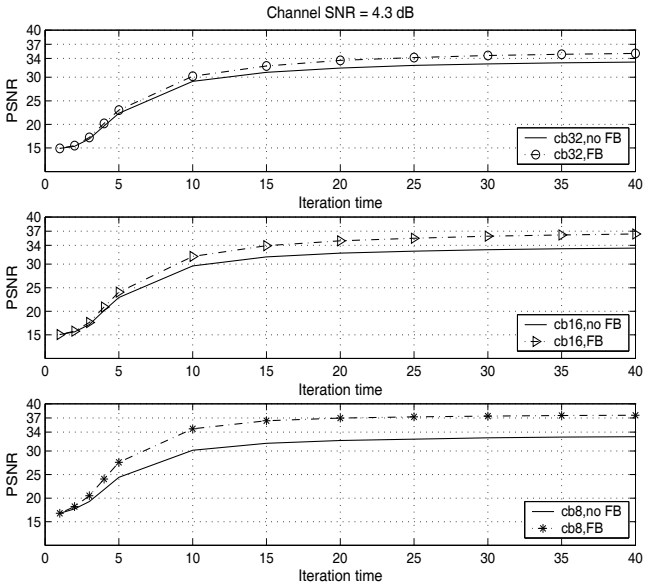

Fig. 5. PSNR vs. iteration number for channel SNR=4.3 dB.

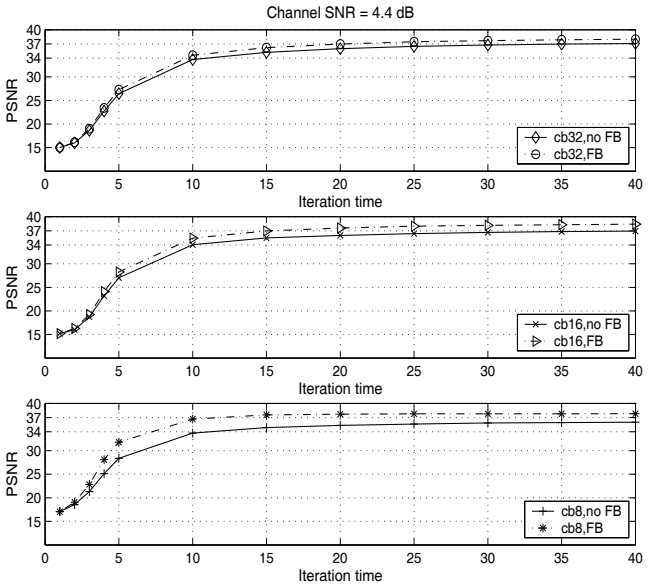

Fig. 6. PSNR vs. iteration number for channel SNR=4.4 dB.

IV. CONCLUSION

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 fewer 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. In channel mismatch cases, the proposed joint decoding method is less sensitive to noise than separate decoding. The use of our joint decoding scheme increases the operational range of the communication system.

REFERENCES

[1] J. Hagenauer, "Source-controlled channel decoding," *IEEE Transactions on Communications*, vol. 43, pp. 2449–2457, Sep. 1995.

[2] Z. Peng, Y. Huang, and D. Costello, "Turbo codes for image transmission–a joint channel and source decoding approach," *IEEE Journal on Selected Areas in Communcations*, vol. 18, pp. 868–879, Jun. 2000.

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

[4] R. G. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[5] D. MacKay and R. Neal, "Good codes based on very sparse matrices," in *Proceedings, 5th IMA Conference Cryptography and coding*, (Berlin, Germany), 1995. Springer Lecture Notes in Computer Science.

[6] D. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, pp. 399–431, Mar. 1999.

[7] R. M. Tanner, "A recursive approach to low-complexity codes," *IEEE Transactions on Information Theory*, vol. 27, pp. 533–547, 1981.

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