Soft decoding and synchronization of arithmetic codes: Application to image transmission over noisy channels

Thomas Guionnet and Christine Guillemot

Abstract— **This paper addresses the issue of robust and joint source-channel decoding of arithmetic codes. We first analyze dependencies between the variables involved in arithmetic coding by means of the Bayesian formalism. This provides a suitable framework for designing a soft decoding algorithm that provides high error-resilience. It also provides a natural setting for "soft synchronization", i.e., to introduce anchors favoring the likelihood of "synchronized" paths. In order to maintain the complexity of the estimation within a realistic range, a simple, yet efficient, pruning method is described. The algorithm can be placed in an iterative source-channel decoding structure, in the spirit of serial turbo codes. Models and algorithms are then applied to context-based arithmetic coding widely used in practical systems (e.g. JPEG-2000). Experimentation results with both theoretical sources and with real images coded with JPEG-2000 reveal very good error resilience performances.**

I. Introduction

Entropy coding, producing variable length codewords (VLC), is a core component of any data compression scheme. However VLCs are very sensitive to channel noise: when some bits are altered by the channel, synchronization losses can occur at the receiver, the position of symbol boundaries are not properly estimated, leading to dramatic symbol error rates. This phenomenon has given momentum to extensive work on the design of codes with better synchronization properties [15], [28], [32], [29]. as well as to the design of procedures of soft decoding [26], [18], [20] and of joint source-channel decoding [19], [5], [2], [9] of VLCs. First research efforts have been focused on Huffman codes [20], [19], [5], [1], and on reversible VLCs [32], [29], [2]). Recently, arithmetic codes have gained increased popularity in practical systems, including JPEG-2000, H.264 and MPEG-4 standards. When used in conjunction with high-order source statistical models, maximum compression factors can be obtained. However, the counterpart is an increased sensitivity to noise.

Methods considered to fight against noise sensitivity consist usually in re-augmenting the redundancy of the bitstream, either by introducing an error correcting code or by inserting dedicated patterns in the chain. The author in [7] reintroduces redundancy in the form of parity check bits. A probability interval not assigned to a symbol of the source alphabet or markers inserted at known positions in the sequence of symbols are exploited for error detection in [3], [8], [25]. This capability can then be coupled with an ARQ procedure [4], [8] or used jointly with an error correcting code. A serial structure concatenating an inner error correction code and an outer error detection code is described in [14]. Sequential decoding of arithmetic codes is investigated in [22] for supporting error correction capabilities.

Here, we first analyze the dependencies and constraints between the variables involved in arithmetic coding and decoding in light of Bayesian networks. As in [9], our starting point is a state space model of the source and of the arithmetic coder. We first consider for the source model an order-1 Markov model. Given this source model, the arithmetic coder codes the *innovation* of the markov chain. The arithmetic coder, driven by the conditional probability of the order-1 source model, gets a sequence of symbols and outputs a sequence of bits, hence operates a clock conversion with a varying rate. The observed output of a memoryless channel corresponds to noisy measurements of these bits. Recovering the transmitted sequence of source symbols from these noisy measurements is equivalent to inferring the sequence of model states. Notice that, like in the case of any VLC, the segmentation of the sequence of measurements into the sequence of model states is random. In addition, here, some transitions may not produce any bits : a measurement may inform about a varying number of transitions. Another difficulty comes from the fact that the codetree grows exponentially with the number of symbols being coded.

The stochastic modeling of coder and decoder sets the basis for the design of error-resilient soft decoding algorithms. The complexity of the underlying Bayesian estimation algorithm growing exponentially with the number of coded symbols, a simple, yet efficient, pruning method is described. It allows to limit the complexity within a tractable and realistic range, at a limited cost in terms of estimation accuracy. The second key aspect is that the models provide a natural setting for introducing "soft synchronization " patterns. These patterns, inserted in the symbol stream at some known positions, serve as anchors for favoring the likelihood of correctly synchronized paths in the trellis. This "soft synchronization idea" augments the autosynchronization power of the chain at a controllable loss in information rate. The algorithm is also amenable for iterative source-channel decoding in the spirit of serially concatenated codes. The estimation, pruning and soft synchronization algorithms are first validated with theoretical Gauss-Markov sources with different correlation factors and for a wide range of channel signal to noise ratio (E_b/N_0). Models and algorithms are in a second step adapted to context-based arithmetic coding widely used in practical systems (e.g. JPEG-2000, H.264) and validated in a JPEG-2000 decoder.

