

# ERROR RESILIENT JPEG2000 DECODING FOR WIRELESS APPLICATIONS

Simone Zezza, Maurizio Martina, Guido Masera

Politecnico di Torino  
VLSI Laboratory  
10129 Torino, Italy

Saeid Nooshabadi

Gwangju Inst. of Sci. and Tech.  
Dept. of Info. and Comm.  
Gwangju, 500-712, Republic of Korea

## ABSTRACT

To improve the JPEG2000 compression standard error resiliency in the wireless environment, the use of ternary MQ arithmetic coders/decoders that are based on the concept of forbidden symbol has been proposed. This paper presents two ternary MQ based techniques to reduce both the computational complexity and the memory requirement during the decoding process, with no or little degradation in the PSNR.

**Index Terms**— Arithmetic coding, JPEG2000, wireless multimedia communications

## 1. INTRODUCTION

The JPEG2000 standard has excellent code compression efficiency. However, suffers from potential errors in the compressed bit-stream [1, 2], making it unattractive in a wireless environment. Parts 1, 2 and 3 of the standard provides for error resilience tools that are based on the insertion of markers at the code stream level, and on the termination of the MQ arithmetic coder after each coding pass at the entropy coding level [3]. These tools are recognized to guarantee a sufficient degree of protection when the transmission conditions are not severe, but they exhibit unsatisfactory performance in presence of difficult channel conditions. The JPWL ad-hoc group is proposing several tools to increase JPEG2000 bit-stream error resilience. Some of these tools, based on the Reed-Solomon codes, are employed not only to yield unequal error protection and graceful degradation of the data concerning the image (or frame) inside the bit-stream [1], [4], but also to preserve JPEG2000 headers. Other techniques are based on the use of sequential arithmetic coding [5, 6]. In [5] soft re-synchronization markers are employed, whereas in [6], the sequential decoding is achieved through the use of a ternary arithmetic coder scheme employing a forbidden symbol.

The work in [6] proposes a *maximum likelihood* (ML) and a *maximum a-posteriori* (MAP) context-based decoder, that are specifically tailored to JPEG2000 arithmetic coder, and are able to carry out both hard and soft decoding of a corrupted code stream. The error resilience tool proposed in [6] exhibits very high performance, in terms of PSNR, when compared to the standard JPEG2000 decoding. However, this

scheme is extremely demanding both in terms of amount of memory and average time required to decode a single frame.

The aim of this paper is to reduce the memory requirement and the computational complexity of the scheme proposed in [6] through the application of two simplifying algorithms, without introducing any significant performance degradation.

## 2. JPEG2000 OVERVIEW

In JPEG2000 codeblocks are processed by the Embedded Block Coding with Optimized Truncation (EBCOT) algorithm [3]. Tier 1 of EBCOT embodies the bit-plane and the arithmetic coders. The bit-plane coder is responsible for the context formation, required to find an appropriate model for the coefficient context. The output of the context formation is a coded coefficient (decision bit)  $d$  and a context  $x$ . The arithmetic coder is responsible for (dec)compressing the data.

JPEG2000 employs the MQ coder [7] that encodes the decision bits  $\mathbf{D} = \{d_0, \dots, d_{L-1}\}$ , which belong to a codeblock in the DWT domain, into a compressed code string  $\mathbf{C}$  of length  $N$ . Each decision bit  $d_i \in \mathbf{D}$  is accompanied by its own context label  $x_i \in \mathbf{X} = \{x_0, \dots, x_{L-1}\}$  where  $L$  is the number of decision bits in a code-block.

## 3. ERROR CORRECTING ALGORITHM

In order to implement the error correcting algorithm in [6], the MQ encoder and decoder have been modified introducing a *forbidden symbol* in the binary alphabet. In the following, the modified MQ will be referred to as MQF.

Assigning a small non zero probability of occurrence to the *forbidden symbol*, the decoder can detect an error in the compressed bit-stream [8] in a continuous manner. In fact, when an error occurs, after a certain delay, the arithmetic decoder selects the interval associated with the *forbidden symbol*. The amount of delay depends on the probability assigned to the *forbidden symbol* [6], [8].

