JOINT SOURCE-CHANNEL DECODING OF JPEG2000 IMAGES WITH UNEQUAL LOSS PROTECTION

by

Sohail Bahmani
Bachelor of Science, Sharif University of Technology 2006

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

In the
School of Engineering Science

SIMON FRASER UNIVERSITY

Fall 2008

APPROVAL

Name:	**Sohail Bahmani**
Degree:	**Master of Applied Science**
Title of Thesis:	**Joint Source-Channel Decoding of JPEG2000 Images with Unequal Loss Protection**

Examining Committee:

Chair: **Dr. Rodney Vaughan**
Professor, School of Engineering Science

____

Dr. Ivan Bajić
Assistant Professor, School of Engineering Science
Senior Supervisor

____

Dr. Atousa HajShirmohammadi
Lecturer, School of Engineering Science
Co-Supervisor

____

Dr. Jie Liang
Assistant Professor, School of Engineering Science
Supervisor

____

Dr. Daniel Lee
Associate Professor, School of Engineering Science
Internal Examiner

Date Defended/Approved: November 27, 2008______________________

ABSTRACT

This thesis proposes the joint decoding of JPEG2000 bitstreams and Reed-Solomon codes in the context of unequal loss protection. Using error resilience features of JPEG2000 bitstreams, the joint decoder helps to restore the erased symbols when the Reed-Solomon decoder fails to retrieve them on its own. The proposed joint decoding technique can deliver significant quality gain, though the process is often computationally exhaustive. To reduce the extra decoding time, we provide three solutions. The first two solutions employ smaller codeblocks and sub-coding-pass error localization to accelerate the process. Furthermore, we show how transmitting a relatively small amount of side information with high reliability may help the joint decoder by reducing the size of the search space and bypassing some of the JPEG2000 decoding iterations needed to verify the correctness of the restored source information. The improved joint decoder can perform significantly faster than the basic one.

Keywords: Unequal Loss Protection; JPEG2000; Reed-Solomon code; Joint Source-Channel Decoding
Subject Terms: Image Transmission; Coding Theory

تقدیم به مادر و پدر عزیزم مریم و حسین،
برادرم افشین و خواهرم کیمیا

To my dear parents Maryam and Hossein,
my brother Afshin and my sister Kimia

ACKNOWLEDGEMENTS

I sincerely would like to thank Dr. Bajić, my senior supervisor, and Dr. HajShirmohammadi, my co-supervisor, for their excellent guidance and support throughout my research and studies in Simon Fraser University. Their constructive feedbacks on my work were always encouraging and motivating.

I would like to thank Dr. Liang for reviewing this thesis carefully and providing his valuable comments. I would also thank Dr. Lee for reviewing and examining my thesis, and Dr. Vaughan for chairing the thesis defence.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Aiming for improvements in the delivered quality, robust transmission of multimedia information over lossy channels has been a topic of intense research for years. By introduction of scalable image and video coding standards, such as JPEG2000 [1] and H.264/SVC [2], transmission of the fragile scalable bitstreams has gained more attention in the research community. The progressive nature of scalable bitstreams makes them sensitive to error propagation; in fact, the first erroneous or erased bit in these bitstreams can damage all the information carried by the remainder of the bitstream. Therefore, Unequal Error Protection (UEP) and Unequal Loss Protection (ULP) schemes have been developed for protecting scalable bitstreams against errors and erasures, respectively.

There are many different frameworks proposed for transmission of scalable images with UEP or ULP [3-8]. In the procedure explained in [3], for instance, the source-coded bitstream is first divided into blocks of the same length. Each of these blocks is then encoded, perhaps at different code rates, by a Rate-Compatible Punctured Convolutional (RCPC) code for UEP or a Reed-Solomon (RS) code for ULP. The optimal code-rates are chosen using a dynamic programming algorithm, which maximize the expected Peak-Signal-to-Noise-Ratio (PSNR) of the decoded image with respect to the limited transmission rate budget. Unlike many other UEP or ULP schemes, which can start decoding only after the whole transmission is finished, these methods can decode the source information blocks “progressively” upon their arrival at the receiver (i.e., the intact source blocks that come before the first unrecoverable block can be decoded on the fly).

Furthermore, different decoding techniques aiming at robust communications have been proposed and analyzed in the literature. In [9] a maximum a posteriori (MAP) decoder is proposed for decoding of variable length codes transmitted through noisy channels. Corresponding to the transmitted source bitstream, this joint source-channel decoder finds the bitstream that minimizes the error between the transmitted and received source information. This technique operates solely based on the source and channel statistics available at the decoder, and it is not involved with channel coding. However, other analyzed approaches for error resilient decoding promote joint decoding of source codes and channel codes. In [10], the authors present Joint Source-Channel Decoding (JSCD) of JPEG2000 bitstreams and LDPC codes over error-prone Additive White Gaussian Noise (AWGN) channels. Similarly, a soft decoding algorithm, similar to the BCJR algorithm [11], for decoding arithmetic codes with a “forbidden symbol,” is presented in [12]. The authors have evaluated the efficiency of their approach, in particular, by iterative decoding of a JPEG2000 bitstream, which is transmitted through an AWGN channel, using the concatenation of the soft-input-soft-output-adapted JPEG2000 decoder and a convolutional decoder. Although the sensitivity of the arithmetic code to bit errors is the fundamental idea used in similar iterative decoding algorithms, soft decoders of JPEG2000 bitstreams might not be directly beneficial for correcting erasures, since no soft information is available at the receiver when erasures occur.

Throughout this thesis, we have focused on transmission of JPEG2000 images over packet-based networks, where packet loss is the dominant form of channel impairment; therefore, the ULP structure is chosen as the protection framework. To protect scalable bitstreams in ULP schemes with erasure correction codes, several approaches have been proposed in [13-15], among others. In these schemes, assuming a known erasure probability distribution of the channel, first a quality measure is formulated (e.g., expected distortion, or expected PSNR of the received bitstream). Then, this measure is optimized as an objective function to find a good feasible solution for optimal allocation of parity symbols to different segments of the bitstream. This optimization is usually performed by one of the various optimization algorithms such as local search or hill climbing, under a constraint on the rate budget (i.e., total number of bits allocated for the transmission) [13-15].

A well-optimized ULP structure is able to deliver all source bits to the receiver most of the time, provided that accurate channel state information is available at the transmitter. If this is not the case, for example when channel conditions change rapidly, some of the transmitted source bits may be rendered undecodable. In [16] we proposed a JSCD technique which restores some of the transmitted JPEG2000 bits that cannot be recovered by conventional decoding. To the best of our knowledge, the proposed technique is the first of its kind that introduces a joint decoder for JPEG2000 bitstreams and RS codes where erasure correction is the main concern. By means of the JPEG2000 Error Resilience (ER) features, JSCD technique could help to extract source information from the received ULP packets beyond what the plain ULP decoding can achieve. Depending on the amount of restored source information, JSCD can yield significant quality improvement.

Since the basic JSCD algorithm of [16] is often very time-consuming, we proposed three solutions to speed up the joint decoding process. One way is fairly straightforward and involves using smaller JPEG2000 codeblocks. In fact, short coding passes of smaller codeblocks result in better error localization, which is crucial for JSCD. The second solution is to locate not only the coding pass in which the JPEG2000 detects an error, as the basic JSCD does, but also locate the symbol in that particular coding pass, at which the decoding is interrupted. Later in the thesis, we explain how this slightly improved error localization is particularly useful in decoding of the coding passes that are relatively long, and potentially perpetuate the joint decoding. In contrast to the previous solutions, which completely rely on the JPEG2000 features, the third speedup method relies on the use of "auxiliary bits" which are transmitted as side information, and which reduce the search space of the joint decoder. These auxiliary bits also have a parity-like function, which helps to reduce the number of calls to the JPEG2000 decoder drastically and, consequently, save a significant amount of time. Some solutions to limit the number of bits spent on the side information are also provided.

This thesis contains materials from our previous works presented in [16-18]; the last two are under review and have not been published yet. The thesis is organized as follows. Chapter 2 provides the required background knowledge, which includes a brief description of JPEG2000 and some of its features, and an overview of the ULP technique. In Chapter 3, basic JSCD is explained. This chapter also contains the results that demonstrate the quality improvement that JSCD can offer. The accelerated JSCD processes are detailed in Chapters 4. The complexity model, which estimates the effect of the auxiliary bits on the JSCD, is introduced and analyzed in Chapter 5. Conclusions are presented in Chapter 6. Finally, Appendices A and B provide descriptions of the implementation issues and the user guide, respectively.

CHAPTER 2: BACKGROUND

This chapter provides the required background knowledge about the components of the JSCD. Structure and features of JPEG2000, the scalable source coder that we chose for the JSCD, are concisely described in the next section. Section 2.2 explains the ULP and the way it functions.

2.1. JPEG2000

2.1.1. STRUCTURE

JPEG2000 is an image compression standard developed based on Discrete Wavelet Transform (DWT) [1], [19]. After applying the colour transforms on the raw image, the resulting matrices can be optionally cut up into rectangular tiles to be compressed separately. Multiple tiles are generally used for large images.

The low-pass and high-pass filtering along with the sub-sampling by the DWT, result in several *subbands* in the transformed tile-components. As illustrated in Figure 2-1, in the JPEG2000 hierarchical structure, in each tile-component, each set of subbands within a common frequency band are referred to as a *resolution*. With the exception of the lowest resolution, subbands in resolution #n are labelled by HL_n, LH_n or HH_n, depending on whether they are low-pass filtered or high-pass filtered in the horizontal and

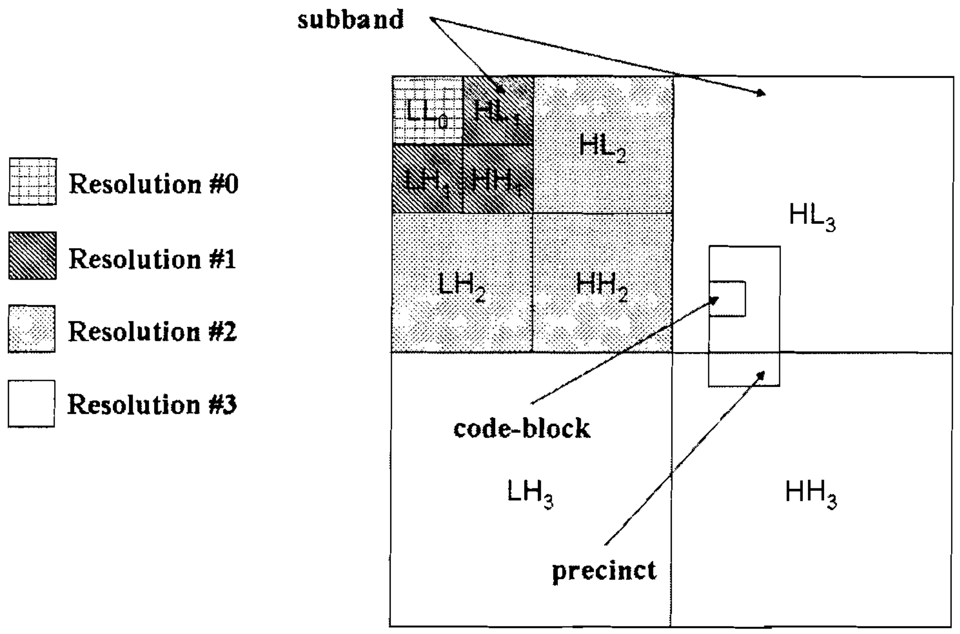

Figure 2-1. Structure of a typical JPEG2000 tile-component

vertical directions. The LL_0 subband, or in fact resolution #0, contains most of the information in smooth areas of the tile-component. The information of each resolution level provides the details required to be added to the information of the previous resolution levels to produce a higher resolution image.

Subbands are further partitioned into rectangular-shaped segments named *codeblocks*, which are the smallest units in the JPEG2000 standard encoded independently. Each dimension of these codeblocks should be a power of two not less than four, and the total area that a codeblock covers must be less than or equal to 4096 samples. Furthermore, there are virtual rectangular subdivisions of the resolutions, called *precincts*, each of which includes a whole number of codeblocks. In all resolution levels except resolution #0, width (height) of the codeblocks would be the smaller number between the maximum width (height) assigned to the codeblock and half the width (height) of the precinct that contains the codeblock. This restriction implies that each precinct outside of the LL_0 subband contains at least two codeblocks both vertically and horizontally. The precincts are basically employed to organize the compressed bitstream of the codeblocks in the JPEG2000 bitstream. For each precinct, there is a *packet* of the entropy-coded data in the final JPEG2000 bitstream. The headers of these packets not only include the spatial information of the precinct and the codeblocks it contains, but also they can contain some markers that are helpful for error detection. Nonetheless, precincts do not have a direct role in the compression process.

2.1.2. FEATURES

2.1.2.i) Scalability

One of the important parts of the JPEG2000 standard is bit-plane coding. Using the MQ-coder, a context-adaptive binary arithmetic coder, the bit-plane coder encodes the information bits carried by the quantized wavelet samples in each codeblock from the most significant bits level (most significant bit-plane) to the least significant one, consecutively [1]. This encoding process is performed in three *coding passes* for each bit-plane: the *significance propagation* pass, the *magnitude refinement* pass, and the *cleanup* pass. A sample is called "significant", if its most significant non-zero bit has been previously scanned. In the significance propagation pass, the current “insignificant” sample bit is encoded, if at least one of its eight immediate spatially adjacent samples is significant. Note that the sign bit of a sample is coded following the encoding of the most significant non-zero bit of that sample. The magnitude refinement pass involves the encoding of the remaining samples in the current bit-plane that are significant and have not been encoded in the previous pass. Finally, the cleanup pass encodes all the remaining samples in the current bit-plane, which are insignificant and have not been included in the previous passes.

The bit-plane coding mechanism generates an embedded compressed bitstream for each codeblock in the image. By different ordering of the generated coding pass segments from all of the codeblocks, the JPEG2000 encoder can produce bitstreams with specific *scalability* properties. These scalability features include quality (layer), resolution, position, and (color) component scalabilities. A *quality-scalable* bitstream, for instance, is organized such that the quality of the resulting image improves progressively as the decoding continues. Figure 2-2 illustrates an example of this quality improvement when the number of decoded “quality layers” increases. High frequency patterns such as the sharp edges and the textures with many details would be more visible when the quality layers corresponding to higher source rates are decoded.

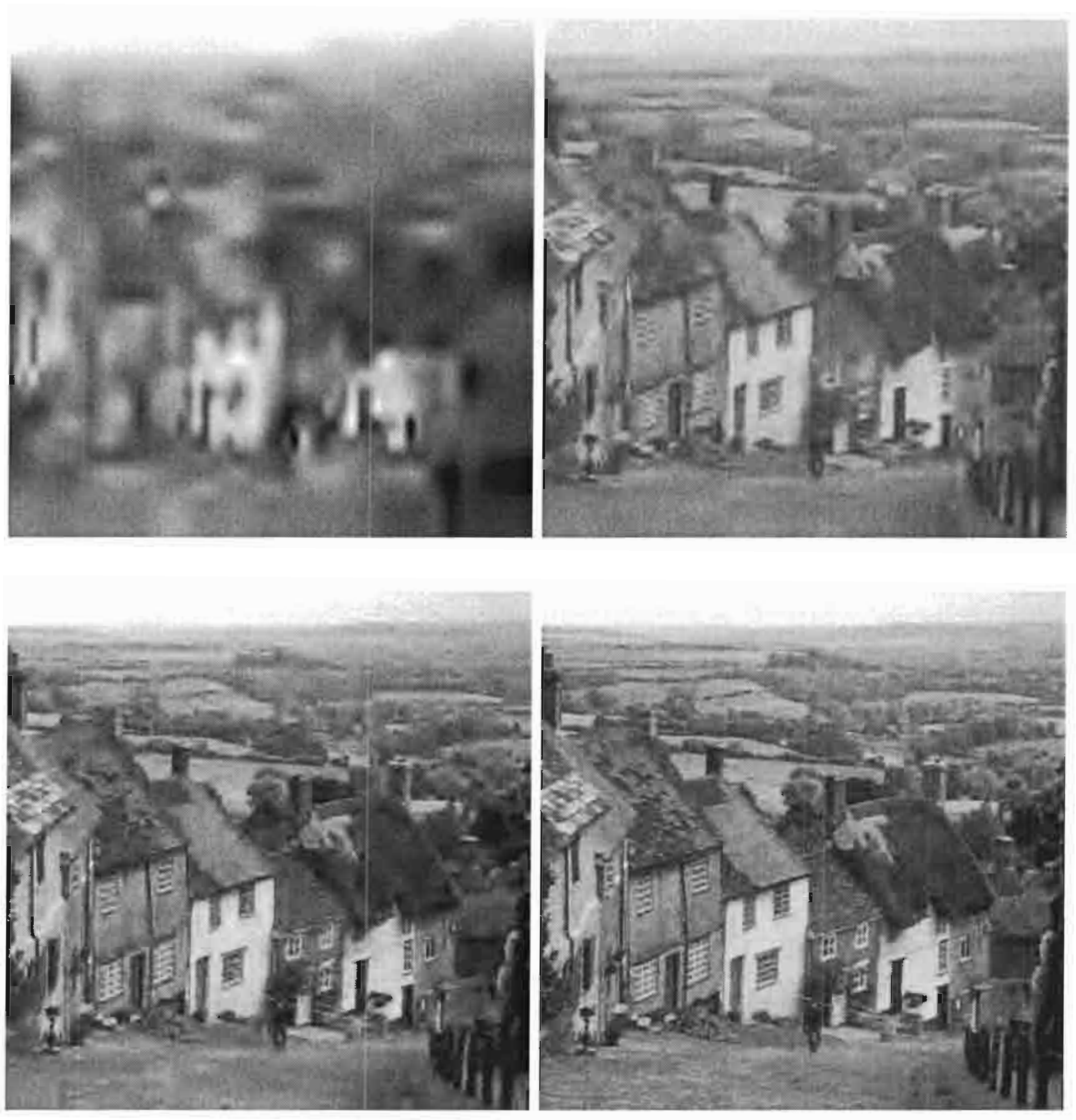

Figure 2-2. Four-layer quality-scalable *Goldhill* image decoded at the first layer with 0.01 bpp (top left), the second layer with 0.05 bpp (top right), the third layer with 0.1 bpp (bottom left), and the fourth layer with 0.2 bpp (bottom right).