The rest of the paper is organized as follows. Section II describes part of the notation we use and states the problem addressed. After a review of the main principles of arithmetic coding in section III, section IV addresses modeling issues : This material is exploited in the sequel (section V) for explaining the estimation algorithms together with the pruning and "soft synchronization" principles. Section VI outlines the construction of the iterative joint source-channel decoding procedure based on arithmetic codes. Adaptations of the algorithm to context-based arithmetic coding are described in section VII. Experimental results are discussed in section VIII.

II. Notations

Let $S = S_1 \dots S_K$ be the sequence of quantized source symbols taking their values in a finite alphabet $\mathcal{A}$ composed of $M = 2^q$ symbols, $\mathcal{A} = \{a_1, a_2, \dots, a_i, \dots, a_M\}$, and coded into a sequence of information bits $U = U_1 \dots U_N$, by means of an arithmetic coder. k and n represent generic time indexes for the symbol

All authors are with IRISA-INRIA, Campus de Beaulieu, 35042 Rennes Cedex, FRANCE.

E-mail:firstname.lastname@irisa.fr

clock and the bit clock, respectively. The length N of the information bit stream is a random variable, function of S. Lengths K and N, of the symbol stream and of the bit stream, are supposed to be known. The sequence $S = S_1 \dots S_K$ is assumed to form an order-1 Markov chain. The bitstream U is sent over a memoryless channel and received as measurements Y ; so the problem we address consists in estimating S given the observed values y. Notice that we reserve capital letters to random variables, and small letters to values of these variables. For handling ranges of variables, we use the notation $X_u^v = \{X_u, X_{u+1}, \dots, X_v\}$ or $\bar{X}_I$ where I is the index set $\{u, u+1, \dots, v\}$.

III. Arithmetic coding: principle

Let us first review the principle of arithmetic coding on a simple example of a source taking values in the alphabet $\{a_1, a_2, a_3, a_4\}$ with the stationary distribution $\mathbb{P}_s$ = [0.6 0.2 0.1 0.1]. Fig. 1 illustrates the arithmetic coding principle. The interval $[0, 1[$ is partitioned into four cells representing the four symbols of the alphabet. The size of each cell is the stationary probability of the corresponding symbol. The partition (hence the bounds of the different segments) of the unit interval is given by the cumulative stationary probability of the alphabet symbols. The interval corresponding to the first symbol to be coded is chosen. It becomes the current interval and is again partitioned into different segments. The bounds of the resulting segments are driven by the model of the source. Considering an order-1 Markov chain, these bounds will be governed by $\mathbb{P}(S_{l+1}|S_l)$, hence in this particular case, will be function of both the probability of the previous symbol and of the cumulative probability of the alphabet symbols. Therefore, the arithmetic coder adapts in this case to the conditional entropy *rate* $H(S_{l+1}|S_l)$ of process S, i.e. it compresses the innovation of the Markov chain S.

In the example above, when the sequence $a1, a4, a2, a1$ has been coded, the current interval is $[0.576, 0.5832[$. Any number in this interval can be used to identify the sequence. Let us consider 0.576. The decoding of the sequence is performed by reproducing the coder behavior. First the interval $[0, 1[$ is partitioned according to the cumulative probability of the source. Since the value 0.576 belongs to the interval $[0, 0.6[$, it is clear that the first symbol coded has been $a1$. Therefore the first symbol is decoded and the interval $[0, 0.6[$ is partitioned according to the cumulative probability of the source. The process is repeated until full decoding of the sequence. Notice that the coder and the decoder are not synchronized. The conversion they both perform between the bit clock and the symbol clock differs.