Fig. 1 shows the sub-division scheme for MQ with a sub-interval allocated for the *forbidden symbol*. A forbidden region  $\mu$  with the probability  $Q_f$  has been introduced at the base of the probability interval. The probability of the least probable symbol (LPS) is represented by  $Q_e$ . The presence of  $Q_f$  requires a minor modification of the encoding procedure. At each encoding step,  $Q_f$  must be added to the code string

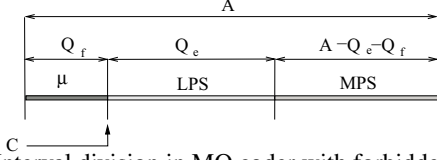


Fig. 1. Interval division in MQ coder with forbidden symbol

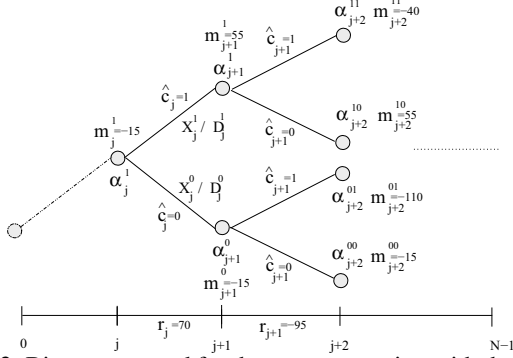


Fig. 2. Binary tree used for the error correction with the modified MQ decoder. Metrics  $m_j$  are updated according to algorithm SMA introduced in Sec. 4.2

$\mathbf{C}$ , in order to skip the forbidden interval. The introduction of the *forbidden symbol* allows for error detection at the decoder, which is able to stop decoding as soon as the received code  $\mathbf{C}$  falls into the forbidden region. This event can be easily monitored by checking the inequality condition  $\mathbf{C} < Q_f$ .

The continuous error detection capability of MQF makes it possible to design a sequential decoder, able to exploit the *forbidden symbol* redundancy for error correction [6].

Consider the  $j^{\text{th}}$  decoding step of the MQ decoder. During the decoding of the single bit  $c_j$ ,  $j = 0, \dots, N-1$  from the code string  $\mathbf{C}$  a variable  $K_j$  number of decision bits  $\mathbf{D}_j = \{d_0^j, \dots, d_{K_j-1}^j\}$ , are decoded.  $\mathbf{D}_j$  is accompanied by the corresponding context  $\mathbf{X}_j = \{x_0^j, \dots, x_{K_j-1}^j\}$ .

Let us suppose that the MQF decoder receives a code string  $\mathbf{R} = \{r_0, \dots, r_{N-1}\}$ , transmitted across a noisy channel. The samples  $r_j$ ,  $j = 0, \dots, N-1$  represent the channel output and can be either hard decoded bits or soft values. In the presence of noise, the decoder goal is to estimate the best code string  $\hat{\mathbf{C}} = \{\hat{c}_0, \dots, \hat{c}_{N-1}\}$  that does not cause a *forbidden symbol* detection. Consequently the decoding task can be formulated as a constrained *MAP* estimation problem:

$$\hat{\mathbf{C}} = \arg \max_{\mathbf{C}} P(\mathbf{C}|\mathbf{R}) \quad (1)$$

where  $P(\mathbf{C}|\mathbf{R})$  can be expressed in the additive form as [6]:

$$\log P(\mathbf{C}|\mathbf{R}) \propto \sum_{j=0}^{N-1} m_j \quad (2)$$

with

$$\begin{aligned} m_j^{MAP} &= \log P(r_j|c_j) + \log(\mathbf{D}_j) \\ m_j^{ML} &= \log P(r_j|c_j) \propto -\frac{(r_j - c_j)^2}{2\sigma^2} \end{aligned} \quad (3)$$

where  $\sigma^2$  is the noise variance.

We use the simpler ML criterion as its performance is similar to MAP criterion [6]. The best candidate  $\hat{\mathbf{C}}$  can be

obtained by means of a sequential search along the decoding tree, which represents all the possible code strings of length  $N$  as shown in Fig. 2. In the illustration of Fig. 2, the bit index  $j = 0, \dots, N-1$  is reported on the horizontal axis, and each node  $\alpha_j$  corresponds to a certain MQF decoding state at step  $j$ . Moreover, the node stores the value of the accumulated *a-posteriori* probability  $= \sum m_j^{ML}$ .