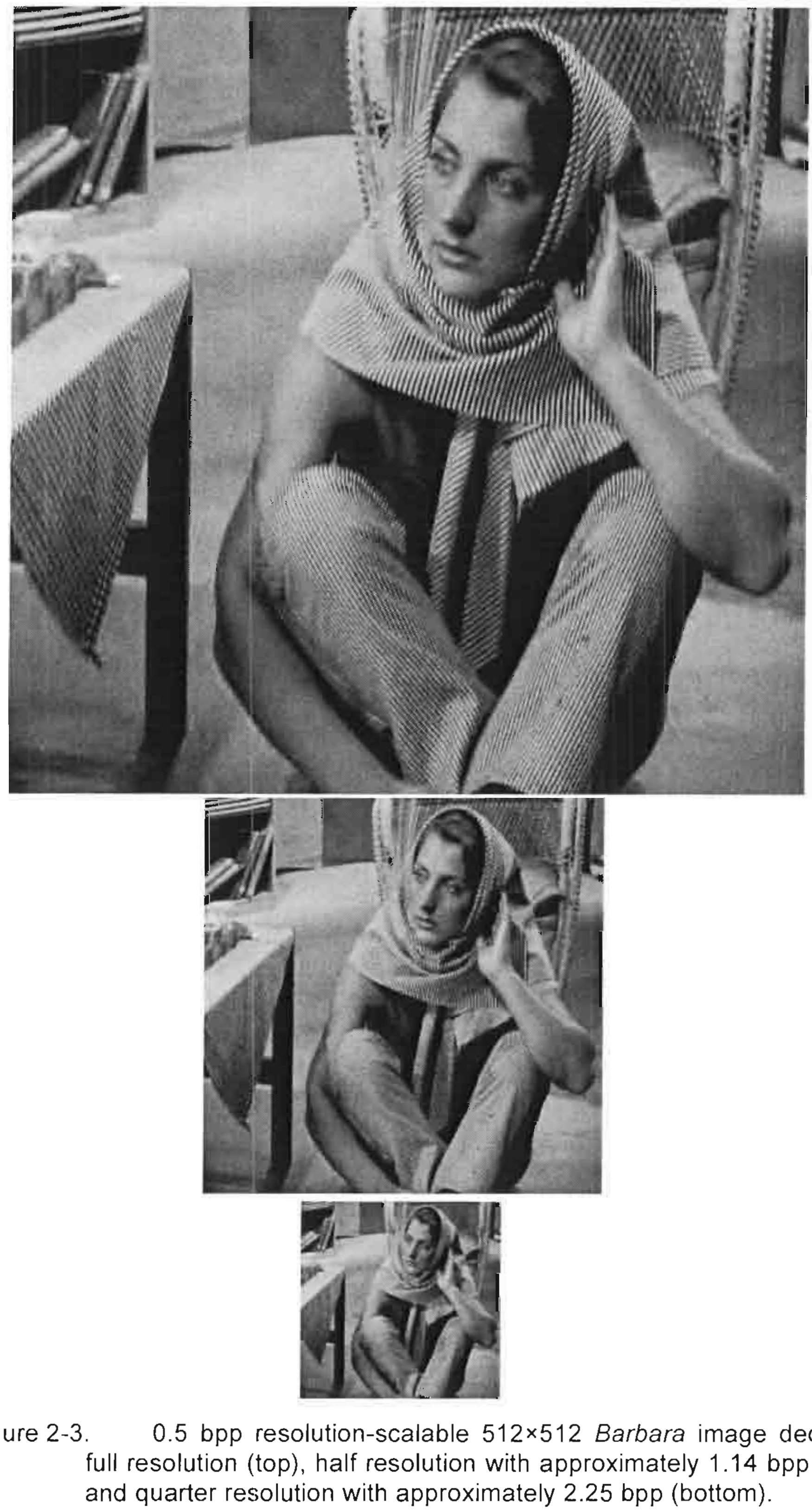
Figure 2-3. 0.5 bpp resolution-scalable 512×512 *Barbara* image decoded at full resolution (top), half resolution with approximately 1.14 bpp (middle) and quarter resolution with approximately 2.25 bpp (bottom).

Since in our proposed JSCD technique, coding passes of a codeblock should not be scattered throughout the bitstream, we have used *resolution-scalable* JPEG2000 bitstreams to keep the information of each codeblock contiguous in the bitstream. As Figure 2-3 depicts, truncation of the information of higher DWT levels from resolution-scalable JPEG2000 bitstreams results in lower resolution images.

2.1.2.ii) Error Resilience Features

Since entropy coders are usually imperfect in terms of compression, some residual redundancy always remains in the bitstream that they generate, which can be exploited for error detection by the decoder [20]. For example, in Huffman codes, bit errors may result in invalid codewords, which are recognizable to the decoder. In contrast, arithmetic codes are more susceptible to error propagation because they do not have any intrinsic resynchronization capability [21]. Nonetheless, the JPEG2000 standard provides some additional means by which its arithmetic coder, the MQ-coder, can be manipulated to generate bitstreams that are more error-sensitive. On decoding failures, the remaining section of the codeblock's bitstream would be partially decoded or completely discarded. The reaction of the decoder to an error depends on the features used at the encoder side. For instance, when the RESET mode of Kakadu encoder [1] is enabled, the context states used by the MQ coder would be reset at the beginning of each coding pass. Although some compression efficiency may be lost with this encoding constraint, the JPEG2000 decoder can now tolerate the errors in the generated bitstream to some extent. In fact, based on the type of the coding pass where the first error is detected, the decoder can decide which parts of the remained codeblock bitstream are still decodable. In [22], this error resilient decoding technique of JPEG2000 images is investigated along with several others.

In this thesis, we use the basic ER feature based on two coding modes, namely ERTERM and RESTART, because they fit well with the constraints of our JSCD technique. With these two modes switched on, inter-codeblock error propagation is prevented. The activated RESTART mode forces the MQ-coder to restart at the beginning of each coding pass, and causes the length of the coding passes to be saved up in their corresponding packet headers. As a result, each coding pass obtains a separate arithmetic codeword segment. Note that the RESTART mode does not initialize the context states, therefore the coding passes are not independently decodable unless RESET mode is also utilized. In addition, the ERTERM switch decides on the predictable termination policies for each MQ and/or raw codeword segment. When encoding is performed in the RESTART mode, these policies help to interrupt the decoding with high probability within only a few coding passes after the first corrupted coding pass. In other words, the mentioned decoding modes together provide a reliable tool for error localization in the JPEG2000 bitstreams. Some statistical results on the efficiency of this error-resilient decoding method for error localization are presented in [10]. For example, by assuming a residual bit error rate of 5.81×10^{-4} in the test JPEG2000 images which have 32×32 codeblocks, they have reported only 0.49% failure of detecting the first corrupt coding pass of the codeblocks. In addition, they have mentioned that 90% of these detection failure cases occurred when the first corrupt coding pass of a codeblock was its last coding pass.

2.2. UNEQUAL LOSS PROTECTION

Scalable source encoders, such as JPEG2000, are designed such that the earlier parts of the bitstreams that they generate carry the more important data, and the following parts of the bitstream can append the details as their importance decreases gradually. Ultimately, there are fully embedded bitstreams for which even one more decoded bit can result in noticeable difference in the outcome. Therefore, when it comes to channel coding of scalable bitstreams, it is reasonable to spend a larger fraction of the parity symbols on the earlier parts of the compressed bitstream compared to parts that come later. ULP schemes are developed based on this unequal distribution of the parity symbols to provide better protection for scalable bitstreams against packet erasures. As shown in Figure 2-4, a ULP structure can be represented by a simple matrix. The packets, which will be transmitted through the channel, are represented by the rows of this matrix.

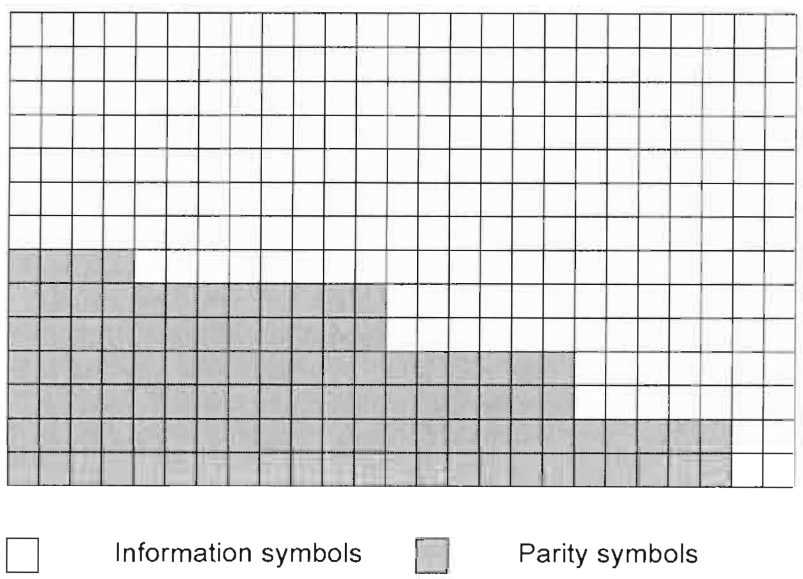

Figure 2-4. A typical ULP matrix.

One may ask how the source message is sorted inside the ULP matrix. To preserve the scalability features, the source bitstream should not be cluttered; however, as long as the number of parity symbols that each particular source symbol receives does not change, it is possible to place the source information in the ULP matrix in different, but known, orders. However, in this work, the source information simply fills in the ULP matrix column by column.

Due to their erasure correction competence, Reed-Solomon (RS) codes [23] are the typical choice for ULP, so we choose them for our work as well. The fundamental idea behind the construction of the RS codes is that when a single-variable polynomial of degree less than K is oversampled at $N>K$ different points, knowing any K of the resulting samples is sufficient to reconstruct the polynomial (i.e., to find the value of its coefficients). Using this fact, an RS(N,K) encoder assigns a unique codeword (i.e., channel codeword) of N symbols to each block of K source symbols; all of the symbols in the resulting codewords belong to the same finite field, which should contain at least N different symbols. The important property of these codes is that the number of positions in which any pair of them have different symbols (i.e., the Hamming distance between the two codewords) is greater than N-K. In other words, the RS codes can restore the message if the number of erased symbols does not exceed N-K.

The RS encoders that use oversampling do not generate systematic codewords. Note that different encoding/decoding techniques for RS codes, which are more efficient computationally and/or able to work with systematic codes, have been developed, but they are not theoretically as simple as the original oversampling algorithm proposed in [24]. However, for a given message length and codeword length, the codewords that these encoders create have the same minimum distance, and consequently the same error and erasure correction capability [23].

To achieve the best outcome of using ULP, it is necessary to find the optimal distribution of the parity symbols among the columns of the ULP matrix. Given the operational distortion-rate curve of the image that will be transmitted, the expected distortion, which is the objective function of the required optimization, can be written as:

$$E[D]=\sum_{i=0}^{N} p_i D\left(m \sum_{1\le j\le L | f_j \ge i} \left(N-f_j\right)\right), \tag{2.1}$$

where $D(.)$ is the operational distortion-rate function, m is the number of bits per each symbol, N is the number of packets, f_j is the number of parity symbols used in the j–th column, L is the packet length (i.e., the number of symbols in each packet), and p_i is the probability of having exactly i erasures. Since the rate budget for transmission is limited, this is a constrained optimization. It is common to apply the rate constraint by fixing the number of packets (N) and their length (L) [13-15].

Under these conditions, the optimization algorithm in [15], which uses local search, provides competitive results with low computational cost. In this algorithm the search begins from a rate-optimal pattern in which $f_1 = f_2 = f_2 = \ldots = f_L$ and the objective function attains its minimum. Then, starting from the leftmost column in a consecutive order, the algorithm increases f_j's by one. At each step, if the value of the objective function is reduced, then the best value found so far will be updated by evaluation of the objective function for the generated pattern. This process continues until further changes in the number of parity symbols do not improve the best value found so far. The final pattern and the corresponding value of the objective function form an approximate solution for the global optimization problem. We used this algorithm in our simulations to find the number of parity symbols in each column of the ULP matrix; though, instead of minimizing the expected distortion, the same algorithm is used to maximize the expected PSNR as the objective function.

For $1 \leq j \leq L$, let f_j^* be the number of parity symbols that is assigned to the *j*–th column by the optimization algorithm. If $k_j = N - f_j^*$ represents the number of source symbols in the *j*–th column, then for each *j* from 1 to *L*, k_j symbols are read from the JPEG2000 bitstream and encoded by an RS(N,k_j) encoder. The *j*–th column in the ULP matrix is then filled with the generated RS codeword and the process continues for the remaining columns.

Suppose that after transmission, *e* packets are erased from the ULP matrix. Then, at the decoder side, only RS codewords with $f_j^* \geq e$ can be successfully decoded. Once these codewords are decoded, the task of the ULP scheme is over and the information part of the decoded codewords is ready to be decoded by a JPEG2000 decoder. However, ER features of JPEG2000 can be exploited to decode the bitstream beyond this point, and thus obtain an image with higher quality. In the next chapter we explain how this can be done.

CHAPTER 3: JOINT SOURCE-CHANNEL DECODING

This chapter describes the JSCD technique that we proposed in [16]. In the following section, the joint decoding process is explained after the motivation for using the JSCD is briefly discussed. The simulation results are presented in Section 3.2.

3.1. THE JOINT DECODING PROCEDURE

Most ULP schemes, such as those in [13-15], try to minimize the expected distortion or alternatively maximize the expected quality (PSNR) of the received image. Hence, they deliver fairly good image quality *on the average*, when the actual channel model matches the one assumed in the optimization. However, when this is not the case, significant quality degradation can occur. Even when the channel model assumption is correct, worst-case performance may be significantly lower than the average performance, because some channel realizations may suffer from much higher loss (burst loss) than the average loss rate. Image quality degradations caused by these effects can be alleviated by our proposed JSCD method.

In the JSCD technique presented in [10], the ER features of JPEG2000 are utilized to accelerate the convergence of the LDPC soft decoder in an iterative fashion. Analogously, in our JSCD for ULP, JPEG2000 ER features are exploited to help the decoder decode more columns from the ULP matrix, as explained below.

As mentioned in Chapter 2, each column of the ULP matrix is an RS codeword. Therefore, it can be decoded as long as the number of erased symbols (i.e., the number of lost packets) is not greater than the number of their parity symbols. Consider an RS(N,k) codeword, and suppose that $e = N-k+d$ packets are lost during transmission ($d \geq 1$). If we guess the values of d of the erased symbols in this codeword, the RS decoder can fill the remaining erasure locations. Correctness of the guessed symbols and the corresponding restored symbols can be verified by the ER features of JPEG2000. Once the guessing iteration is finished and the RS decoder fills all of the erasure locations, we try to decode the information portion of the RS codeword by the JPEG2000 decoder. If the JPEG2000 decoder complains, we modify the guesses and repeat the procedure. Figure 3-1, illustrates a simplified block diagram of the JSCD.

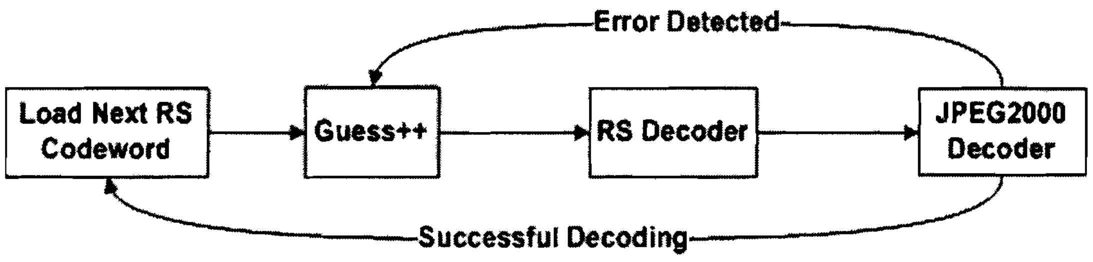

Figure 3-1. Simplified block diagram of the basic JSCD.

Note that each of the guess values applied to a column of the ULP matrix leads to a valid RS codeword, but only one results in the correct JPEG2000 bitstream. To prove this statement, we first show that the number of RS(N,k)

codewords that all coincide at the same *k*-*d* positions is equal to $2^{m\times d}$. Let I_1 and I_2 be two disjoint sets, which contain *k*-*d* and *d* indices of positions in the RS codewords, respectively. Furthermore, let $u(C) = (u_1(C), u_2(C))$ be a *k*-tuple in which $u_1(C)$ and $u_2(C)$ represent the symbols of the RS codeword *C* at positions indicated by I_1 and I_2, respectively. If there are more than $2^{m\times d}$ different codeword *C*'s with the same $u_1(C)$, then, since the number of possible choices for $u_2(C)$ is $2^{m\times d}$, at least two of them would have the same $u_2(C)$ which contradicts the fact that the minimum distance between these codewords is *N*-*k*+1. Therefore, if for each (*k*-*d*)-tuple *u* we denote the number of codewords *C* with $u_1(C) = u$ by $s(u)$, we have:

$$s(u) \le 2^{m\times d} . \tag{3.1}$$

Then we can write the following inequality:

$$\begin{aligned}\text{Total number of the RS codewords} &= \sum_{C} 1 = \sum_{u} \sum_{C:u_1(C)=u} 1 \\ &= \sum_{u} s(u) \le 2^{m(k-d)} \times 2^{m\times d} = 2^{m\times k} .\end{aligned} \tag{3.2}$$

On the other hand, there are exactly $2^{m\times k}$ different RS codewords with *k* message symbols each of which has *m* bits. Therefore, the inequality in (3.2) never becomes strict and for any *u* we should have $s(u) = 2^{m\times d}$. In other words, when $u_1(C) = u$ is known, each of the $2^{m\times d}$ possible values that $u_2(C)$ can take (i.e., the guess values) results in a valid RS codeword, which means that the RS decoding on its own cannot help to identify the correct bitstream.

To explain the joint decoding procedure in more detail, let f_j^*'s be the solution to the optimization problem discussed in Chapter 2. Let $f^{CS} = \max_{e > f_i^*} f_i^*$ be the number of parity symbols in those columns where the number of parity symbols is just less than *e*, the number of erasures. The superscript *CS* stands for *critical segment,* a part of the ULP matrix that consists of columns that have exactly f^{CS} parity symbols. To retrieve the information portion of a critical segment, it is sufficient to find the correct value of $d = e - f^{CS}$ erased symbols in each of its columns; the remaining erased symbols in the critical segment will be filled in by the RS decoder.