Practical implementations of arithmetic coding have been first introduced in [23] and [21], and developed further in [24]. One problem that may arise when implementing arithmetic coding is the high precision needed to represent very small real numbers. In order to overcome this difficulty, one can base the algorithm on the binary representation of real numbers in the interval $[0, 1[$ [30]. Any number in the interval $[0, 0.5[$ will have its first bit equal to 0, while any number in the interval $[0.5, 1[$ will have its first bit equal to 1. Therefore, during the encoding process, as soon as the current interval is entirely under or over $\frac{1}{2}$, the corresponding bit is emitted, and the interval length is doubled. There is a specific treatment for the intervals straddling $\frac{1}{2}$. When the current interval straddles $\frac{1}{2}$ and is in $[0.25, 0.75[$, it cannot be identified by a unique bit. Its size is therefore doubled, without emitting any bit, and the number of re-scalings taking place before emitting any bit is memorized. When reaching an interval for which one bit $U_i = u_i$ can be emitted, then this bit will be followed by a number of bits $U_{i+1} = u_{i+1} \dots U_{i+n} = u_{i+1}$, where n is the number of scaling operations that have been performed before the emission of U_i, and where $u_{i+1} = u_i + 1 \bmod 2$. The use of this technique guarantees that the current interval always satisfies $low < 0.25 < 0.5 \leq high$ or $low < 0.5 < 0.75 \leq high$, where low and $high$ are respectively the lower and upper bounds of the current interval. This avoids the problems of precision which may otherwise occur in the case of small intervals straddling the middle of the segment $[0, 1[$. Further work on arithmetic coding, essentially to reduce the implementation complexity, can be found in [16], [10], [11], [12] and [31].

A last key element in the encoding process concerns the last step or termination, so that the decoder can identify without ambiguity the last symbol S_K. It is pointed out in [30] that, in order to do so, it is necessary to send a set of bits that will ensure that the coded string identifies uniquely the interval corresponding to the last symbol S_K. Using the scaling strategy described above, the interval corresponding to the last symbol either straddles the interval $]0.25, 0.5]$ or the interval $]0.5, 0.75]$. The message is then terminated respectively by a 0 followed by a sequence of $1's$, or by a 1 followed by a sequence of $0's$. The number of bits N of the message to be coded is then given by $N = \lceil -log_2(high - low) \rceil$.

IV. Models of arithmetic coder and decoder

This section tries to analyze dependencies between the variables involved in the coding and decoding processes. Let us assume that each symbol S_k takes its value in an alphabet composed of M symbols. Models of dependencies can be obtained by considering either the M-ary coding tree or the binary decoding tree.

A. Symbol clock model of the coder

Given the M-symbol alphabet $\mathcal{A}$, the sequence of symbols S_1^K is translated into a sequence of bits U_1^N by an M-ary decision tree. This tree can be regarded as an automaton that models the bitstream distribution. The encoding of a symbol determines the choice of a vertex in the tree. Each node of the tree identifies a state X of the arithmetic coder, to which can be associated the emission of a sequence of bits of variable length. Successive branching on the tree follow the distribution of the source ($\mathbb{P}(S_k|S_{k-1})$ for an order one Markov source).The arithmetic coder realizes a clock conversion which depends on the source realization. Hence, the number of bits being produced by each transition on the above model being random, the structure of dependencies between the sequence of measurements and the states of the coding tree is random. In order to capture this randomness, we consider the augmented Markov chain $(X, N) = (X_1, N_1) \dots (X_K, N_K)$, where $k = 1 \dots K$ denotes the symbol clock instants. The quantity N_k is the number of bits emitted when k symbols have been coded, and X_k is the internal state of the arithmetic coder. Thus, successive branching on the tree correspond to transitions between (X_{k-1}, N_{k-1}) and (X_k, N_k). The coding tree grows exponentially with the number of symbols being coded and the leaves correspond to the possible combinations of sequences being coded. The value of X_k is implementation dependent. It must at least contain a segment of the interval $[0, 1[$ specified for example by its lower and upper bounds l_k and h_k.