Consider a given node  $j$  in the tree: the two departing transitions are labeled with the possible estimates  $\hat{c}_j = 0$  and  $\hat{c}_j = 1$ ; given the node  $\alpha_j$  and the estimated bit  $\hat{c}_j$ , a decoding step is performed along each branch, thus obtaining a variable number of decoded bits  $\mathbf{D}_j$ , according to the contexts  $\mathbf{X}_j$  requested by EBCOT decoding passes. If a *forbidden symbol* is detected, the branch is pruned; otherwise the final node  $\alpha_{j+1}$ , along with the updated *a-posteriori* probability, is stored in the subsequent node. Clearly, the decoder goal is to pick up the survivor path  $\hat{\mathbf{C}}$  with the maximum *a-posteriori* probability. However, for the typical values of the block length  $N$ , it is infeasible to explore the whole binary tree, which grows exponentially with the bit index  $j$ ; therefore the suboptimal M-algorithm has been used in [6]. At each stage, the M-algorithm limits the search space to the  $M$  nodes that exhibit the highest *a-posteriori* probabilities.

#### 4. EFFICIENT IMPLEMENTATION OF AN ML DECODER

Table 1 summarizes the results for the ML based error correcting M-algorithm proposed in [6]. The case  $M = 0$  refers to direct decoding without error correction and it is reported for comparison purposes, whereas  $M = 8$  has been selected as it assures optimal performance [6].

Next we present two novel techniques to reduce the computational complexity and memory requirement of M-algorithm in [6]; *Variable M-Algorithm* (VMA) and *Simplified M-Algorithm* (SMA). The experimental results are worked out using the same conditions assumed in [6]. All the results reported below have been obtained running the modified VM8.6 model on a Centrino Duo 1.8 GHz machine, equipped with 2 GBytes of RAM. The *valgrind tool* [9] was used to estimated the memory requirements.

##### 4.1. Technique 1: Variable M-algorithm (VMA)

After the occurrence of an error in the code stream a variable number of symbols is processed before the error is detected. For a given  $Q_f$ , the probability that  $n$  symbols are processed before the detection of the error is given by [8]:

$$\delta = (1 - Q_f)^n \quad (4)$$

The probability expression in (4) rapidly decays with  $n$ . We use this fact to argue that using the optimum value  $M = 8$ , across all the corrupt compressed code-blocks in a frame, is too conservative and results an unnecessary processing time and memory overhead.

**Table 1.** Average decoding time for a single frame, the corresponding PSNR performance and the memory usage for the cases of decoding without error correction and M-algorithm [6]. (0.5 bpp rate, QCIF (176 × 144) Foreman sequence and  $Q_f = 0.02$ ).

Alg.	M	BER=10 <sup>-3</sup>			BER=10 <sup>-4</sup>		
		Time(ms)	PSNR(dB)	Mem.(kByte)	Time(ms)	PSNR(dB)	Mem.(kByte)
JPEG 2000 Part I	0	2	23.7	160	2	26.8	160
[6]	8	106	29.7	541	37	30.2	420

**Table 2.** Average decoding time for a single frame, the corresponding PSNR and the memory usage for the VMA technique (0.5 bpp rate, QCIF (176 × 144) Foreman sequence and  $Q_f = 0.02$ ).

Alg.	M	M*	M**	BER=10 <sup>-3</sup>			BER=10 <sup>-4</sup>		
				Time(ms)	PSNR(dB)	Mem.(kByte)	Time(ms)	PSNR(dB)	Mem.(kByte)
VM	2	4	4	44.2	29.1	305	9.4	30.1	255
VM	2	4	8	46	29.7	336	11.0	30.2	261
VM	2	8	8	63.5	29.7	436	10.3	30.2	295
VM	4	8	8	58.4	29.7	365	16.9	30.1	334
[6]	8	8	8	106	29.7	541	37	30.2	420

In our modified Variable M-algorithm, we follow a different strategy. We start with a smaller value for  $M$  and progressively apply larger values  $M^{**} > M^* > M$  only for code-blocks that cannot be successfully corrected with  $M$ .

Table 2 presents the simulation results for different choices of  $M$ ,  $M^*$  and  $M^{**}$ . Comparing these results with those in [6] it can be observed that for BER=10<sup>-3</sup>, the particular choice of  $M = 2$ ,  $M^* = 4$  and  $M^{**} = 8$  achieves a reduction of 57% in the decoding time and 37% in the memory usage, without any loss of PSNR performance. The choice of  $M = 2$ ,  $M^* = M^{**} = 4$  allows for a reduction of 58% in the processing time and 44% in the memory usage, at the expense of only 0.6 dB reduction in the PSNR.