To ensure the validity of a repaired column of the critical segment, we let JPEG2000 decoder attempt to decode the information portion of that column. Due to JPEG2000 ER features, an invalid segment in the bitstream would cause the decoder to complain (with high probability); in that case, the incorrect guess is discarded and another guess is made. Since JPEG2000 ER features cannot guarantee detection of all errors, some wrong guesses will go by undetected, but such cases are relatively rare in practice [10]. The pseudocode presented in Figure 3-2 outlines the main steps of the proposed JSCD. A mechanism that we considered to moderate the time spent on JSCD is also included in the pseudocode. When all possible guesses for the present column are made unsuccessfully, this mechanism prevents the joint source-channel decoder from returning to the columns before the past r_{max} columns to correct the undetected wrong guesses.

```
identify the CS;
{$r_1<r_2<\ldots<r_e$} ← indices of the erased packets;
$c_l$ ← the l-th column of the ULP structure;
CP(x,y) ← index of the coding pass which contains the y-th
symbol of the x-th column ($c_x$);
i ← index of the first column of the CS;

LOOP$_1$:
$i^*$ ← i;
while CP(i+1,$r_1$) $\le$ CP($i^*$,$r_d$)
     i++;
$i^{**}$ ← i;

LOOP$_2$:
     while no unverified guess for $c_i$ remains
     {
          reset $c_i$;
          RS: decode $c_i$;
          i--;
     }
     if $i^{**}$-i $\ge$ $r_{max}$
          terminate;
     take the next unverified value as the guess for
     column i;

     RS: decode $c_i$;

     J2K: decode the coding passes contained in $c_i$ through
     $c_{i^{**}}$;

if J2K decoder complained
{
     i ← $i^{**}$;
     goto LOOP$_2$;
}
else
{
     if end of the CS is reached
          terminate;
     i ← $i^{**}$+1;
     goto LOOP$_1$;
}
```

Figure 3-2. Simplified pseudocode of the JSCD.

3.2. RESULTS

We tested our JSCD algorithm on several standard 512×512 gray-scale images: *Lena, Barbara, Gold Hill,* and *Boat.* The simulation setup was as follows. The ULP matrix consists of 255 packets, each 100 bytes long. This results in the total rate budget of 0.7782 bits per pixel (bpp). Test images were encoded at this rate using the Kakadu implementation of JPEG2000 [1], with three different codeblock sizes (16×16, 32×32, and 64×64), 128×128 precincts, and ERTERM and RESTART switches turned on. Although SOP and EPH markers, which indicate the start and the end of a JPEG2000 packets (not to be confused with the network packets), do not affect our simulation results, they were also activated. Then, we derived operational PSNR-rate curves from the encoded images and found the optimal ULP protection levels using the algorithm from [15] for each image at 10%, 20%, and 30% mean loss rates. The resulting patterns for ULP are reported in Tables 3-I to 3-IV.

For convenient integration with Kakadu, 8-bit symbols (i.e., bytes) were used as the RS symbols; in our implementation, channel encoding and decoding is performed by Phil Karn's RS codec [25]. Also, to limit the decoding time, we only considered cases where just one symbol (byte) needs to be guessed in each column of the critical-segment (i.e., $d = 1$). Since any of the erased symbols can be chosen for guessing, in our implementation, we simply choose the first erased symbol in each of the critical segment columns to be guessed.

TABLE 3-I. PATTERN OF THE ULP MATRICES FOR LENA

Loss Rate	10%	column	1	2–7	8–18	19–33	34–73	74–100			
		# of parity symbols	45	43	41	40	39	38			
	20%	column	1	2	3–4	5–8	9	10–21	22–83	84–100	
		# of parity symbols	75	73	72	71	70	69	67	66	
	30%	column	1	2	3–5	6–10	11–24	25	26–98	99	100
		# of parity symbols	104	103	101	100	98	97	96	95	94

TABLE 3-II. PATTERN OF THE ULP MATRICES FOR BARBARA

Loss Rate	10%	column	1	2	3–4	5–6	7	8–100	
		# of parity symbols	44	43	41	40	39	38	
	20%	column	1	2	3–4	5–7	8–100		
		# of parity symbols	73	71	69	67	66		
	30%	column	1–2	3–5	6–8	9	10–13	14–21	22–100
		# of parity symbols	102	98	97	95	94	93	92

TABLE 3-III. PATTERN OF THE ULP MATRICES FOR GOLDHILL

Loss Rate	10%	column	1	2	3–6	7–10	11–16	17–25	26–100			
		# of parity symbols	46	45	43	42	41	40	39			
	20%	column	1	2	3	4–7	8–11	12–18	19–29	30–84	85–98	99–100
		# of parity symbols	77	76	74	73	72	71	70	68	67	66
	30%	column	1–2	3–5	6–8	9–13	14–33	34	35–97	98–99	100	
		# of parity symbols	105	101	100	99	97	96	95	94	93	

TABLE 3-IV. PATTERN OF THE ULP MATRICES FOR BOAT

Loss Rate	10%	column	1	2–3	4	5–9	10–16	17–73	74–100
		# of parity symbols	45	43	42	41	40	39	38
	20%	column	1	2	3–4	5–11	12–100		
		# of parity symbols	102	98	97	95	94		
	30%	column	1–2	3–5	6	7–32	33–98	99	100
		# of parity symbols	102	99	98	97	96	95	94

Figures 3-3 to 3-8 illustrate the theoretical expected gain in PSNR that the JSCD can deliver versus mean packet loss rate, for different images. Given the pattern of ULP, the expected PSNR that is achievable without JCSD can be calculated using (2.1), except that the distortion-rate function should be replaced by PSNR-rate function (i.e., *PSNR*(.)). Similarly, when JSCD by guessing up to *d* erased symbols per RS codeword is also performed, the expected PSNR can be formulated as:

$$E[PSNR_{JSCD}] = \sum_{i=0}^{N} p_i PSNR\left(m \sum_{1\le j\le L \,|\, f_j + d \ge i} \left(N - f_j\right) \right), \tag{3.3}$$

which helps to find the expected PSNR gain achieved by using JSCD. As the graphs show, when the mean loss rate of the channel coincides with the loss rate for which we optimized the ULP matrices, the expected gain never exceeds a few hundredths of a dB. However, channel mismatch can increase the probability of having a CS; consequently, expected PSNR gain can rise to a noticeable level (e.g., Fig 3-3 shows near 1.4 dB gain when the ULP matrix is optimized for 10% mean loss rate, but the channel loss rate is roughly 17%). The peaks on the curves indicate the channel loss rates at which it is more likely to have a CS. The number and the location of these peaks depend not only on *d*, but also on the utilized ULP patterns. If the number of parity symbols changes from one column to the next column of a ULP matrix in smaller steps, fewer peaks can be seen in its corresponding graph. The effect of *d* on variation of the expected gain is not reflected on the graphs because only the case of *d* = 1 is considered.

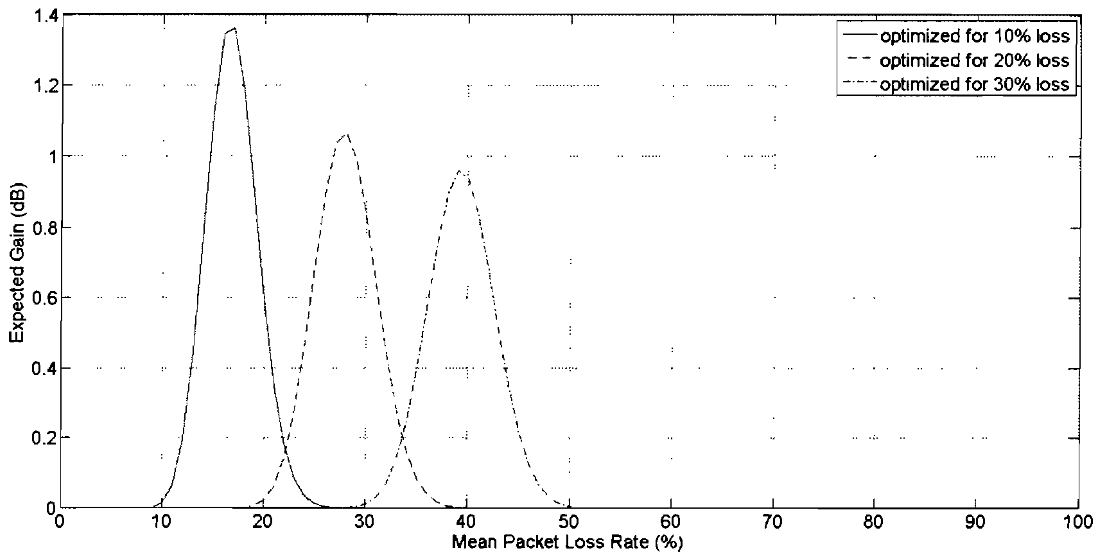

Figure 3-3. Expected gain of using JSCD for Lena (in dB) vs. mean packet loss rate when 255×100 ULP matrices are used.

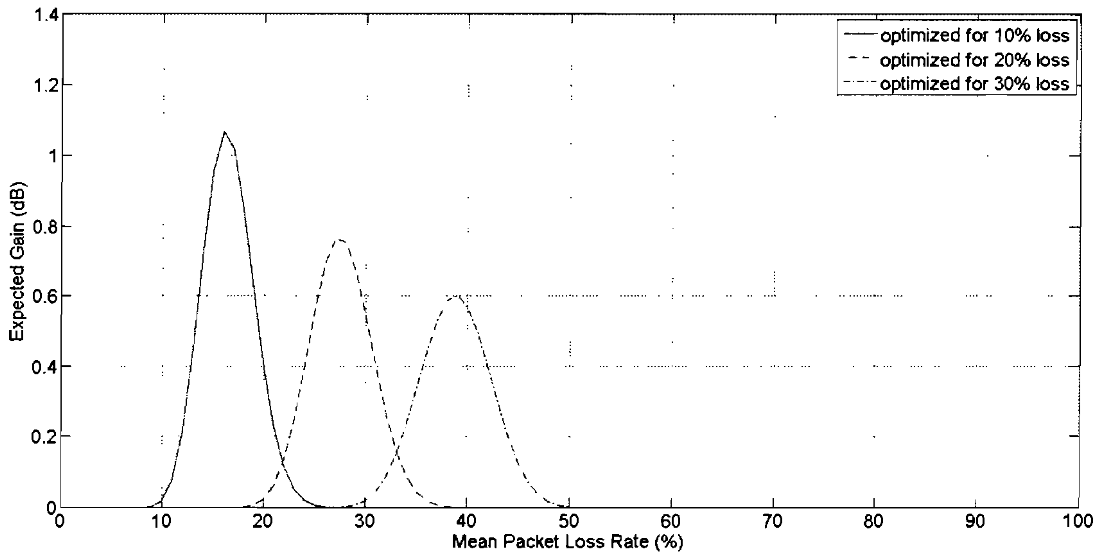

Figure 3-4. Expected gain of using JSCD for Barbara (in dB) vs. mean packet loss rate when 255×100 ULP matrices are used.

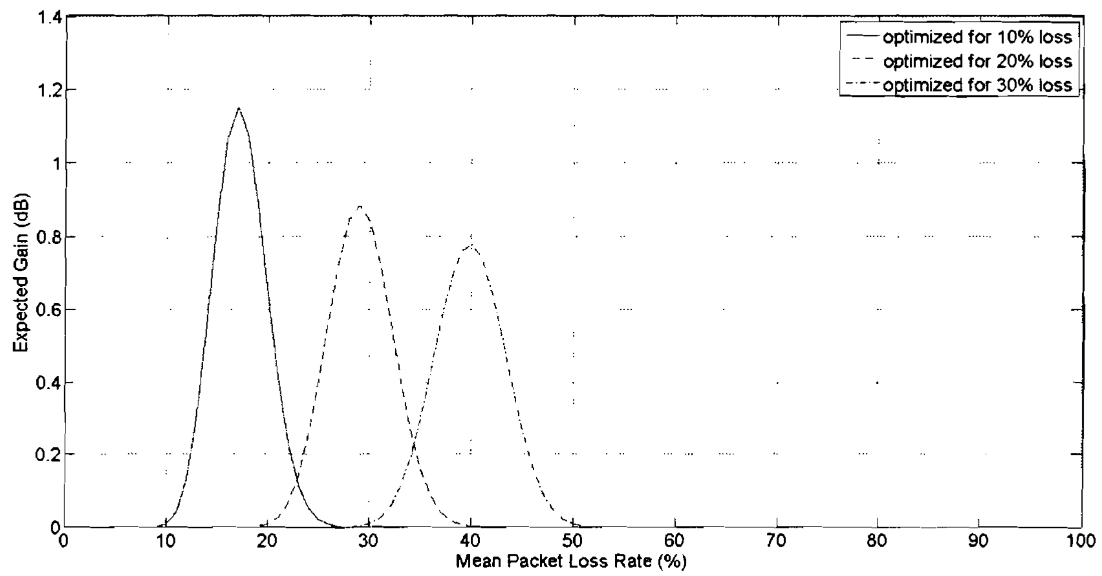

Figure 3-5. Expected gain of using JSCD for Goldhill (in dB) vs. mean packet loss rate when 255×100 ULP matrices are used.

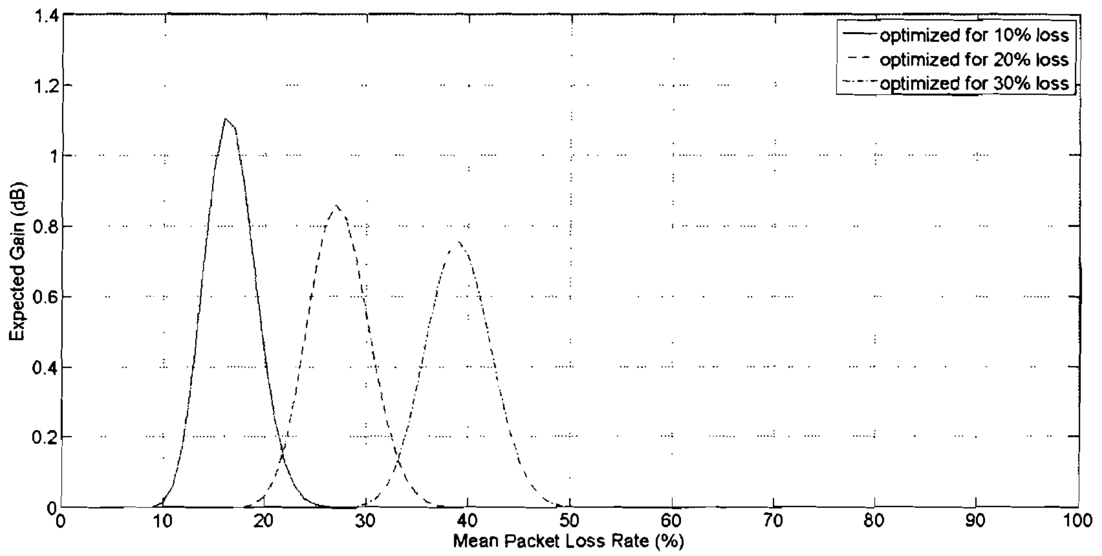

Figure 3-6. Expected gain of using JSCD for Boat (in dB) vs. mean packet loss rate when 255×100 ULP matrices are used.

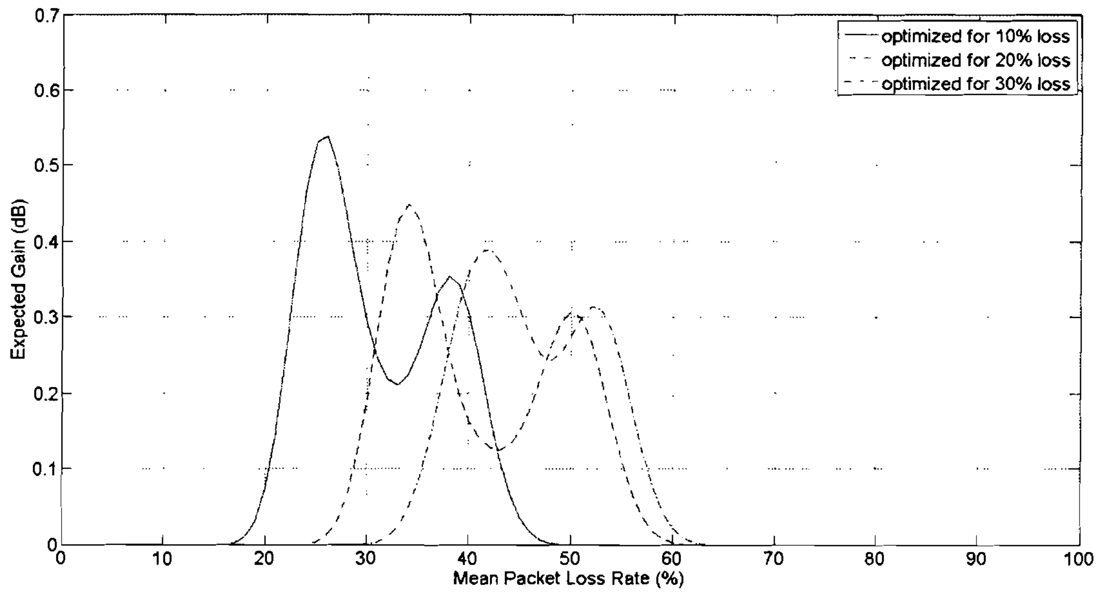

Figure 3-7. Expected gain of using JSCD for Lena (in dB) vs. mean packet loss rate when 255×50 ULP matrices are used.

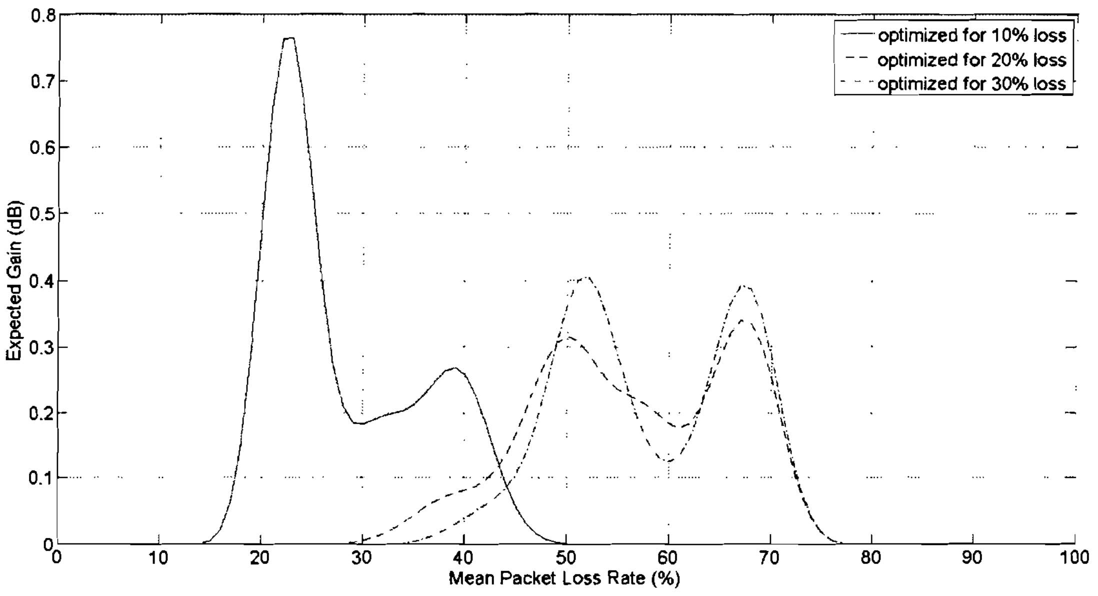

Figure 3-8. Expected gain of using JSCD for Lena (in dB) vs. mean packet loss rate when 255×150 ULP matrices are used.