The arithmetic encoding process starts from the initial state (X_0, N_0) where X_0 is the unit interval and $N_0 = 0$. In the example of section III, if the first symbol coded is a_1, then the vertex that leads to the state (X_1, N_1) is chosen, with $X_1 = [0, 0.6[$

and $N_1 = 0$. If the second symbol coded is a_4, the next state is (X_2, N_2) with $X_2 = [0.54, 0.6[$. This new interval allows to determine that the three first bits are 100, therefore $N_2 = 3$. In this example, we have considered that the state variable X_k is reduced to values of the bounds of the current interval $[l_k, h_k[$. However, in order to avoid numerical precision limitation, the interval scaling technique described in [30] can be considered. The state variable X_k is then composed of the three variables l_k, h_k and $nbScaling_k$, where $nbScaling_k$ stands for the number of scaling operations applied to a given interval without emitting any bit. The initial state is given by $l_0 = 0$, $u_0 = 1$ and $nbScaling_0 = 0$. The operations taking place for each transition from (X_k, N_k) to (X_{k+1}, N_{k+1}), triggered by the symbol $S_{k+1} = a_i$ are given in Tab. I. The quantities $P_{cum}(a_i) = \sum_{t=1}^{i} P(a_t)$ and $P_{cum}(a_i|a_j) = \sum_{t=1}^{i} P(a_t|a_j)$ denote respectively the stationary and conditional cumulative probabilities of the symbols of the alphabet $\mathcal{A}$. To a state (X_k, N_k) is associated the emission of a sequence of bits $\bar{U}_k = U_{N_{k-1}+1}^{N_k}$, of length $N_k - N_{k-1}$, where $N_k \geq N_{k-1}$, leading to the graphical representation of Fig. 2. Gathering measurements $\bar{Y}_k$ of the bits $\bar{U}_k$ at the output of the transmission channel, we have a symbol clock model of the arithmetic coder. Notice that the set of possible values taken by a state (X_k, N_k) varies according to k.

B. Bit clock model of the decoder

A sequence of arithmetically coded bits U_1^n can be translated into a sequence of symbols $S_1^{K_n}$ by a binary decision tree. Each bit determines the choice of a vertex in the tree. Each node ν of the tree corresponds to a tuple U_1^{n-1} from which two transitions are possible $U_n = 0$ or $U_n = 1$. Each node of the tree identifies a state of the arithmetic decoder. These states are represented by a pair (X_n, K_n), where X_n is the internal state of the arithmetic decoder, K_n is the maximum number of symbols that can be decoded at this point and $n = 1 \ldots N$ denotes the bit clock instants. Thus, unlike the arithmetic coder, the decoder is driven by the bit clock. Transitions between (X_{n-1}, K_{n-1}) and (X_n, K_n), are governed by $\mathbb{P}(S_{K_{n-1}+1} \ldots S_{K_n} | S_1^{K_{n-1}})$. This leads to the model depicted in Fig. 3, where at each state (X_n, K_n) of the arithmetic decoder is associated a sequence of decoded symbols $S_{K_{n-1}+1}^{K_n}$. The transition between two states (X_{n-1}, K_{n-1}) and (X_n, K_n) is triggered by the bit U_n from which the measurement Y_n is available.

The set of actions triggered by each transition is given in Tab. II. For an arithmetic decoder, two intervals need to be kept : the first one, $[\boldsymbol{l_n}, \boldsymbol{h_n}[$, is obtained from the n currently decoded bits. The second one, $[\boldsymbol{l_{K_n}}, \boldsymbol{h_{K_n}}[$, corresponds to the K_n currently decoded symbols. These two intervals are initialized to $[0, 1[$. It is not necessary to keep track of the scalings which are performed to avoid precisions problems.

V. Estimation algorithms

The model of dependencies depicted in section IV can be exploited to help the estimation of the bitstream (hence of the symbol sequence). The MAP (maximum *a posteriori*) criterion corresponds to the optimal Bayesian estimation of a process X based on measurements Y :

$$\hat{X} = \arg\max_x \mathbb{P}(X = x|Y) \tag{1}$$

The optimization is performed over all possible *sequences* x. This applies directly to the estimation of the hidden states of the processes (X, N) (symbol clock) and (X, K) (bit clock), given the sequence of measurements.