The results reported for BER=10<sup>-4</sup> are even more impressive; with the choice of  $M = 2$  and  $M^* = 4$ , VMA achieves a 70% saving in the computation time and 39% in the memory usage with no PSNR degradation.

#### 4.2. Technique 2: Simplified M-algorithm (SMA)

In order to select the best  $M$  paths, at each step  $j$ , the M-algorithm requires to sort the accumulated metrics  $\sum m_j$ . However, the computationally expensive sorting step can be avoided through a modification of the AWGN *a-posteriori* ML criterion in [6]. The additive metric (3) can be re-written as:

$$m_j \begin{cases} m_j^1 = m_{j-1} - \frac{(r_j-1)^2}{2\cdot\sigma^2} & \text{if } \hat{c}_j = 1 \\ m_j^0 = m_{j-1} - \frac{(r_j+1)^2}{2\cdot\sigma^2} & \text{if } \hat{c}_j = 0 \end{cases} \quad (5)$$

Recognizing the scaling and translational invariance of the ML criterion, (5) simplifies to:

$$m_j \begin{cases} m_j^1 = m_{j-1} + r_j & \text{if } \hat{c}_j = 1 \\ m_j^0 = m_{j-1} & \text{if } \hat{c}_j = 0 \end{cases} \quad (6)$$

The following describes how metric (6) can replace the sorting function required by the M-algorithm.

Starting from a generic root node  $\alpha_j^1$ , of the tree depicted in Fig. 2, we perform an extension to two new nodes:  $\alpha_{j+1}^1$  and  $\alpha_{j+1}^0$ . The former is the leaf node obtained from the root node  $\alpha_j^1$  decoding  $\hat{c}_j = 1$  (*one extension*) and the latter is that obtained decoding  $\hat{c}_j = 0$  (*zero extension*). If  $m_j^1$  is the accumulated metric of the root node, the metric of  $\alpha_{j+1}^1$  can be computed using the first part of (6) i.e.  $m_{j+1}^1 = m_j^1 + r_j$ , where  $r_j$  is the soft value at the channel output associated with the transition  $j \rightarrow j+1$ . The metric of the node  $\alpha_{j+1}^0$  can be computed using the second part of (6) as  $m_{j+1}^0 = m_j^1$ .

The numerical example in Fig. 2 depicts the application of the metric in (6) in building the binary tree for two sets of generic extensions at steps  $j+1$  and  $j+2$ . At each step  $j$  only  $M$  nodes with the highest metric values are selected and used for the next set of extensions, whereas the rest are pruned.

In order to select the best  $M$  nodes without the use of the computationally demanding full quick-sort algorithm, two simple buffers, *One-Vector* and *Zero-Vector* of length  $M$ , have been used. In the former we load the nodes obtained by the *one extension* and in the latter those obtained by the *zero extension*. At each sorting step the two buffers heads are compared and the one with the highest metric is chosen as one of the  $M$  values. Depending on which buffer provides one of the  $M$  values the content of that buffer is right shifted by one position. The comparison and shifting continues until all the  $M$  values are selected. As an example the reader can follow the node selection process at stage  $j+1$  in Fig. 2.

One important feature of computing the node metrics using the ML criterion in (6) is partial ordering of the metric values in *One-Vector* and *Zero-Vector* buffers. The metric values in two buffers are always ordered in the ascending order from the left to right. This partial ordering permits the selection of the  $M$  nodes with the highest metric values using the simple comparison and shift operations.

**Table 3.** Average time required to decode a single frame, the corresponding PSNR performance and the memory usage for the SMA technique (0.5 bpp rate, QCIF (176 × 144) Foreman sequence and  $Q_f = 0.02$ ).

		BER=10 <sup>-3</sup>			BER=10 <sup>-4</sup>		
Alg.	M	Time(ms)	PSNR(dB)	Mem.(kByte)	Time(ms)	PSNR(dB)	Mem.(kByte)
SMA	8	34	29.7	541	14	30.2	420
[6]	8	106	29.7	541	37	30.2	420

**Table 4.** Comparative performance evaluation of three best samples from the proposed techniques; VMA, and SMA (0.5 bpp rate, QCIF (176 × 144) Foreman sequence and  $Q_f = 0.02$ ).