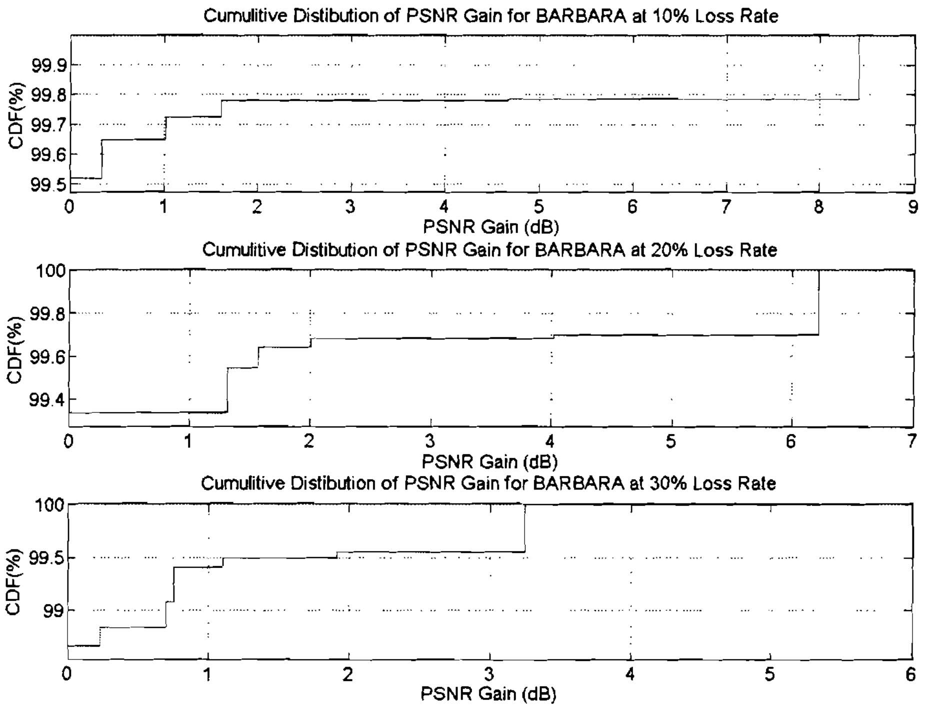

Figure 3-9. Cumulative distribution of PSNR gain provided by JSCD for Barbara at different mean loss rates, where no channel mismatch occurs.

Although JSCD cannot improve the expected PSNR gain when there is a little or no mismatch between the channel and its model, it is still possible to obtain higher quality images by using the proposed JSCD. Figure 3-9 exemplifies the gain of using JSCD at different mean loss rates when there is no channel mismatch. The graphs show that it is very unlikely to have a noticeable PSNR gain from JSCD, though large PSNR gains are not completely impossible to occur. For example, in the case where the mean channel loss rate is supposed to be 20%, the probability of having less than 2 dB gain is just below 99.7%, which means more than 0.3% of the times the gain is 2 dB or more. Therefore, in a scenario where the image is broadcasted to 1000 users through a channel with the same conditions, three users can be expected to gain 2 dB or more by using the JSCD compared to the case where receivers do not perform the joint decoding.

Tables 3-V to 3-VIII show the average PSNR based on outcomes of 100 simulations, when JSCD is not applied ($\overline{PSNR}$), and when basic JSCD is performed ($\overline{PSNR_0}$), both in dB, for different tested images using different codeblock sizes. Similar quantities are reported in Tables 3-IX to 3-XII providing some examples of the performance of JSCD at different transmission rate budgets (i.e., 255 packets of 50 bytes (0.3891 bpp) and 255 packets of 150 bytes (1.1673 bpp)). The loss rate column in the tables represents the mean channel loss rate used by the parity level optimization algorithm [15]. The experiments are carried out for the last three possible critical segments, which are represented in the tables by CS#1, CS#2 and CS#3. The width of each critical segment is also reported in the tables.

The results show that JSCD has the potential to improve the received image quality significantly. For sufficiently wide critical segments, the proposed JSCD demonstrates remarkable improvement in the PSNR (e.g., Table 3-VI, $\overline{PSNR_0}$ at CS#1 , 64×64 codeblock size , and 10% loss rate), while for narrow critical segments (e.g., those with one or two columns), JSCD causes marginal PSNR improvement, if any. For further illustration, Figures 3-10 and 3-11 demonstrate the visual improvement in the quality of the decoded image for two

TABLE 3-V. AVERAGE PSNR BEFORE JSCD ($\overline{PSNR}$), AVERAGE PSNR AFTER PERFORMING JSCD ($\overline{PSNR_0}$), AND WIDTH OF THE LAST THREE CRITICAL SEGMENTS FOR LENA.

LOSS RATE	CODEBLOCK SIZE	CS#1			CS#2			CS#3		
		$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH
10%	16×16	30.6	37.5	27	24.6	30.0	40	23.8	24.3	15
	32×32	32.5	37.1		26.9	32.3		26.2	26.8	
	64×64	32.8	37.0		29.6	32.7		27.6	29.5	
20%	16×16	30.2	33.8	17	23.9	30.0	62	23.4	23.7	12
	32×32	32.2	33.5		26.2	32.1		24.3	26.2	
	64×64	32.7	35.6		27.6	32.6		24.3	27.5	
30%	16×16	30.0	30.1	1	30.0	30.0	1	23.8	30.0	73
	32×32	32.3	32.4		32.1	32.3		26.2	32.1	
	64×64	32.7	32.7		32.7	32.7		27.5	32.6	

TABLE 3-VI. AVERAGE PSNR BEFORE JSCD ($\overline{PSNR}$), AVERAGE PSNR AFTER PERFORMING JSCD ($\overline{PSNR_0}$), AND WIDTH OF THE LAST THREE CRITICAL SEGMENTS FOR BARBARA.

LOSS RATE	CODEBLOCK SIZE	CS#1			CS#2			CS#3		
		$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH
10%	16×16	22.2	32.0	93	21.7	21.8	1	21.1	21.6	2
	32×32	22.2	31.6		22.0	22.0		21.0	22.0	
	64×64	22.1	32.1		22.0	22.0		21.0	22.0	
20%	16×16	21.7	29.4	93	20.9	21.6	3	20.0	20.9	2
	32×32	22.0	29.0		20.8	22.0		20.0	20.8	
	64×64	22.0	28.3		20.8	22.0		20.0	20.8	
30%	16×16	22.5	27.5	79	22.0	22.5	8	21.7	22.0	4
	32×32	23.3	27.2		22.5	23.3		22.0	22.5	
	64×64	23.4	27.0		22.5	23.4		22.0	22.5	

TABLE 3-VII. AVERAGE PSNR BEFORE JSCD ($\overline{PSNR}$), AVERAGE PSNR AFTER PERFORMING JSCD ($\overline{PSNR_0}$), AND WIDTH OF THE LAST THREE CRITICAL SEGMENTS FOR GOLDHILL.

LOSS RATE	CODEBLOCK SIZE	CS#1			CS#2			CS#3		
		$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH
10%	16×16	24.7	33.4	75	23.9	24.3	9	23.7	23.8	6
	32×32	26.5	33.0		25.9	26.4		25.3	25.9	
	64×64	27.6	32.1		26.3	27.5		25.2	26.3	
20%	16×16	30.6	31.1	2	29.3	30.6	14	24.4	29.2	55
	32×32	31.2	31.4		30.4	31.2		26.4	30.3	
	64×64	31.9	32.1		30.5	31.7		27.5	29.8	
30%	16×16	29.3	29.4	1	29.2	29.3	2	24.3	29.1	63
	32×32	30.4	30.5		30.2	30.4		26.4	30.1	
	64×64	30.6	30.6		30.5	30.6		27.5	29.0	

TABLE 3-VIII. AVERAGE PSNR BEFORE JSCD ($\overline{PSNR}$), AVERAGE PSNR AFTER PERFORMING JSCD ($\overline{PSNR_0}$), AND WIDTH OF THE LAST THREE CRITICAL SEGMENTS FOR BOAT.

LOSS RATE	CODEBLOCK SIZE	CS#1			CS#2			CS#3		
		$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH
10%	16×16	29.3	33.5	27	22.6	29.0	57	21.9	22.1	7
	32×32	29.0	33.0		23.9	28.8		22.8	23.8	
	64×64	28.9	32.8		24.3	28.7		22.8	24.3	
20%	16×16	22.1	30.5	89	21.1	21.9	7	19.5	21.0	2
	32×32	22.8	31.5		21.0	22.8		19.5	21.0	
	64×64	22.8	30.3		21.0	22.8		19.5	21.0	
30%	16×16	29.0	29.0	1	28.9	29.0	1	22.8	28.9	66
	32×32	28.9	29.1		28.8	28.9		24.5	28.8	
	64×64	28.9	28.9		28.8	28.9		25.5	28.4	

TABLE 3-IX. AVERAGE PSNR BEFORE JSCD ($\overline{PSNR}$), AVERAGE PSNR AFTER PERFORMING JSCD ($\overline{PSNR_0}$), AND WIDTH OF THE LAST THREE CRITICAL SEGMENTS FOR LENA WHEN ULP MATRICES CONSIST OF 255 PACKETS OF 50 BYTES.

LOSS RATE	CODEBLOCK SIZE	CS#1			CS#2			CS#3		
		$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH
10%	16×16	32.2	32.5	1	25.4	32.2	13	24.2	25.2	9
	32×32	32.3	32.4		27.0	32.3		26.3	26.9	
	64×64	33.4	33.6		29.4	33.4		27.5	29.3	
20%	16×16	29.7	29.7	1	24.5	29.7	21	24.3	24.4	3
	32×32	31.5	31.6		26.9	31.5		26.7	26.8	
	64×64	31.9	31.9		28.8	31.9		28.3	28.8	
30%	16×16	25.4	29.2	17	24.4	25.4	2	24.2	24.4	9
	32×32	27.0	30.6		26.9	27.0		26.3	26.9	
	64×64	29.3	30.6		29.0	29.3		27.5	29.0	

TABLE 3-X. AVERAGE PSNR BEFORE JSCD ($\overline{PSNR}$), AVERAGE PSNR AFTER PERFORMING JSCD ($\overline{PSNR_0}$), AND WIDTH OF THE LAST THREE CRITICAL SEGMENTS FOR BARBARA WHEN ULP MATRICES CONSIST OF 255 PACKETS OF 50 BYTES.

LOSS RATE	CODEBLOCK SIZE	CS#1			CS#2			CS#3		
		$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH
10%	16×16	21.9	28.0	43	20.9	21.8	3	20.5	20.9	1
	32×32	22.1	27.7		20.8	22.0		20.4	20.8	
	64×64	22.0	27.6		20.8	22.0		20.4	20.8	
20%	16×16	25.5	26.8	11	21.8	25.4	31	20.7	21.7	4
	32×32	25.3	26.7		22.0	25.3		20.7	22.0	
	64×64	25.3	26.6		22.0	25.3		20.7	22.0	
30%	16×16	23.6	24.3	1	23.4	23.6	2	22.9	23.4	9
	32×32	24.5	24.6		24.4	24.5		23.9	24.4	
	64×64	24.7	24.8		24.6	24.7		24.0	24.6	

TABLE 3-XI. AVERAGE PSNR BEFORE JSCD ($\overline{PSNR}$), AVERAGE PSNR AFTER PERFORMING JSCD ($\overline{PSNR_0}$), AND WIDTH OF THE LAST THREE CRITICAL SEGMENTS FOR LENA WHEN ULP MATRICES CONSIST OF 255 PACKETS OF 150 BYTES.

LOSS RATE	CODEBLOCK SIZE	CS#1			CS#2			CS#3		
		$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH
10%	16×16	24.0	36.3	130	23.3	23.7	11	23.1	23.3	1
	32×32	26.2	36.0		24.3	26.1		24.3	24.3	
	64×64	27.5	35.4		24.3	27.4		24.3	24.3	
20%	16×16	29.4	29.5	1	29.3	29.4	1	29.3	29.3	1
	32×32	29.8	29.9		29.4	29.8		29.4	29.4	
	64×64	31.0	31.0		31.0	31.0		31.0	31.0	
30%	16×16	29.1	29.1	1	29.0	29.1	1	29.0	29.0	1
	32×32	28.5	28.5		28.5	28.5		28.4	28.5	
	64×64	31.0	31.0		31.0	31.0		31.0	31.0	

TABLE 3-XII. AVERAGE PSNR BEFORE JSCD ($\overline{PSNR}$), AVERAGE PSNR AFTER PERFORMING JSCD ($\overline{PSNR_0}$), AND WIDTH OF THE LAST THREE CRITICAL SEGMENTS FOR BARBARA WHEN ULP MATRICES CONSIST OF 255 PACKETS OF 150 BYTES.

LOSS RATE	CODEBLOCK SIZE	CS#1			CS#2			CS#3		
		$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH	$\overline{PSNR}$	$\overline{PSNR_0}$	WIDTH
10%	16×16	30.3	30.4	1	22.7	30.2	119	21.7	22.6	22
	32×32	30.1	30.1		23.6	30.0		22.0	23.5	
	64×64	28.8	29.3		23.8	28.8		22.0	23.7	
20%	16×16	28.4	28.5	1	28.4	28.4	1	23.4	28.4	69
	32×32	27.8	27.9		27.7	27.9		24.6	27.7	
	64×64	27.3	27.4		27.3	27.3		25.0	27.1	
30%	16×16	25.9	26.0	1	25.8	25.9	1	25.8	25.8	1
	32×32	25.8	25.8		25.8	25.8		25.7	25.8	
	64×64	25.8	25.8		25.7	25.8		25.5	25.7	

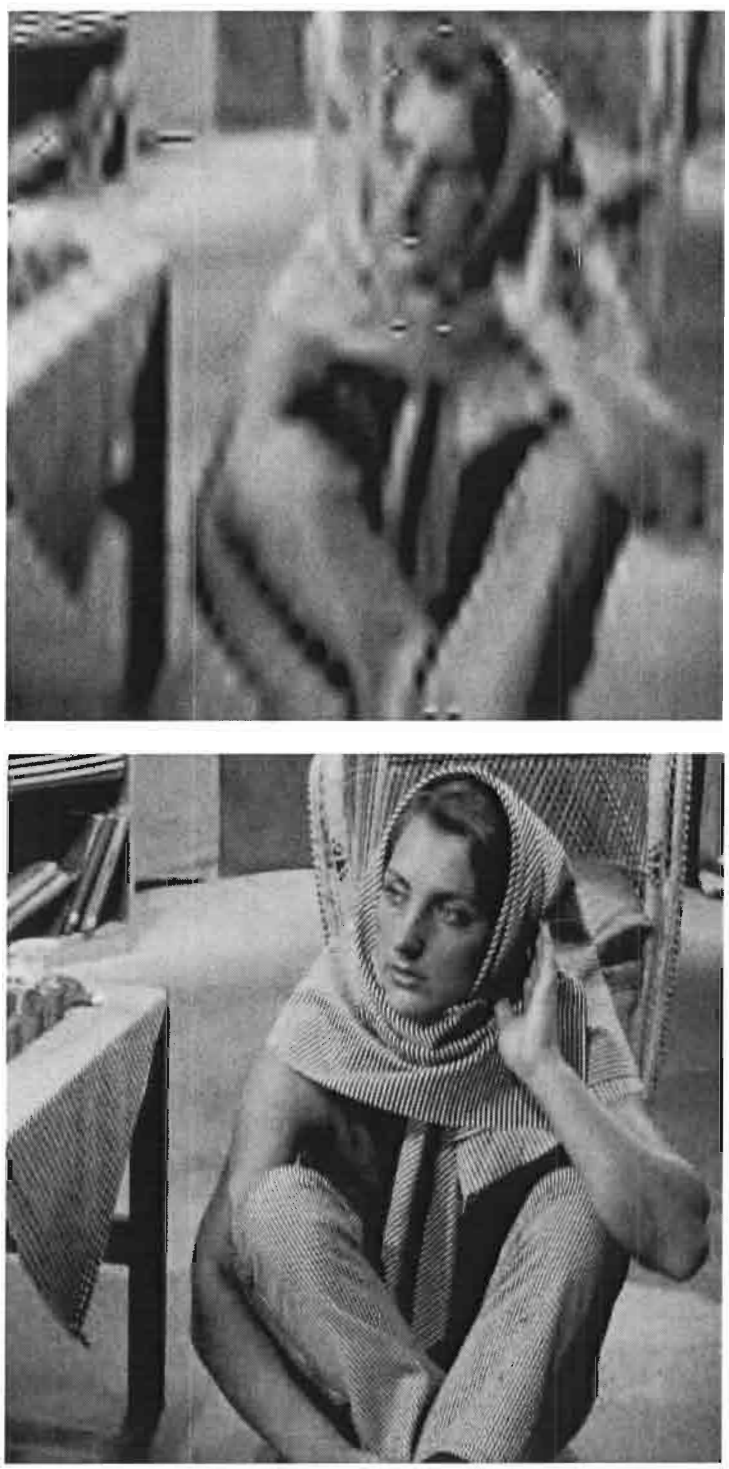

Figure 3-10. Barbara image with 64×64 codeblocks transmitted through a channel with 10% mean loss rate, before JSCD on top (*PSNR* = 22.1dB) and after JSCD on bottom (*PSNR* = 32.4dB).