A. Estimation on the symbol clock model

Estimating the set of hidden states $(X, N) = (X_1, N_1) \ldots (X_K, N_K)$ on the symbol clock model (Fig. 2) is equivalent to estimating the sequence of transitions between these states, i.e. to estimating the sequence $S = S_1 \ldots S_k \ldots S_K$, given the measurements Y_1^N at the output of the channel. This can then be expressed as

$$\begin{aligned} \mathbb{P}(S_1^K|Y_1^N) &= \mathbb{P}(S_1^K) \cdot \mathbb{P}(U_1^N|Y_1^N) \\ &\propto \mathbb{P}(S_1^K) \cdot \mathbb{P}(Y_1^N|U_1^N), \end{aligned} \tag{2}$$

where $\propto$ denotes a renormalization factor, which here is $\frac{\mathbb{P}(U_1^N)}{\mathbb{P}(Y_1^N)} = 1$, since the bits emitted by an arithmetic coder are uniformly independently distributed. The quantity $\mathbb{P}(S_1^K|Y_1^N)$ can be computed by a forward sweep recursion expressed as

$$\begin{aligned} \mathbb{P}(S_1^k|Y_1^{N_k}) \propto\ & \mathbb{P}(S_1^{k-1}|Y_1^{N_{k-1}}) \cdot \mathbb{P}(S_k|S_{k-1}) \cdot \\ & \mathbb{P}(Y_{N_{k-1}+1}^{N_k}|Y_1^{N_{k-1}}, U_1^{N_k}), \end{aligned} \tag{3}$$

where N_k is the number of bits that have been transmitted when arriving at the state X_k. Assuming S to be an order-1 Markov chain and considering a memoryless channel, this *a posteriori* probability can be rewritten as

$$\begin{aligned} \mathbb{P}(S_1^k|Y_1^{N_k}) \propto\ & \mathbb{P}(S_1^{k-1}|Y_1^{N_{k-1}}) \cdot \mathbb{P}(S_k|S_{k-1}) \cdot \\ & \mathbb{P}(Y_{N_{k-1}+1}^{N_k}|U_{N_{k-1}+1}^{N_k}) \end{aligned} \tag{4}$$

The process is initialized with $\mathbb{P}(S_1 = s_1|Y_1^{N_1=n_1} = y_1^{N_1=n_1})$, for each of the M possible values of s_1, given by

$$\begin{aligned} \mathbb{P}(S_1 = s_1|Y_1^{N_1=n_1}) \propto\ & \mathbb{P}(S_1 = s_1) \cdot \\ & \mathbb{P}(Y_1^{N_1} = y_1^{n_1}|U_1^{N_1} = u_1^{n_1}), \end{aligned} \tag{5}$$

where the first term on the right hand side is given by the stationary distribution of the source and where the second term models the noise introduced by the channel. This second term, under the assumption that the channel is memoryless, can be computed as

$$\begin{aligned} &\mathbb{P}(Y_{N_{k-1}+1}^{N_k} = y_{n_{k-1}+1}^{n_k}|U_{N_{k-1}+1}^{N_k} = u_{n_{k-1}+1}^{n_k}) = \\ &\begin{cases} \prod_{n=N_{k-1}+1}^{N_k} \mathbb{P}(Y_n = y_n|U_n = u_n) \textbf{ if } N_k > N_{k-1} \\ 1 \textbf{ otherwise.} \end{cases} \end{aligned} \tag{6}$$

Then the update equation (Eqn. 4) is used to compute $\mathbb{P}(S_1^k|Y_1^{N_k})$ for each level consecutively. At the end of the estimation process, the posterior marginals $\mathbb{P}(S_1^K = s_1^K|Y_1^{N_K})$ are known for all possible sequences realizations s_1^K. As we consider all the possible symbol sequences, the trellis of the estimation algorithm is an *M-ary* tree of depth K. Therefore the complexity of the algorithm is $O(M^K)$, which is untractable. Section V-C describes a pruning technique that allows to maintain the complexity of the algorithm within a realistic level.