				BER=10 <sup>-3</sup>			BER=10 <sup>-4</sup>		
Alg.	M	M*	M**	Time(ms)	PSNR(dB)	Mem.(kByte)	Time(ms)	PSNR(dB)	Mem.(kByte)
VMA	2	4	8	46	29.7	336	11.0	30.2	261
SMA	8	8	8	34	29.7	541	14	30.2	420
VM+SMA	2	4	8	17	29.7	336	6	30.2	261
[6]	8	8	8	106	29.7	541	37	30.2	420

Performance of SMA in terms of average time and memory required to decode a single frame and PSNR are reported in Table 3. Comparing these results with those reported in [6], it can be observed that SMA exhibits the same PSNR performance and requires the same amount of memory as the error correction algorithm proposed in [6]. However, SMA provides for a speed-up improvement of about 68% and 62% for the BER values of 10<sup>-3</sup> and 10<sup>-4</sup>, respectively.

#### 4.3. Comparison of decoding schemes

From the data in Table 4, we conclude that for BER=10<sup>-3</sup> the SMA provides the best decoding time with a speed-up improvement of about 68% and no memory saving with respect to [6]. The speed-up improvement and memory saving for the VMA technique are 57% and 38%, respectively.

On the other hand, for the less severe transmission conditions, (BER=10<sup>-4</sup>), the VMA provides for the best decoding time with speed-up improvement of about 70% and a memory saving of 38% with respect to [6]. SMA instead offers a speed-up improvement of about 62% and no memory saving.

#### 4.4. Combined technique

Table 4 also reports the results obtained by combining VMA ( $M = 2$ ,  $M^* = 4$ ,  $M^{**} = 8$ ), and SMA techniques. Comparing the results for the combined technique and those in [6] the speed-up for BER=10<sup>-3</sup> is about 84% with the memory reduction of 39%, and no loss of PSNR performance. For BER=10<sup>-4</sup> the speed up is about 84%, the memory reduction is 38%, and there is no loss of PSNR.

### 5. CONCLUSIONS

In this paper we described two novel techniques to reduce the processing time and the memory requirements of the MQF decoding for JPEG2000 with a forbidden symbol. The proposed techniques effectively reduce the computational complexity of M-algorithm by up to 87%, with no or insignificant reduction in the PSNR or visual image quality. The proposed

techniques provide a good solution for error correction for motion JPEG2000 on a noisy wireless channel.

### 6. ACKNOWLEDGEMENT

This work was supported by SEA(ICT-214063) an Information and Communication Technologies project co-funded under EU's seventh framework (FP7).

### 7. REFERENCES

- [1] D. Nicholson, C. Lamy-Bergot, X. Naturel, and C. Poulliat, "JPEG 2000 backward compatible error protection with reed-solomon codes," *IEEE Trans. on Cons. Ele.*, vol. 49, no. 11, pp. 885–860, Nov 2003.
- [2] B. A. Banister, B. Belzer, and T. R. Fisher, "Robust image transmission using JPEG2000 and turbo-codes," *IEEE Sig. Proc. Let.*, vol. 9, no. 4, pp. 117–119, Apr 2002.
- [3] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. on Img. Proc.*, vol. 9, no. 7, pp. 1158–1170, Jul 2000.
- [4] A. Natu and D. Taubman, "Unequal protection of JPEG 2000 code-streams in wireless channels," in *IEEE GLOBECOM*, 2002, pp. 534–538.
- [5] T. Guionnet and C. Guillemot, "Soft decoding and synchronization of arithmetic codes: application to image transmission over noisy channels," *IEEE Trans. on Img. Proc.*, vol. 12, no. 12, pp. 1599–1609, Dec 2003.
- [6] M. Grangetto, E. Magli, and G. Olmo, "A syntax preserving error resilience tool for JPEG 2000 based on error correcting arithmetic coding," *IEEE Trans. on Img. Proc.*, vol. 15, no. 4, pp. 807–818, Apr 2006.
- [7] W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, and R. B. Arps, "An overview of the basic principles of the Q-coder adaptive binary arithmetic coder," *IBM Jou. of Res. and Dev.*, vol. 32, no. 6, pp. 717–726, Nov 1988.
- [8] I. Kozintsev, J. Chou, and K. Ramchandran, "Image transmission using arithmetic coding based continuous error detection," in *Data Compression Conf.*, 1998, pp. 339–348.
- [9] valgrind.org, "valgrind: Suite of tools for debugging and profiling linux programs(available at <http://valgrind.org>)." .