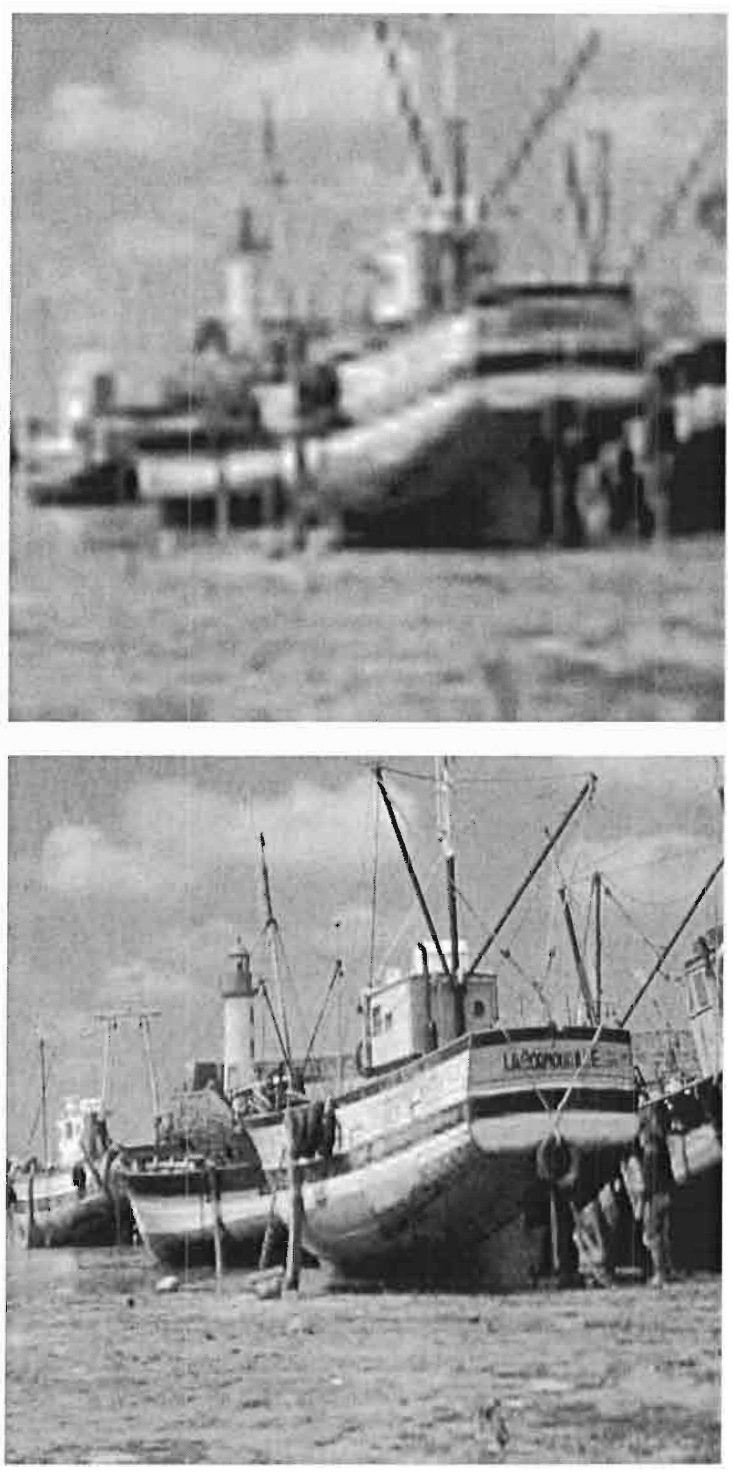

Figure 3-11. Boat image with 64×64 codeblocks transmitted through a channel with 20% mean loss rate, before JSCD on top (*PSNR* = 22.8dB) and after JSCD on bottom (*PSNR* = 30.7dB).

examples, when JSCD is applied compared to separate source and channel decoding. Note that the PSNR values reported for some of the ULP matrices designed for lower mean loss rates are lower than the corresponding PSNR values at higher mean loss rate (e.g., Table 3-VI, CS#2, loss rate at 20% and 30%). One may find these results unreasonable by missing the fact that the compared CS's may contain different parts of the JPEG2000 bitstream because their corresponding ULP matrices are different.

Furthermore, as the tables present, in contrast with the fact that the quality of decoded JPEG2000 images at a given source rate should ideally increase using larger codeblocks [1], in a few cases (e.g., Table 3-VI, CS#1 at 20% loss rate) the trend does not conform to the expectation. These abrupt fluctuations have two reasons. The first reason is the effect of the restriction we imposed on the algorithm to prevent the decoder from correcting the undetected wrong guesses far back in the bitstream. As we mentioned earlier in this chapter, the joint decoder is only allowed to correct the wrong guesses located just a few columns (e.g., three columns) before the current column. If it becomes necessary to correct a wrong guess beyond this boundary, the joint decoding will be terminated. Although this constraint could be advantageous for controlling the execution time, in occasional cases where it terminates the JSCD process too early in the CS, most of the source information in that segment will remain corrupted. As a result, the ultimate image quality that JSCD could deliver will not be achieved. The second reason is that JSCD is more prone to leave undetected wrong guesses in the images with larger codeblocks. Both of these circumstances are less likely to happen when smaller codeblocks are used; because smaller codeblocks produce shorter coding passes, which essentially reduce the chance of missing the wrong guess, by providing better error localization [10].

When the joint decoder has to search through numerous candidates among the RS codewords, the basic JSCD can become extremely time-consuming. Our simulations on a standard desktop PC show that, to restore a relatively wide critical segment, the joint decoder may need even more than five minutes, much longer than an ordinary image decoding time. In the next chapter, we address this downside and provide some solutions to speed up the process.

CHAPTER 4: SPEED IMPROVEMENTS

Although the basic JSCD described in the previous chapter is able to deliver significant quality improvements in some cases, the search for the correct values of the erased symbols can be very time-consuming. In this chapter, we describe how the search process may be accelerated. The work presented in this chapter is also described in [17] and [18].

4.1. SOLUTIONS

4.1.1. USING SMALLER CODEBLOCKS

One possible method to accelerate JSCD is to use smaller codeblocks, which in turn causes shorter coding passes. These shorter coding passes are less likely to spread into more than one column of the ULP matrix. Moreover, JPEG2000 ER features work better for smaller codeblocks because the arithmetic decoder is resynchronized more frequently [22]. As a result, when the JSCD of the next RS codeword begins, a wrong guess value can rarely remain undetected. However, it is important to notice that smaller codeblocks cause reduction in compression efficiency and lead to lower image quality for a given bit budget, as shown in tables 3-I to 3-IV. In fact, this quality degradation is the cost we pay to speed up the proposed JSCD procedure.

4.1.2. USING FINER ERROR LOCALIZATION

When the JPEG2000 bitstream is generated with ER features turned on, the decoder can not only indicate the coding pass in which the error has occurred, but also signal the location of the symbol (byte) in the coding pass where it becomes aware of the error. The following example explains how this information can expedite the search by detecting the earliest wrong guess more quickly.

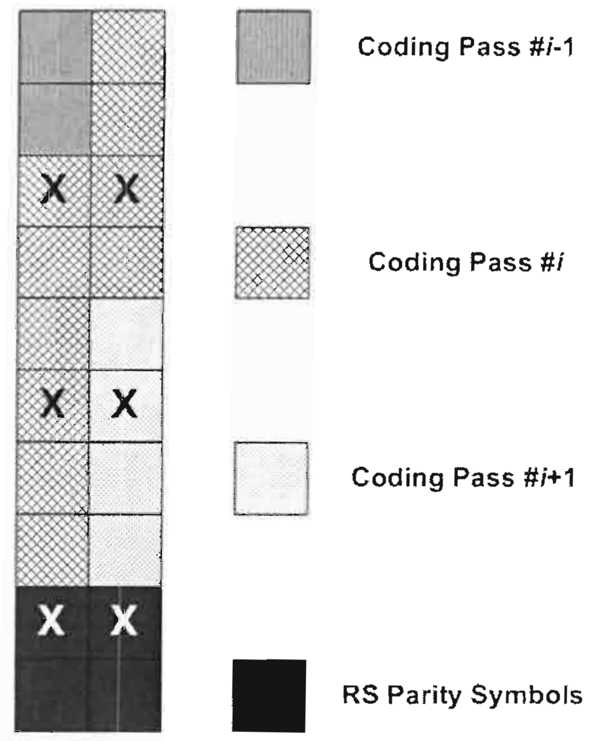

Figure 4-1. An arrangement of coding passes in a critical segment for which the finer error localization increases the speed of the JSCD. Erased symbols are marked by X.

Suppose that the joint decoder is trying to decode coding pass #*i* depicted in Figure 4-1. Since each column-wise piece of this coding pass contains at least one of the erased symbols (marked by crosses), successful decoding of this coding pass depends on the values of multiple erased symbols. Suppose that the first row of the erased symbols are guessed during the JSCD. If the JPEG2000 decoding failure occurs on or after the second guessed symbol, the guess values in either of the columns might be wrong, so the decoder may need to change both. But if the decoder complains of an error before it passes the second guessed symbol, further verification of the possible guess values for this symbol is unnecessary since it is known that the guess value in the first column is incorrect.

4.1.3. AUXILIARY BITS

As shown in Section 3.1, by filling any set of *d* erased symbols in a critical segment column with arbitrary values, the RS decoder is able to fill the other erased locations and generate a complete and valid RS codeword. Therefore, the information carried by non-erased symbols is not helpful to reduce the number of possible guesses. Consequently, accelerating the JSCD beyond what the previous approaches can offer requires some extra information being available at the decoder side. Some side information transmitted from the encoder side, which requires only a relatively small additional bit budget, can be used to facilitate the joint decoding at the receiver.

One possible choice for this side information is a collection of bit values from the RS symbols, for example, the first few most significant bits of each symbol. When transmitted error-free, by reducing the number of bits that need to be guessed, these *auxiliary* bits can help to shrink the search space for JSCD, and hence accelerate the process. Using only one auxiliary bit per symbol, for instance, can reduce the number of possible guesses by a factor of 2^d. Furthermore, auxiliary bits from different symbols in a particular RS codeword can be used to verify the validity of the guesses without performing the JPEG2000 decoding, which is the most time-consuming part of the guessing iterations.

Figure 4-2 illustrates a schematic diagram of the joint decoder which exploits auxiliary bits. As the diagram shows, in the improved JSCD the time-consuming JPEG2000 decoding is performed subsequent to the RS decoding only if the result of auxiliary bit verification is positive. If there are erased symbols which have received auxiliary bits, excluding the guessed symbol(s), any mismatch between their auxiliary bits and the corresponding bits of their restored values indicates that the guess values have been wrong, and there is no need to further call the JPEG2000 decoder to verify the guess. Consequently, the overall computational load of the JSCD would be significantly reduced.

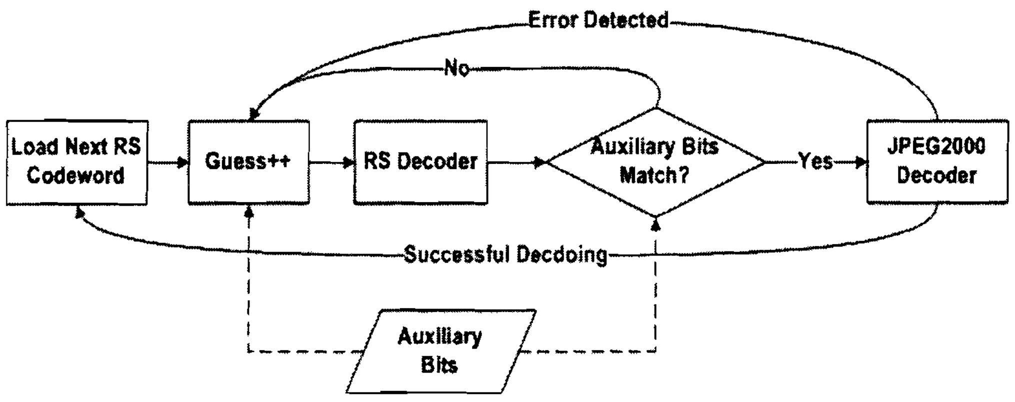

Figure 4-2. Simplified diagram of the joint decoder when auxiliary bits are available.

The auxiliary bits are helpful to decrease the extra decoding time required for JSCD significantly; however, we should also consider the cost of using them. For example, if we use the common RS(255, *k*) codes in the ULP scheme, each RS symbol is 8 bits long. Assigning even one auxiliary bit to each of these symbols would result in 12.5% increase in the total number of transmitted bits. Using such a large number of auxiliary bits might be questionable because not only it would be difficult to guarantee their error-free transmission, but also, part of this bit budget could have been spent on parity symbols in the first place. Therefore, it is necessary to assign the auxiliary bits selectively.

We can choose to assign auxiliary bits only to the first few packets, because, for a given number of packet erasures, the first few erasures that erase the symbols whose values we should later guess, are more likely to occur during the first few packet transmissions rather than later transmissions. For a precise explanation, suppose that there are *e* erasures among *N* packets. Then, the probability of having *t* erased packets in the first k_0 packets is:

$$Q_{N,e}(t) = \binom{N-k_0}{e-t}\binom{k_0}{t} \Big/ \binom{N}{e}. \qquad (4.1)$$

Depending on the desirable value of *t*, one can choose k_0 such that the probability of having at least *t* erasures in the first k_0 packets (i.e., $F_{N,k_0,e}(t) = \sum_{\tau \geq t} Q_{N,k_0,e}(\tau)$) is above a safe threshold. The cases where $t > 1$

are also considered because they show how often there are multiple erased symbols among the first k_0 symbols, and thus it is possible to detect the incorrect guess values by comparing the available auxiliary bits and their corresponding bits in the restored symbols. Figure 4-3 shows how F varies by changing k_0 and t, for e = 39 and N = 255. For instance, the probability of having at least one erased packet in the first 20 packets is more than 95%. Since auxiliary bits are used only when their corresponding symbols are erased, and the probability of the first erased packet being within the first few packets is high, placing auxiliary bits in this way ensures that they will be used with high probability and there would be no

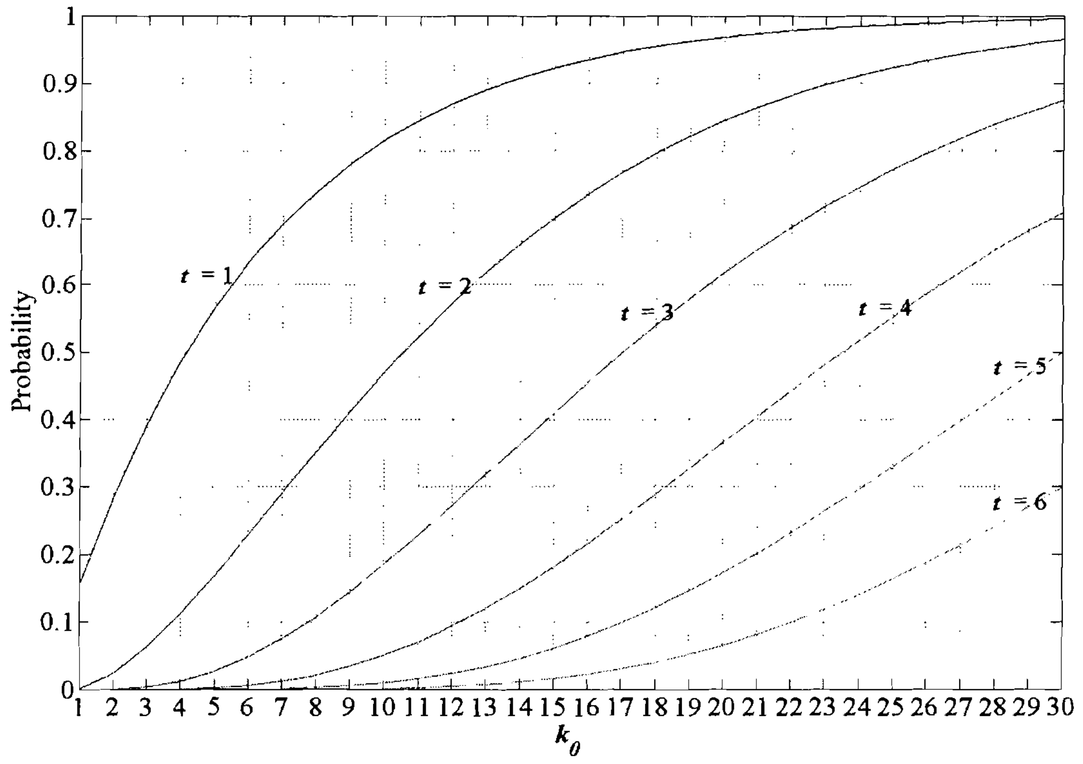

Figure 4-3. Trends of the probability function, F, for different values of t and k_0 when e = 39 of the total of N = 255 packets are erased.

need to place them elsewhere. This way, we can approach the performance obtained when auxiliary bits are assigned to all symbols, but without using as many bits.

We can further reduce the number of auxiliary bits by assigning them only to the symbols that belong to coding passes whose decoding takes a longer time. In Figure 4-1, suppose that symbols in one of the erased packets (say the first erased packet) are chosen to be guessed. Depending on the number of columns in the critical segment that contain erased symbols of a certain coding pass, the joint decoder may have to generate guess values for multiple erased symbols which belong to different columns. For example, to decode coding pass #*i* in Figure 4-1 perfectly, the joint decoder should fill in both of the erased symbols in the first erased packet with correct values. In such cases, concatenation of these multiple symbols is considered as a longer binary word whose value the joint decoder guesses sequentially, starting from zero. If one of the corresponding erased symbols is filled with a wrong guess, an unsuccessful search may be perpetuated through all of the possible values that the less significant part of the binary word can take. Therefore, on average, it can be expected that in these cases, the decoding time of the coding pass rises sharply.

Since the decoding time is highly dependent on JPEG2000 error detection performance, there is no simple way to identify all of the coding passes that require more time to be decoded. However, we can argue that coding passes that have more than one symbol in the first erased packet are more likely to lengthen the JSCD. Therefore, for each coding pass, we choose to assign auxiliary bits only to the symbols that are adjacent to another symbol from the same coding pass located in the same network packet. For example, in Figure 4-1, some auxiliary bits would be assigned for the symbols of coding pass #*i* in the third and the fourth row, but the symbols of the same coding pass in the sixth row would not receive any auxiliary bits. This policy provides a compromise between improving the decoding speed and the number of extra bits required. Therefore, by distributing auxiliary bits only among these particular kinds of coding passes, the required extra bit budget would be reduced substantially, while the joint decoding speed still shows significant improvements.