B. Estimation on the bit clock model

The estimation can be run similarly on the bit clock model depicted in Fig. 3. In this case, the forward sweep recursion computes $\mathbb{P}(S_1^{K_n}|Y_1^n)$, where K_n is a random variable denoting

the number of symbols decoded at the reception of bit n. The term $\mathbb{P}(S_1^{K_n}|Y_1^n)$ can be expressed as

$$\mathbb{P}(S_1^{K_n}|Y_1^n) \quad \propto \quad \mathbb{P}(S_1^{K_n}) \cdot \mathbb{P}(Y_1^n|U_1^n) \tag{7}$$

Once again, under the assumption that S is an order-1 Markov process, we have

$$\mathbb{P}(S_1^{K_n}) = \mathbb{P}(S_1^{K_{n-1}}) \cdot \mathbb{P}(S_{K_{n-1}+1}^{K_n}|S_1^{K_{n-1}}), \tag{8}$$

where

$$\mathbb{P}(S_{K_{n-1}+1}^{K_n}|S_1^{K_{n-1}}) = \begin{cases} \prod_{k=K_{n-1}+1}^{K_n} \mathbb{P}(S_k|S_{k-1}) \textbf{ if } K_n > K_{n-1} \\ 1 \textbf{ otherwise.} \end{cases} \tag{9}$$

For a memoryless channel, Eqn. 7 can be rewritten as

$$\begin{aligned} \mathbb{P}(S_1^{K_n}|Y_1^n) \quad &\propto \quad \mathbb{P}(S_1^{K_{n-1}}) \cdot \mathbb{P}(Y_1^{n-1}|U_1^{n-1}) \cdot \\ &\quad \mathbb{P}(Y_n|U_n) \cdot \prod_{k=K_{n-1}+1}^{K_n} \mathbb{P}(S_k|S_{k-1}) \\ &\propto \quad \mathbb{P}(S_1^{K_{n-1}}|Y_1^{n-1}) \cdot \mathbb{P}(Y_n|U_n) \cdot \\ &\quad \prod_{k=K_{n-1}+1}^{K_n} \mathbb{P}(S_k|S_{k-1}) \end{aligned} \tag{10}$$

Notice that when running the estimation on the symbol clock model, the value of N does not have to be known. In contrast, with the bit clock model, N has to be known and K can be estimated.

C. Pruning

The number of states increasing exponentially with k, one has to consider pruning to limit the complexity of the estimation. Let us consider first the estimation running on the symbol clock model. Assuming that N is known, one can ensure that the last bit N of the chain terminates a symbol, by adding an extra measurement (or constraint) on the last state (X_K, N_K), as $N_K = N$. At the last step of the estimation, only the solutions for which the value of N_K is equal to N are valid. In addition, the paths in the trellis that lead to values of N_k larger than N can be ignored. One can consider the above constraint as a first pruning criterion.

The main pruning criterion is based on a threshold applied on the likelihood of the path. The likelihood of a path, hence of a node of the M-ary tree, is given by Eqn. 2. The nodes corresponding to a sequence of bits which is unlikely (i.e., with a probability below a certain threshold T) may be pruned. A constraint on the number of nodes of the trellis at each step k of the estimation process can also be added : the W nodes with the highest probability are kept. With this strategy, the complexity of the algorithm is bounded by $O(KW)$. The choice of W then depends on the affordable complexity. This constraint has been refined by keeping the w best nodes for each N_k. In this case, we have $W = (N_{kmax} - N_{kmin} + 1)w$, where N_{kmax} and N_{kmin} depend on the probabilities of the most and less probable sequences.

Pruning is performed similarly when running the estimation on the bit clock model. Assuming that K is known, the constraint $K_n = K$ can be added on the last state (X_n, K_n). Considering the bit clock model, the likelihood of a path, hence of a node of the binary tree, is given by Eqn. 7. Thus, a threshold T on the probability of the nodes can be defined, and the number of nodes at each step n of the estimation can be limited to the w best ones for each K_n.

D. Soft synchronization

Termination constraints mentioned in section V-C can be regarded as means to force synchronization at the end of the sequence : they indeed either constraint the decoder to have the right number of symbols ($K_N = K$) (if known) after decoding the estimated bit stream $\hat{U}$, or alternately ensure that the K symbols produced by the estimation do match the N bits sequence. They add some kind of "perfect observation" on the last state of the models. These constraints ensure synchronization at the end of the bit stream, but do not ensure synchronization in the middle of the sequence. One can introduce extra information specifically to help the resynchronization "in the middle" of the sequence. For this, we can introduce extra bits at some known positions $I_s = \{i_1, \ldots, i_s\}$ in the symbol stream.