If ρ denotes the ratio of the required auxiliary bit budget to the main bit budget, we have the following upper bound:

$$\rho \le \lambda \times \frac{k_0 \times a}{\cancel{N} \times m} \times \frac{\cancel{N}}{N - e_{\max}} = \frac{\lambda k_0 a}{m\left(N - e_{\max}\right)}, \qquad (4.2)$$

in which λ is the proportion of columns in the ULP matrix that contain the coding passes chosen to receive auxiliary bits, a is the number of auxiliary bits assigned to each symbol, m is the number of bits in each RS symbol, and e_{max} is the maximum number of erasures that can be recovered by the JSCD. Note that in (4.2) we have also included the cost of protecting the auxiliary bits by assigning RS codes of rate $(N\text{-}e_{max})/N$ to them. Therefore, the entire required extra bit budget is counted in this upper bound.

4.2. RESULTS

The setup settings that we mentioned in Section 3.2, such as bitrate of 0.7782 bpp, the ERTERM+RESTART encoding mode, and use of SOP and EPH markers, are also used for simulations of this chapter. In addition, in the accelerated JSCD method we considered one, two, and three auxiliary bits for the symbols which need it. The numbers reported in tables 4-V to 4-VIII are average outcomes of 100 simulations run on a desktop PC with an Intel® Core™ 2 Duo 2.13 GHz CPU and 2GB of RAM.

The auxiliary bits and the other improvement tools mentioned in Section 4.1 are essentially exploited to boost the speed of JSCD; however, using these techniques can also deliver a small extra quality enhancement. These minor improvements occur when a coding pass, especially the last coding pass, of a particular codeblock contains a wrong guess, but JPEG2000 decoder fails to detect it during the basic JSCD and finishes decoding the codeblock. Consequently, the source decoder would eventually complain for an error induced by RS decoding to the next codeblock(s). However, due to the restrictions that Kakadu [1] imposes, in our implementation, we cannot simply decode a coding pass of an already-decoded codeblock. As a result, the decoded codeblock will remain corrupted, even if we find the correct value that corresponds to the last undetected error, using the information of the next codeblock. Nonetheless, wherever the auxiliary bits are applied, they can reduce the chance of having undetected wrong guesses, by shrinking the set of all possible guesses. For example, suppose that the original value of an erased symbol is 135. Also, assume that there is a wrong guess for this symbol with a value of less than 128, which JPEG2000 cannot detect. In this case, even by using only one auxiliary bit, the wrong undetectable guess will be skipped because the search begins at 128. Hence, the auxiliary bits can slightly improve the quality of the output.

The simulation results in tables 4-I to 4-IV show that in most cases, especially when codeblock size is smaller than 64×64, the accelerated JSCD, which fully exploits the error detection information and the auxiliary bits, does not increase the PSNR of the decoded image. However, because of the reasons mentioned above, in a few cases there is up to 1.4 dB gain in the PSNR when the improved JSCD is applied (e.g., Table 4-III, CS#3, 64×64 codeblock size, at 30% loss rate).

In tables 4-V to 4-VIII, the average time costs of basic JSCD ($\overline{\Delta T_0}$) and improved JSCD with different number of auxiliary bits ($\overline{\Delta T^*(a)}$'s) are reported in seconds, with one decimal point precision. The average time that the separate source and channel decoding requires ($\bar{T}$) was equal to 0.1s consistently. Note that we reported these numbers in one decimal point precision because limited number of simulations for each case, and the fluctuations in the computer's memory and CPU usage caused by background programs cause errors with magnitude of hundredths of a second.

Table 4-I. Average PSNR Performance (in dB) of Basic JSCD, and Improved JSCD with a = 0, 1, 2 and 3 Auxiliary Bits on Lena.

LOSS RATE	CODEBLOCK SIZE	CS#1					CS#2					CS#3				
		$\overline{PSNR_0}$	$\overline{PSNR^*}(a)$				$\overline{PSNR_0}$	$\overline{PSNR^*}(a)$				$\overline{PSNR_0}$	$\overline{PSNR^*}(a)$			
			$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$
10%	16×16	37.5	37.5	37.5	37.5	37.5	30.0	30.0	30.0	30.0	30.0	24.3	24.3	24.3	24.3	24.3
	32×32	37.1	37.1	37.1	37.1	37.1	32.3	32.3	32.3	32.3	32.3	26.8	26.8	26.8	26.8	26.8
	64×64	37.0	37.0	37.1	37.1	37.1	32.7	32.7	32.7	32.7	32.7	29.5	29.5	29.5	29.5	29.5
20%	16×16	33.8	33.8	33.8	33.8	33.8	30.0	30.0	30.0	30.0	30.0	23.7	23.7	23.7	23.7	23.7
	32×32	33.5	33.5	33.5	33.5	33.5	32.1	32.1	32.1	32.1	32.1	26.2	26.2	26.2	26.2	26.2
	64×64	35.6	35.6	35.6	35.6	35.6	32.6	32.6	32.7	32.7	32.7	27.5	27.5	27.5	27.5	27.5
30%	16×16	30.1	30.1	30.1	30.1	30.1	30.0	30.0	30.0	30.0	30.0	30.0	30.0	30.0	30.0	30.0
	32×32	32.4	32.4	32.4	32.4	32.4	32.3	32.3	32.3	32.3	32.3	32.1	32.1	32.1	32.1	32.1
	64×64	32.7	32.7	32.7	32.7	32.7	32.7	32.7	32.7	32.7	32.7	32.6	32.6	32.6	32.6	32.7

Table 4-II. Average PSNR Performance (in dB) of Basic JSCD, and Improved JSCD with a = 0, 1, 2 and 3 Auxiliary Bits on Barbara.

LOSS RATE	CODEBLOCK SIZE	CS#1					CS#2					CS#3				
		$\overline{PSNR_0}$	$\overline{PSNR^*}(a)$				$\overline{PSNR_0}$	$\overline{PSNR^*}(a)$				$\overline{PSNR_0}$	$\overline{PSNR^*}(a)$			
			$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$
10%	16×16	32.0	32.0	32.0	32.0	32.0	21.8	21.8	21.8	21.8	21.8	21.6	21.6	21.6	21.6	21.6
	32×32	31.6	31.6	31.6	31.6	31.6	22.0	22.0	22.0	22.0	22.0	22.0	22.0	22.0	22.0	22.0
	64×64	32.1	32.1	32.1	32.1	32.1	22.0	22.0	22.0	22.0	22.0	22.0	22.0	22.0	22.0	22.0
20%	16×16	29.4	29.4	29.4	29.4	29.4	21.6	21.6	21.6	21.6	21.6	20.9	20.9	20.9	20.9	20.9
	32×32	29.0	29.0	29.0	29.0	29.0	22.0	22.0	22.0	22.0	22.0	20.8	20.8	20.8	20.8	20.8
	64×64	28.3	28.3	28.4	28.4	28.4	22.0	22.0	22.0	22.0	22.0	20.8	20.8	20.8	20.8	20.8
30%	16×16	27.5	27.5	27.5	27.5	27.5	22.5	22.5	22.5	22.5	22.5	22.0	22.0	22.0	22.0	22.0
	32×32	27.2	27.2	27.2	27.2	27.2	23.3	23.3	23.3	23.3	23.3	22.5	22.5	22.5	22.5	22.5
	64×64	27.0	27.0	27.1	27.1	27.1	23.4	23.4	23.4	23.4	23.4	22.5	22.5	22.5	22.5	22.5

TABLE 4-III. AVERAGE PSNR PERFORMANCE (IN dB) OF BASIC JSCD, AND IMPROVED JSCD WITH a = 0, 1, 2 AND 3 AUXILIARY BITS ON GOLDHILL.

LOSS RATE	CODEBLOCK SIZE	CS#1					CS#2					CS#3				
		$\overline{PSNR_0}$	$\overline{PSNR^*}(a)$				$\overline{PSNR_0}$	$\overline{PSNR^*}(a)$				$\overline{PSNR_0}$	$\overline{PSNR^*}(a)$			
			$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$
10%	16×16	33.4	33.4	33.4	33.4	33.4	24.3	24.3	24.3	24.3	24.3	23.8	23.8	23.8	23.8	23.8
	32×32	33.0	33.0	33.0	33.0	33.0	26.4	26.4	26.4	26.4	26.4	25.9	25.9	25.9	25.9	25.9
	64×64	32.1	32.1	33.0	33.2	33.3	27.5	27.5	27.5	27.5	27.5	26.3	26.3	26.3	26.3	26.3
20%	16×16	31.1	31.1	31.1	31.1	31.1	30.6	30.6	30.6	30.6	30.6	29.2	29.2	29.2	29.2	29.2
	32×32	31.4	31.4	31.4	31.4	31.4	31.2	31.2	31.2	31.2	31.2	30.3	30.3	30.3	30.3	30.3
	64×64	32.1	32.1	32.1	32.1	32.1	31.7	31.7	31.9	31.9	31.9	29.8	29.8	30.4	30.5	30.5
30%	16×16	29.4	29.4	29.4	29.4	29.4	29.3	29.3	29.3	29.3	29.3	29.1	29.1	29.1	29.1	29.1
	32×32	30.5	30.5	30.5	30.5	30.5	30.4	30.4	30.4	30.4	30.4	30.1	30.1	30.1	30.1	30.1
	64×64	30.6	30.6	30.6	30.6	30.6	30.6	30.6	30.6	30.6	30.6	29.0	29.0	30.3	30.3	30.4

TABLE 4-IV. AVERAGE PSNR PERFORMANCE (IN dB) OF BASIC JSCD, AND IMPROVED JSCD WITH a = 0, 1, 2 AND 3 AUXILIARY BITS ON BOAT.

LOSS RATE	CODEBLOCK SIZE	CS#1					CS#2					CS#3				
		$\overline{PSNR_0}$	$\overline{PSNR^*}(a)$				$\overline{PSNR_0}$	$\overline{PSNR^*}(a)$				$\overline{PSNR_0}$	$\overline{PSNR^*}(a)$			
			$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$
10%	16×16	33.5	33.5	33.5	33.5	33.5	29.0	29.0	29.0	29.0	29.0	22.1	22.1	22.1	22.1	22.1
	32×32	33.0	33.0	33.0	33.0	33.0	28.8	28.8	28.8	28.8	28.8	23.8	23.8	23.8	23.8	23.8
	64×64	32.8	32.8	32.9	33.0	33.0	28.7	28.7	28.8	28.8	28.8	24.3	24.3	24.3	24.3	24.3
20%	16×16	30.5	30.5	30.5	30.5	30.5	21.9	21.9	21.9	21.9	21.9	21.0	21.0	21.0	21.0	21.0
	32×32	31.5	31.5	31.5	31.5	31.5	22.8	22.8	22.8	22.8	22.8	21.0	21.0	21.0	21.0	21.0
	64×64	30.3	30.3	30.9	31.0	31.1	22.8	22.8	22.8	22.8	22.8	21.0	21.0	21.0	21.0	21.0
30%	16×16	29.0	29.0	29.0	29.0	29.0	29.0	29.0	29.0	29.0	29.0	28.9	28.9	28.9	28.9	28.9
	32×32	29.1	29.1	29.1	29.1	29.1	28.9	28.9	28.9	28.9	28.9	28.8	28.8	28.8	28.8	28.8
	64×64	28.9	28.9	28.9	28.9	28.9	28.9	28.9	28.9	28.9	28.9	28.4	28.4	28.8	28.8	28.8

The results show that the speedup techniques can reduce the time required for the JSCD by up to two orders of magnitude, even for 64×64 codeblocks (e.g., Table 4-VII, CS#3, 64×64 codeblock size, at 20% loss rate). However, to achieve such acceleration, the amount of increase in the assigned bit budget due to the addition of the auxiliary bits must be affordable. Also, note that the JSCD still takes much more time than separate source and channel decoding ($\bar{T} = 0.1\text{s}$).

As stated earlier, using smaller codeblocks can cause fast joint decoding, but complexity reduction brought by this approach may be offset by the reduction in PSNR. For most of the cases reported in tables 4-V to 4-VIII the joint decoding speed increases with the decrease in the size of the codeblocks, except for some of the very narrow critical segments (e.g., Table 4-V, CS#1 at 30% loss rate). In these cases, the JSCD takes longer time when smaller codeblocks are used because the critical segment is just large enough to accommodate a fraction of the bitstream of a large codeblock, but perhaps bitstreams of a few smaller codeblocks can be placed in the critical segment. Therefore, for larger codeblocks, only one undetected wrong guess can cause early termination of the joint decoding. If smaller codeblocks are used, however, there is more chance that a complete search for the correct guess values would be conducted; hence, the decoded image may end up with a higher PSNR than when larger codeblocks are used.

TABLE 4-V. EXTRA DECODING TIME (IN SECONDS) IN CASE OF LENA FOR BASIC JSCD, AND IMPROVED JSCD WITH a = 0, 1, 2 AND 3 AUXILIARY BITS.

LOSS RATE	CODEBLOCK SIZE	CS#1					CS#2					CS#3				
		$\overline{\Delta T_0}$	$\overline{\Delta T^*}(a)$				$\overline{\Delta T_0}$	$\overline{\Delta T^*}(a)$				$\overline{\Delta T_0}$	$\overline{\Delta T^*}(a)$			
			$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$
10%	16×16	0.5	0.5	0.5	0.5	0.5	0.8	0.8	0.8	0.8	0.8	0.3	0.3	0.3	0.3	0.3
	32×32	0.6	0.6	0.6	0.6	0.6	0.9	0.9	0.9	0.9	0.9	0.3	0.3	0.3	0.3	0.3
	64×64	1.5	1.0	0.8	0.8	0.8	10.9	6.7	1.6	1.5	1.4	0.6	0.6	0.6	0.6	0.6
20%	16×16	0.7	0.7	0.7	0.7	0.7	2.8	2.7	2.7	2.7	2.7	0.5	0.5	0.5	0.5	0.5
	32×32	0.7	0.7	0.7	0.7	0.7	2.8	2.8	2.8	2.8	2.8	0.5	0.5	0.5	0.5	0.5
	64×64	1.7	1.3	1.0	0.9	0.9	62.3	34.2	5.8	3.6	3.2	0.6	0.6	0.6	0.6	0.6
30%	16×16	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	5.1	5.0	5.0	5.0	5.0
	32×32	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	5.2	5.3	5.3	5.3	5.2
	64×64	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1	124.8	68.0	7.3	6.5	6.3

TABLE 4-VI. EXTRA DECODING TIME (IN SECONDS) IN CASE OF BARBARA FOR BASIC JSCD, AND IMPROVED JSCD WITH a = 0, 1, 2 AND 3 AUXILIARY BITS.

LOSS RATE	CODEBLOCK SIZE	CS#1					CS#2					CS#3				
		$\overline{\Delta T_0}$	$\overline{\Delta T^*}(a)$				$\overline{\Delta T_0}$	$\overline{\Delta T^*}(a)$				$\overline{\Delta T_0}$	$\overline{\Delta T^*}(a)$			
			$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$
10%	16×16	1.9	1.9	1.8	1.9	1.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	32×32	2.0	2.0	2.0	2.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	64×64	34.1	24.8	5.8	4.3	4.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20%	16×16	3.9	3.8	3.8	3.8	3.8	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	32×32	4.0	3.9	3.9	3.9	4.0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	64×64	53.4	40.7	8.0	7.4	7.3	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
30%	16×16	5.5	5.5	5.5	5.5	5.5	0.6	0.6	0.6	0.6	0.6	0.3	0.3	0.3	0.3	0.3
	32×32	5.7	5.7	5.7	5.7	5.7	0.6	0.6	0.6	0.6	0.6	0.2	0.2	0.2	0.2	0.2
	64×64	93.3	69.2	11.9	7.4	6.3	22.8	14.3	0.6	0.5	0.5	0.2	0.2	0.2	0.2	0.2

TABLE 4-VII. EXTRA DECODING TIME (IN SECONDS) IN CASE OF GOLDHILL FOR BASIC JSCD, AND IMPROVED JSCD WITH a = 0, 1, 2 AND 3 AUXILIARY BITS.

LOSS RATE	CODEBLOCK SIZE	CS#1					CS#2					CS#3				
		$\overline{\Delta T_0}$	$\overline{\Delta T^*}(a)$				$\overline{\Delta T_0}$	$\overline{\Delta T^*}(a)$				$\overline{\Delta T_0}$	$\overline{\Delta T^*}(a)$			
			$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$
10%	16×16	1.5	1.5	1.5	1.5	1.5	0.2	0.2	0.2	0.2	0.2	0.2	0.1	0.1	0.1	0.1
	32×32	1.7	1.7	1.7	1.7	1.7	0.2	0.2	0.2	0.2	0.2	0.2	0.1	0.1	0.1	0.1
	64×64	52.5	28.9	5.3	3.1	2.7	12.5	7.4	0.9	0.4	0.3	0.2	0.2	0.2	0.2	0.2
20%	16×16	0.1	0.1	0.1	0.1	0.1	0.6	0.6	0.6	0.6	0.6	2.3	2.3	2.3	2.3	2.3
	32×32	0.1	0.1	0.1	0.1	0.1	0.6	0.6	0.6	0.6	0.6	2.6	2.5	2.5	2.5	2.5
	64×64	0.1	0.1	0.1	0.1	0.1	34.3	29.1	1.1	0.6	0.5	339.7	72.6	7.1	3.2	2.7
30%	16×16	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	4.6	4.6	4.7	4.6	4.6
	32×32	0.0	0.0	0.0	0.0	0.0	0.2	0.1	0.1	0.1	0.1	4.7	4.7	4.7	4.7	4.7
	64×64	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2	233.5	151.4	6.9	5.3	5.0

TABLE 4-VIII. EXTRA DECODING TIME (IN SECONDS) IN CASE OF BOAT FOR BASIC JSCD, AND IMPROVED JSCD WITH a = 0, 1, 2 AND 3 AUXILIARY BITS.

LOSS RATE	CODEBLOCK SIZE	CS#1					CS#2					CS#3				
		$\overline{\Delta T_0}$	$\overline{\Delta T^*}(a)$				$\overline{\Delta T_0}$	$\overline{\Delta T^*}(a)$				$\overline{\Delta T_0}$	$\overline{\Delta T^*}(a)$			
			$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$		$a=0$	$a=1$	$a=2$	$a=3$
10%	16×16	0.5	0.5	0.5	0.5	0.5	1.2	1.2	1.2	1.2	1.2	0.2	0.2	0.2	0.2	0.2
	32×32	0.6	0.6	0.6	0.6	0.6	1.3	1.3	1.3	1.3	1.3	0.2	0.2	0.2	0.2	0.2
	64×64	13.6	10.9	2.9	2.3	2.2	12.7	8.3	3.2	2.5	2.4	0.2	0.2	0.2	0.2	0.2
20%	16×16	3.8	3.7	3.7	3.7	3.7	0.3	0.3	0.3	0.3	0.3	0.1	0.1	0.1	0.1	0.1
	32×32	3.8	3.8	3.8	3.8	3.8	0.3	0.3	0.3	0.3	0.3	0.1	0.1	0.1	0.1	0.1
	64×64	80.3	39.8	9.5	7.0	6.5	0.3	0.3	0.3	0.3	0.3	0.1	0.1	0.1	0.1	0.1
30%	16×16	0.1	0.1	0.1	0.1	0.1	0.0	0.0	0.0	0.0	0.0	4.9	4.9	4.9	4.9	4.9
	32×32	0.1	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1	5.1	5.1	5.1	5.1	5.1
	64×64	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	151.5	99.1	15.7	11.7	10.6

Since JPEG2000 decoding consumes most of the joint decoding time, the average number of times that the JPEG2000 decoder detects an error can be used as a measure of the complexity of the proposed JSCD methods. This is shown in Figure 4-4, where the height of each bar indicates how much reduction in the average number of JPEG2000 error detections is achieved compared to basic JSCD. The number of auxiliary bits used in the improved JSCD is shown on the horizontal axis under each group of bars. Evidently, utilization of auxiliary bits has significantly reduced the number of error detections by the JPEG2000 decoder, up to 99% in some cases. However, when this number becomes sufficiently close to a certain threshold, further use of auxiliary bits cannot bring much improvement. For example, we can see in Figure 4-4 that the difference between the results at a = 1, 2 and 3 is relatively small. In the case of *Lena*, auxiliary bits have not improved the speed as much as they have in other cases. The moderate improvement in this particular case is because most of the error detections by JPEG2000 decoder are due to the errors occurring in the symbols that have not received auxiliary bits. Therefore, adding to the number of auxiliary bits would not reduce the size of the search space significantly. Nevertheless, the trends imply that the improved JSCD methods often avoid many of the wrong guesses which are examined by JPEG2000 decoder in the basic JSCD.

The test cases shown in Figure 4-4 suggest that using more than one auxiliary bit does not seem to offer further significant acceleration of the JSCD. Hence, to estimate the auxiliary bit budget needed for reasonable JSCD acceleration, we set a = 1 to compute the upper bound provided by (4.2). Also for the test images and the employed ULP matrix configurations, we have $\lambda \leq 0.2$ and $e_{\max} \leq 105$. Thus (4.2) results in, $\rho < 0.42\%$, which means that with at most 0.42% increase in the rate budget, the improved JSCD can be performed reasonably faster than its basic counterpart. This amount of extra information can all be put together in just one additional column to be added to the utilized ULP matrix.

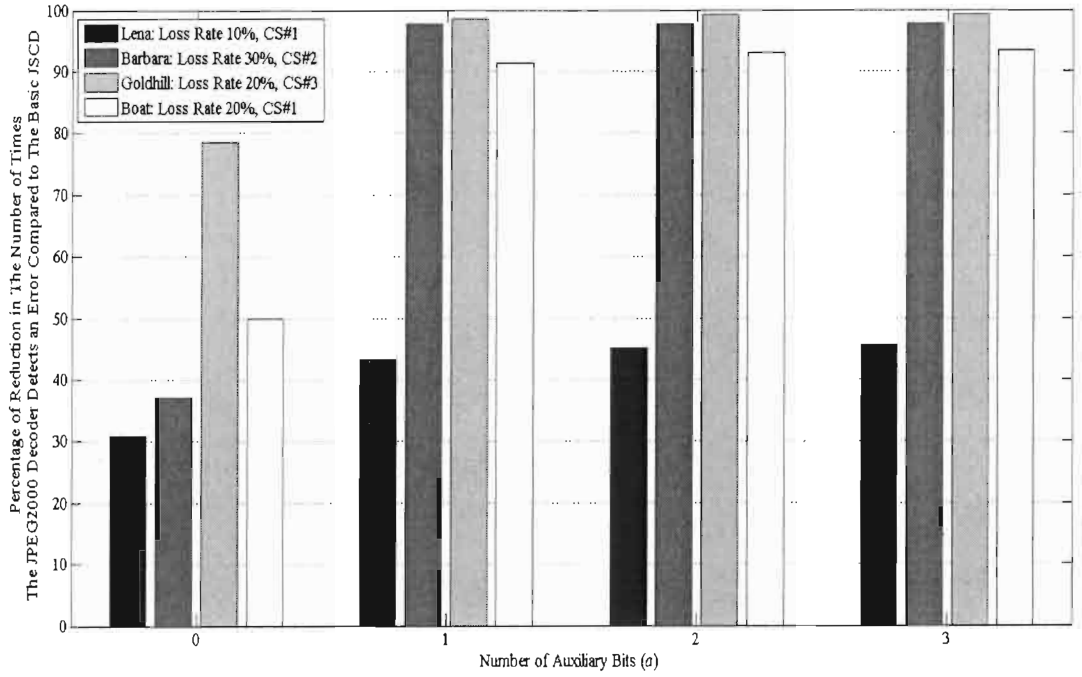

Figure 4-4. Percentage reduction in the number of times the JPEG2000 decoder has detected an error using a = 0, 1, 2, and 3 auxiliary bits, with repect to the basic JSCD.

CHAPTER 5: COMPLEXITY MODEL

To aid the JSCD design, we develop a model to estimate the number of guesses verified during JSCD. The number of required verifications is a crucial design factor, which provides a measure of trade-off between the speed of the JSCD, and the number of auxiliary bits spent. Note that the proposed complexity model only takes the availability of the auxiliary bits into account. In other words, intricate effects of using finer error localization and checking for mismatch in auxiliary bits, explained in sections 4.1.2 and 4.1.3, are excluded in order to obtain a tractable model. Also it is assumed that the first erased packet is always among the first few packets among which the auxiliary bits are distributed. This chapter covers part of our work in [17].

5.1. THE MODEL

For a given pattern of coding passes in the ULP matrix, and a given location for the first erasure, say the *i*-th packet, denote the set of all symbols of the coding pass *c* in the *i*-th row of the existing critical segment by $V_i(c)=\{v_{i,1}(c),v_{i,2}(c),\cdots\}$, where $v_{i,j}(c)$ is the *j*–th leftmost element of $V_i(c)$, with respect to its location in the ULP structure. Figure 5-1 shows how the elements of the set $V_i(c)$ can be determined. Obviously, this set might be empty for many choices of *c* and *i*. In the following, we assume that the JPEG2000 decoder can detect errors caused by a wrong guess inside a coding pass, before it begins to verify the guesses inside any of the subsequent coding passes. This assumption may not always hold, because there may be coding passes that are noticeably less sensitive to errors [10], but it helps us arrive at a desirably simple model.

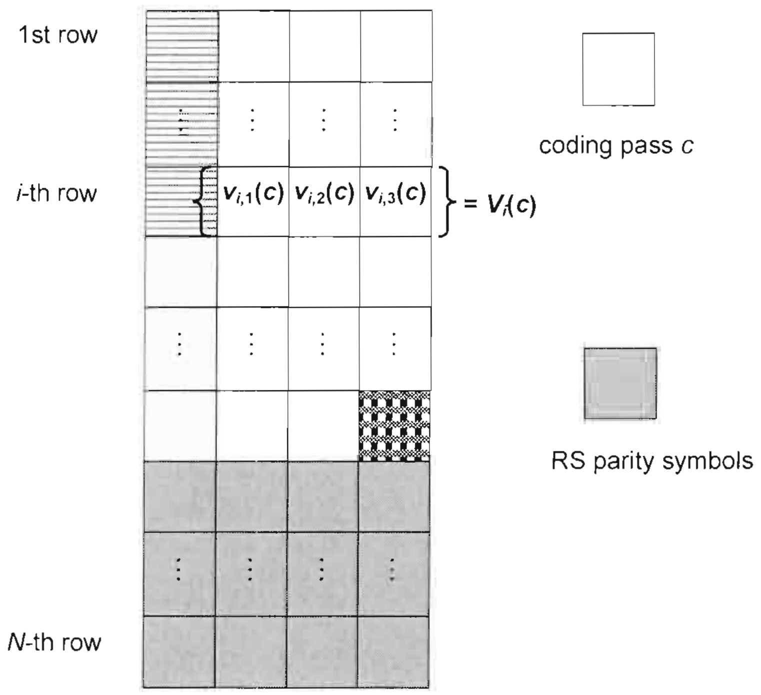

Figure 5-1. An illustration of a critical segment that shows how the elements of $V_i(c)$ are located.

Since auxiliary bits are not allocated for c's with $|V_i(c)| = 1$, the contribution of these c's to the total number of verifications would be $v_{i,1}(c)+1$ (i.e., the number of guesses that should be verified, starting from 0, to find the correct value). When $|V_i(c)| > 1$, the calculation of the estimate is slightly different. Because m-bit symbols are arranged inside the ULP structure, if we assign a auxiliary bits to a symbol, the guess for the m-a least significant bits of that symbol along with the a auxiliary bits (i.e., the a most significant bits of the original symbol), would constitute the guess value. Therefore, by combining the contribution of all elements of $V_i(c)$, there are

$$1+\sum_{1\le j\le |V_i(c)|} r_{i,j}(c)\times 2^{(m-a)(|V_i(c)|-j)} \tag{5.1}$$

guesses that should be verified, where $r_{i,j}(c)$ is the remainder of the division of $v_{i,j}(c)$ by 2^{m-a}. Consequently, the total number of verifications would be

$$g_i=\sum_{c:|V_i(c)|=1}(1+v_{i,1}(c))+\sum_{c:|V_i(c)|>1}(1+\sum_{1\le j\le |V_i(c)|} r_{i,j}(c)\times 2^{(m-a)(|V_i(c)|-j)}). \tag{5.2}$$

The expected number of required verifications is then equal to

$$E[g]=\sum_{1\le i\le K} q_i\times g_i\ , \tag{5.3}$$

where K is the number of source symbols in each column of the critical segment, and q_i is the probability of the first erasure occurring at the i-th packet.

5.2. RESULTS

In Figure 5-2, the estimated number of verifications, based on the model presented in the previous section, and the average number of verifications, obtained by 1000 simulations, for four different cases of images with 64×64 codeblocks are illustrated as a function of the number of auxiliary bits. Standard deviations of the simulation results are also depicted on the graphs, by vertical bars. For this specific series of simulations, we used neither the ability of the JPEG2000 decoder to signal the bytes in which errors are detected, nor the accelerating capacity described in section 4.1.3 that mismatching auxiliary bits provide. Therefore, those behaviours of the joint decoder that are difficult to predict are excluded, and the proposed complexity model can be analysed more fairly. In each graph, trends of the simulation and the estimation curves are similar, and the estimates lie within one standard deviation of their corresponding simulation averages; however, there are a few discrepancies that show the inaccuracy of the estimation, noticeably in the case of the *Boat* image. These discrepancies are, basically, the results of the simplifying assumption, which we adopted when formulating the expected number of verifications.

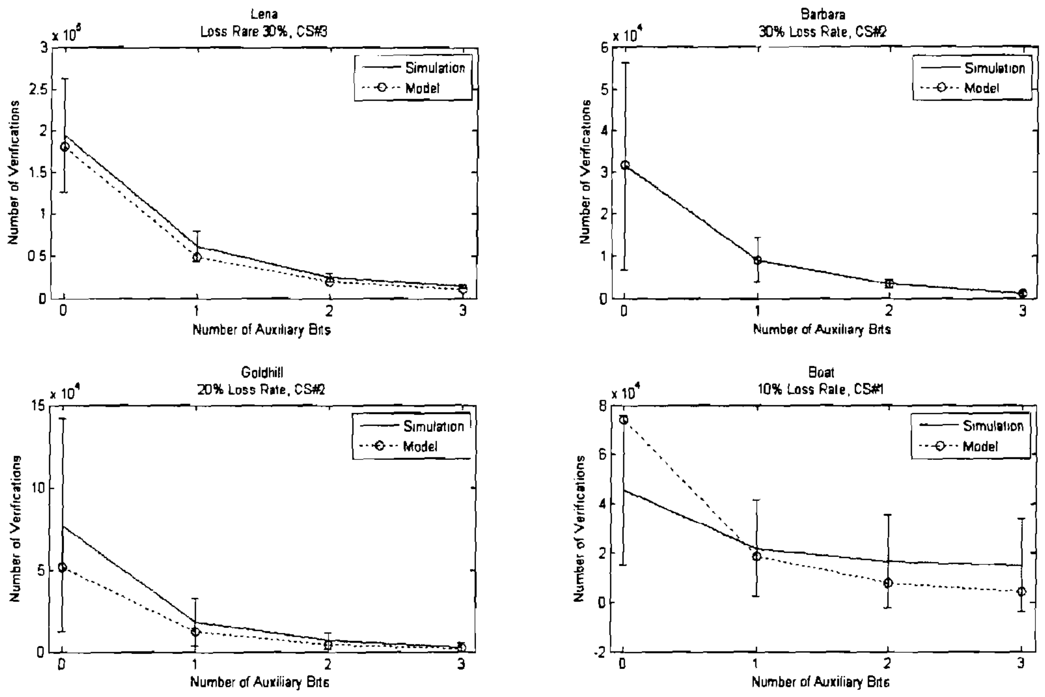

Figure 5-2. Number of verifications (derived by simulation and the complexity model) vs. Number of auxiliary bits, for four different cases.

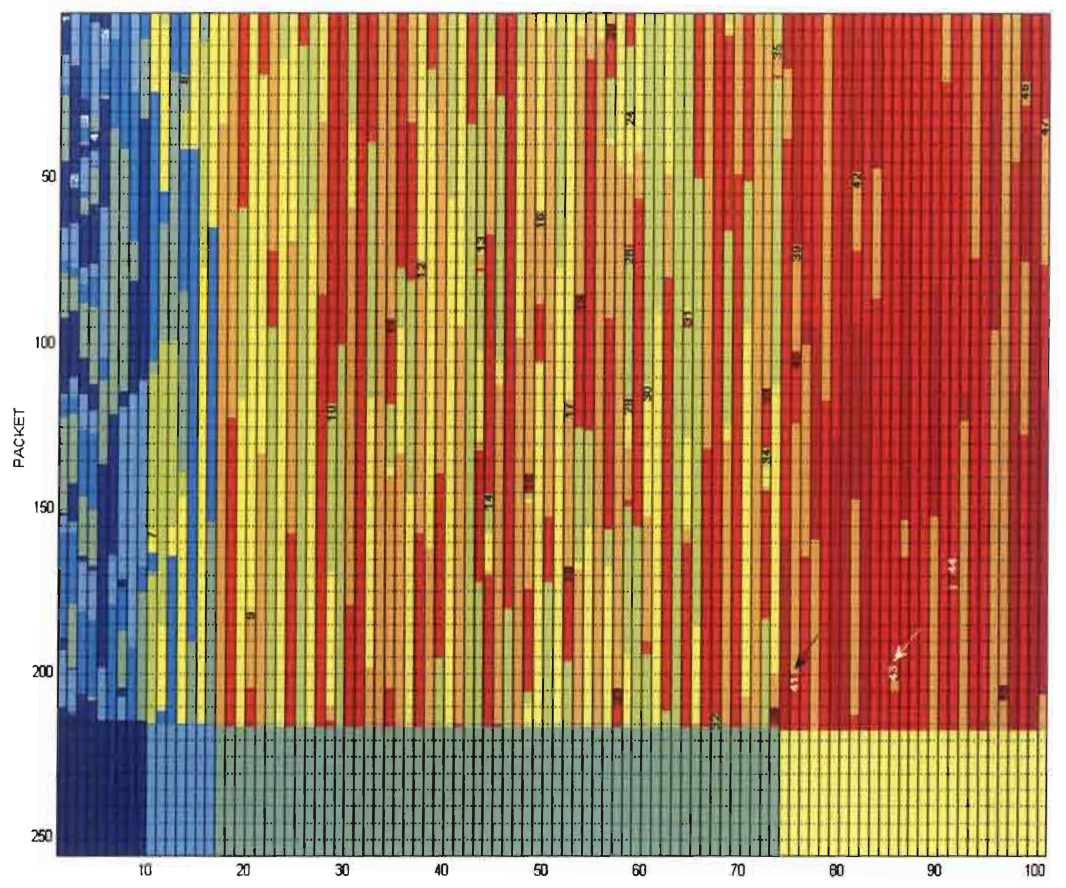

Figure 5-3. Arrangement of codeblocks and coding passes of the *Boat* image, inside the ULP matrix.

To explain the effects of our assumption on the estimate, we use Figure 5-3, which shows how coding passes and codeblocks of the *Boat* image are organized in the ULP structure, under the same conditions as shown in Figure 5-2; the numbers written inside the ULP matrix mark the beginning of the codeblock bitstreams by their indices. Alternating colors indicate the regions that belong to different coding passes. The last three critical segments can be identified by the color of their corresponding parity symbols.

The bottom-right graph in Figure 5-2 shows that when auxiliary bits are not used (i.e., $a = 0$), the calculated average has noticeably exceeded the simulation average. Some coding passes, which most of the time are located at the tail of their codeblock, have less error resilience than the other coding passes because there are not enough checkpoints left in the codeblock bitstream letting the JPEG2000 decoder detect possible errors [10]. The evident overestimation occurs when some of these coding passes contain more than one symbol of the first erased packet, because these coding passes might be decoded and

passed by much earlier than what we counted in the estimation formula. The last coding pass of the codeblock pointed to by a black arrow (i.e., the 41st codeblock) is an example of such coding passes. As mentioned before, the effect of these coding passes is diminished by increasing the number of auxiliary bits.

There is yet another type of inaccuracy which becomes dominant as the effect of the first inaccuracy fades by increasing the number of auxiliary bits. This time, due to late error detection, corruption of some coding passes might not be detected within the same column(s) of the ULP matrix in which they are located. Therefore, to restore their correct values during the JSCD, these coding passes should be decoded together with other coding passes in the following column(s). For example, the last two coding passes of the codeblock pointed to by a white arrow (i.e., the 43rd codeblock), potentially, need to be decoded together. In fact, compared to other cases, errors in the first coding pass are more likely to trigger a decoding complaint at the second coding pass. Since such complaints are indistinguishable from the complaints generated when only the second coding pass is corrupted, correctness of the first coding pass is highly uncertain unless the second coding pass is also decoded successfully. Nonetheless, coupled decoding of such distinct coding passes is not considered in the estimation formula, and depending on the original values of the erased symbols, we may underestimate the number of verifications significantly. In our example, if the combined coding passes have not received auxiliary bits, increasing the number of auxiliary bits would not reduce this type of inaccuracy either. Negligible change in the standard deviations also confirms this observation.

CHAPTER 6: CONCLUSIONS

In this thesis, we have presented a JSCD algorithm for ULP schemes, which assists in decoding beyond the limit of separate source and channel decoding of ULP packets. If the number of erasures happens to be more than the maximum number of erasures that the employed ULP matrix can completely recover, the proposed JSCD technique can retrieve the lost data partially. The results imply that, depending on the amount of information restored by the joint decoding process, the proposed JSCD can noticeably increase the PSNR. However, our simulations show that the basic JSCD can be undesirably time-consuming.

To accelerate the JSCD process, we examined a combination of techniques. First, we investigated the effect of using different size of codeblocks on the JSCD speed. The results show the JSCD performs much faster for smaller codeblocks; however, some quality loss in return can be expected. Since this quality reduction may be intolerable, other solutions are also considered to tackle the slow joint decoding problem.

The basic joint decoder is designed to verify all of the possible guess values for a coding pass consecutively, until it finds the correct value for the erased symbols. In the second acceleration technique that we proposed, part of the consecutive guessing iterations can be skipped, using the ability of the JPEG2000 decoder to indicate where in the bitstream of corrupted coding passes the decoding is interrupted. Given the location of decoding interruption, the joint decoder may sooner correct the wrong guess values it has left undetected, and thus avoid several unsuccessful guessing iterations. The simulation results show that this acceleration technique can speed up the process to some extent, but this amount of acceleration may still be inadequate for reducing the joint decoding time below a desirable level.

Furthermore, we considered another speed-up technique in which some extra information, the auxiliary bits, help the decoder to reduce the number of trials during the joint decoding. This approach is advantageous in two ways. First, when these auxiliary bits are available, several guess values can be eliminated from the search space in the first place. Second, by comparing the available auxiliary bits with their corresponding values generated after an RS decoding, the incorrect guessed values could be detected without calling the time-consuming JPEG2000 decoding procedure. The results show that even one auxiliary bit per symbol increases the speed of JSCD significantly. The number of transmitted auxiliary bits, however, must be relatively small; therefore, we proposed two policies to assign the auxiliary bits only to a selection of symbols, which potentially affect the basic JSCD execution time more than the other symbols. While this selective utilization of auxiliary bits keeps the additional bit budget relatively low, it provides a speed improvement comparable to that when the auxiliary bits are assigned to every symbol.

Finally, we provided a simplified model for evaluating the complexity of the algorithm. We exclude some the improved JSCD attributes from the model because they are difficult to interpret theoretically. The simulation results obtained using the reduced version of the improved joint decoder show that the complexity model often provides a passable estimate of the expected execution time; though it may present inaccurate estimates on less likely cases where JPEG2000's error detection mechanisms perform poorly.

APPENDIX A IMPLEMENTATION ISSUES

Implementation of the proposed JSCD algorithm using the Kakadu software [1] imposes some limitations that should be mentioned. First, the order by which the codeblock decoder procedure of Kakadu is called does not conform to the order of codeblocks information in the JPEG2000 bitsream. In fact, the codeblocks are decoded in a pre-determined order. Therefore, codeblocks of a resolution-scalable image and a quality-scalable image, for instance, are decoded in the same order. Although this implementation of the codeblock decoder might be beneficial for code optimization purposes, it can negatively affect the developed JSCD. Since we implemented most of the JSCD algorithm inside the codeblock decoder function of Kakadu, the joint decoding may suffer from inappropriate order of the codeblocks mentioned before. For instance, as Figure A-1 illustrates, Kakadu decodes several codeblocks in resolution #3 of the image after decoding some of the codeblocks in resolution #4. This problem results in the underestimation of the expected quality improvement that proposed JSCD can achieve, because with this implementation, some codeblocks that have less contribution in the total quality improvement might be decoded during JSCD, and instead, other more crucial codeblocks might be simply discarded. In terms of the joint decoding speed, however, improvements cannot be guaranteed, but we can expect roughly the same decoding times.

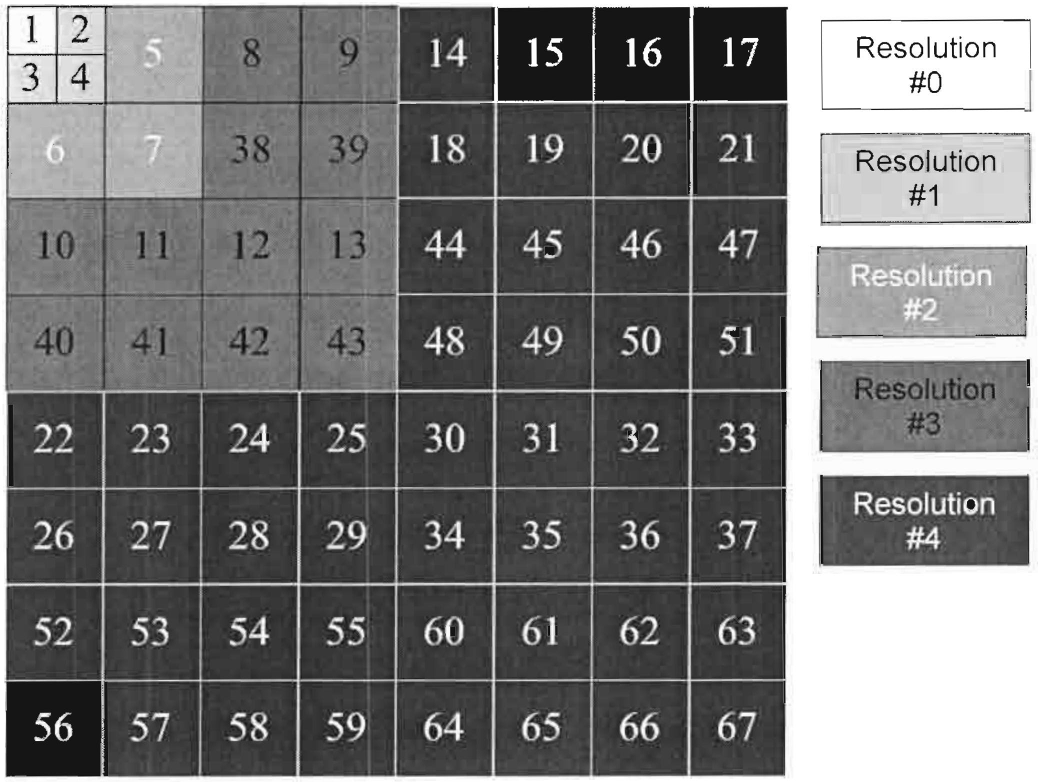

Figure A-1. A wavelet domain structure with five resolution levels. The codeblocks are labelled by their index in the sequence of decoding.

The second limitation in using Kakadu, which is closely related to the first one, is the lack of flexibility for accessing individual codeblocks information. When the decoding of a particular codeblock is finished, it cannot be accessed anymore, unless the structure of Kakadu is substantially modified. Therefore, if the joint decoder needs to correct a wrong guess left in the bitstream of the last decoded codeblock, it can only make the correction virtually to continue JSCD. As a result, the previous codeblock will remain corrupted even if the correct values of its corrupt symbols are found during joint decoding of the following codeblocks. This limitation also can cause some quality reduction, though the improved JSCD technique explained in Chapter 4 is very likely to prevent these events.

APPENDIX B USER GUIDE

The CD-ROM attached forms a part of this work.

The results file can be opened with MSExcel or other spreadsheet program. In addition to the executable files of the JSCD implementation, a MATLAB code (m-file) is also included to run the simulations automatically.

Data Files:

• Simulation Results	
results.xls	133 KB
• ULP Patterns (directory)	3.53 KB

C Source Codes:

• mdfec (directory)	3.00 MB

Executable Files:

• Separate Source and Channel Decoder	
▪ sscd.exe	284 KB
• Basic Joint Source-Channel Decoders	
▪ jscd_a0.exe	316 KB
▪ jscd_a1.exe	320 KB
▪ jscd_a2.exe	320 KB
▪ jscd_a3.exe	320 KB
• Improved Joint Source-Chanel Decoders	
▪ enh_dec_a0.exe	320 KB
▪ enh_dec_a1.exe	320 KB
▪ enh_dec_a2.exe	320 KB
▪ enh_dec_a3.exe	320 KB

MATLAB File:

• Automatic Simulator	
▪ autosim.m	5.78 KB

The following steps show how to reproduce the simulation results:

- COMPRESSING THE RAW IMAGE INTO ERROR RESILIENT JPEG2000 FILES:

  The following command line exemplifies how we generated the JPEG2000 files for the simulations using Kakadu encoder:

  ```
  kdu_compress -i lena.pgm -o lena.j2c -rate .778
  Cuse_sop=yes Cuse_eph=yes "Cmodes={ERTERM|RESTART}"
  Cprecincts={128,128} Cblk={64,64}
  ```

  This command produces a JPEG2000 file ('lena.j2c') at 0.778 bpp source rate from the raw image 'lena.pgm'. Cuse_sop and Cuse_eph arguments indicate the use of the start of packet (SOP) and the end of packet header (EPH) markers, which are designed for detecting errors in the header information of the JPEG2000 bitstream; however, these markers are optional and do not have any positive effect on our proposed JSCD. The ERTERM and RESTART modes of the encoder are also switched on using the Cmode argument to provide the required error resilience of the bitstream. In this particular example, the precinct size and the maximum codeblock size are set to 128×128 and 64×64 using the Cprecincts and Cblk arguments, respectively; the first and the second numerical elements in each of these fields correspond to height and width of the division (i.e., precinct or codeblock) in that order.

- GENERATING OPERATIONAL DISTORTION-RATE FUNCTION:

  Before the optimization step, we need to produce a file containing the operational PSNR-Rate or Distortion-Rate function of the compressed image. Data should be stored in a two-column text file, in which the first column contains the number of source bits decoded, with 8-bit steps, and the second column shows the corresponding quality measure. To produce the necessary information, we truncate the JPEG2000 bitstream at 100 equidistant rates, and decode the truncated files to find the corresponding PSNR's. Then the remaining points on the PSNR-Rate curve are generated by linear interpolation of the available points.

- OPTIMIZATION OF THE NUMBER OF PARITY SYMBOLS:

  The original Microsoft® Visual C++ project for the optimization algorithm introduced in [15] is included under the folder *\mdfec\mdf_stankovic* in the CD-ROM. The text file produced in the previous step is used as the input of this program to find a sub-optimal solution to the parity symbol allocation problem of ULP. Note that some initial information should be determined manually in the C code of the algorithm. This information include the name of the PSNR-Rate text file, the number of samples (i.e., the length of the JPEG2000 file in bits), number and length of the packets, and the presumed average channel loss rate.

- JOINT SOURCE-CHANNEL DECODING:

The outcome of the previous step is a text file ('levels.txt') that contains the (close-to-) optimal number of parity symbols in each column of the ULP matrix, from the leftmost column to the rightmost. This file is used as an input for the JSCD. There are eight executable files used for simulation of the JSCD with different number of auxiliary bits. The files named *jscd_a*.exe* are the implementations of the basic JSCD. These joint decoders perform the JSCD explained in Chapter 3; however, they can also use auxiliary bits to simply reduce the number of possible values for the first erased symbol. Furthermore, *enh_dec_a*.exe* files are the accelerated joint decoders with the improvements proposed in Chapter 4. In both of these sets of files the number in place of the * indicates the number of auxiliary bits used for the JSCD. We also included another executable file called 'sscd.exe', which is the implementation of the separate source and channel decoding, in the CD-ROM.

It is necessary to have a realization of the channel erasures before beginning the JSCD of the compressed image. The channel erasure pattern should be written in a text file named 'e_pattern.txt'. This file is only a sorted list from the index of the erased packets. Since we limited the joint decoding to the cases where the value of one symbol only should be guessed, the number of erasures reflected in the erasure pattern file should be one more than one of the numbers stored in the 'levels.txt' file. Otherwise, an error message might be received following the attempt for joint decoding.

Note that the 'levels.txt' and 'e_pattern.txt' files must be in the same directory that the joint decoder and the compressed image are. The following command line is an example that shows how to use the joint decoder:

```
jscd_a0.exe -i lena.j2c -o test.pgm -resilient
```

The resilient argument is necessary to force the decoder to use the ER features of the bitstream for error detection.

To facilitate the simulations we used a MATLAB code ('autosim.m') to run the joint decoders, automatically. This code runs all of the JSCD executables as well as the separate source and channel decoder for the last three critical segments of an image encoded with 16×16, 32×32, and 64×64 maximum codeblock size. Note that the name of the target image would be determined by the variable *img_name* in the M-file, and all of the three versions of that image should be placed in the same location as the executable files are. Moreover, the filename of the test JPEG2000 images should contain the maximum width/height of their codeblocks as a postfix. For example, the test JPEG2000 images created from the raw *Lena* image would be 'lena16.j2c', 'lena32.j2c', and 'lena64.j2c' which have 16×16, 32×32, and 64×64 maximum codeblock size, respectively. At the end of each simulation, the MATLAB code writes the resulting average execution times and the average PSNR's in an excel file ('results.xls') under a worksheet labelled by the image name. The mean and standard deviation of the number of times that the JPEG2000 decoder detects errors during the JSCD will also be written in the same worksheet under the headings μ and σ, respectively. Each worksheet contains the results of the basic JSCD and the improved JSCD.

REFERENCE LIST

[1] D. S. Taubman and M. W. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards and Practice*, Kluwer, Norwell, MA, 2001.

[2] H.-C. Huang, W.-H. Peng, T. Chiang, and H.-M. Hang, "Advances in the scalable amendment of H.264/AVC," *IEEE Commun. Mag.*, vol. 45, no. 1, pp. 68-76, Jan. 2007.

[3] V. Chande and N. Farvardin, "Progressive transmission of images over memoryless noisy channels," *IEEE J. Selec. Areas Commun.*, vol. 18, no. 6, pp. 850-860, Jun. 2000.

[4] P. Sherwood and K. Zeger, "Error protection for progressive image transmission over memoryless and fading channels," *IEEE Trans. Commun.*, vol. 46, no. 12, pp. 1555-1559, Dec. 1998.

[5] J. Kim, R. A. Mersereau, Y. Altunbasak, "Bit-plane-wise unequal error protection for internet video applications," *Proc. IEEE ICC'02,* vol. 4, pp. 2508-2512, New York City, NY, Apr. 2002.

[6] C. Lan, K. R. Narayanan, Z. Xiong, "Scalable image and video transmission using irregular repeat-accumulate codes with fast algorithm for optimal unequal error protection," *IEEE Trans. Commun.*, vol. 6, pp. 1092-1101, Jul. 2004.

[7] T. Stockhammer, C. Weiss, "Channel and complexity scalable image transmission," *Proc. IEEE ICIP'01*, vol. 1, pp. 102-105, Thessaloniki, Greece, Oct. 2001.

[8] B. Banister, B. Belzer, and T. Fischer, "Robust image transmission using JPEG2000 and turbo-codes," *IEEE Signal Processing Lett.*, vol. 9, no. 4, pp. 117-119, Apr. 2002.

[9] K. P. Subbalakshmi, "Joint source-channel decoding of variable-length encoded sources with applications to image transmission," PhD Dissertation, School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada, Jul. 2000.

[10] L. Pu, Z. Wu, A. Bilgin, M. W. Marcellin, and B. Vasic, "LDPC-based iterative joint source-channel decoding for JPEG2000," *IEEE Trans. Image Processing*, vol. 16, no. 2, pp. 577-581, Feb. 2007.

[11] L. R. Bahil, J. Cocke, F. Jelinek, and J. Raviv "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. 20, no. 2, pp 284-287, Mar. 1974.

[12] M. Grangetto, B. Scanavino, G. Olmo, S. Benedetto, "Iterative decoding of serially concatenated arithmetic and channel codes with JPEG2000 applications," *IEEE Trans. Image Processing*, vol. 16, no. 6, pp. 1557-1567, Jun. 2007.

[13] R. Puri and K. Ramachandran, "Multiple description source coding using forward error correction codes," *Proc. Asilomar*, vol. 1, pp. 342-346, Oct. 1999.

[14] A. E. Mohr, E. A. Riskin, and R. E. Ladner, "Unequal loss protection: Graceful degradation of image quality over packet erasure channels through forward error correction," *IEEE J. Select. Areas Commun.*, vol. 18, no. 6, pp. 819-828, Jun. 2000.

[15] V. M. Stanković, R. Hamzaoui, and Z. Xiong, "Real-time error protection of embedded codes for packet erasure and fading channels," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 14, no. 8, pp. 1064-1072, Aug. 2004.

[16] S. Bahmani, I. V. Bajić, and A. HajShirmohammadi, "Joint source-channel decoding of JPEG2000 images with unequal loss protection," *Proc. IEEE ICASSP'08*, pp. 1365-1368, Las Vegas, NV, Mar. 2008.

[17] S. Bahmani, I. V. Bajić, and A. HajShirmohammadi, "Joint source-channel decoding of JPEG2000 images with unequal loss protection," submitted for publication in *IEEE Trans. Image Processing*, Aug. 2008.

[18] S. Bahmani, I. V. Bajić, and A. HajShirmohammadi, "Improved joint source-channel decoding of JPEG2000 images and Reed-Solomon codes," submitted for presentation at *IEEE ICC'09*, Sep. 2008.

[19] ISO/IEC 15444-1. JPEG2000 image coding system, 2000.

[20] K. Sayood, H. H. Otu, and N. Demir, "Joint source/channel coding for variable length codes," *IEEE Trans. Commun.*, vol. 48, no. 5, pp. 787-794, May 2000.

[21] C. Boyd, J. G. Cleary, S. A. Irvine, I. Rinsma-Melchert, and I. H. Witten, "Integrating error detection into arithmetic coding," *IEEE Trans. Commun.*, vol. 45, no. 1, pp 1-3, Jan. 1997.

[22] Z. Wu, A. Biglin, and M. W. Marcellin, "Error resilient decoding of JPEG2000," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 12, pp. 1752-1757, Dec. 2007.

[23] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, New York, NY, 1977.

[24] I. S. Reed, G. Solomon, “Polynomial codes over certain finite fields”, *J. Soc. Indust. and Appl. Math.*, vol. 8, no. 2, pp. 300-304, Jun. 1960.

[25] P. Karn, C++ class library for Galois field arithmetic and algebra, with RS encoder/decoder, [Online] Available: http://www.ka9q.net/code/fec/fec-3.0.1.tar.bz2