This extra information can take the form of dummy symbols (in the spirit of the techniques described in [3], [6], [22], [7] and [25]), or of dummy bit patterns which are inserted respectively in the symbol or bit stream, at some known symbol clock positions. Bit patterns can have arbitrary length and frequency, depending on the degree of redundancy desired. Notice that, unlike long words used in standards to identify the beginning and the end of the sequence of information bits, these extra symbols are not "perfect" synchronization points : the probability to recognize them is far from 1, whence the name of "soft synchronization points". Since they are placed at some known positions in the symbol stream (at symbol clock instants), their position in the bitstream is random.

Models and algorithms above have to account for this extra information. Inserting an extra dummy symbol at known positions in the symbol stream amounts to add a section on the M-ary tree with deterministic transitions. The presence of this extra information can be exploited by the estimation, which can be either run on the symbol clock or bit clock models. When the estimation is run on the symbol clock model, the a-priori information on the deterministic transitions in the M-ary tree are directly accounted for in Eqn. 4, by the term $\mathbb{P}(S_k|S_{k-1})$. When considering the bit clock model, it is similarly exploited in Eqn.8.

Let us now consider the insertion of dummy bit patterns. Inserting extra bits at known symbol clock positions amounts to terminating some symbols by appending a "suffix" $\tilde{U}_{K_n} \Rightarrow \tilde{U}_{K_n} B_1 \ldots B_{l_s}$. In other words, for $K_n \in I_s$, one must use an extended binary tree, where some nodes are followed by the suffix. Transitions are deterministic in this extra part of the binary tree. Observe that the memory of the symbol produced is not lost by X. As a consequence, the symbol is considered as finished when the suffix is finished. In other words, the K part of the state is augmented only when the end of the "suffix" is reached. The extra information is exploited respectively in Eqn. 6 (symbol clock model) and in Eqn. 10 (bit clock model) via the term $P(Y_n = y_n|U_n = u_n)$. Since desynchronized paths do not display the expected value of the bit pattern for time k, they "pay" via an assumed transmission error (whence a lower likelihood). These extra bit patterns will hence allow to "pinch" the distribution $P(k, n|Y)$ around the right value of n for $k \in I_s$. Since "de-synchronized" paths are very likely to be pruned, these markers also help to prevent excessive growth of the trellis.

The insertion of bit patterns, rather than an arithmetically coded dummy symbol, allows a more flexible control of the redundancy added. Also, notice that, when we insert an arithmetically coded symbol, the memory of the sequence encoded so far is lost. In the sequel, we consider only soft synchronization

based on bit patterns. We saw above that the position of these patterns in the bitstream depends on the variable N_k, hence is random. One may also consider periodic transmission of side information such as e.g. the value of the bit counter N_k for given positions k. This information must be protected. This strategy ensures good synchronization and very efficient pruning.

VI. Iterative CC-AC decoding algorithm

The soft synchronization mechanism described above significantly increases the reliability of the segmentation and estimation of the sequence of symbols. One can however consider in addition the usage of an error correction code, e.g. of a systematic convolutional channel code (CC). Both codes can be concatenated in the spirit of serial turbo codes. Adopting this principle, one can therefore work on each model (arithmetic coder and channel coder) separately and design an iterative estimator, provided an interleaver is introduced between the models [9]. The structure of dependencies between the variables involved is outlined on Fig. 4. Such a scheme requires extrinsic information on the bits U_n, $Ext_{U_n}(Y|Y_n) \propto \frac{\mathbb{P}(U_n|Y)}{\mathbb{P}(U_n|Y_n)}$ to be transmitted by the CC to the soft arithmetic decoder, and reciprocally. This information represents the modification induced by the introduction of the rest of the observations Y_1^{n-1}, Y_{n+1}^N on the conditional law of U_n given Y_n. The iterative estimation proceeds by first running a BCJR algorithm on the channel coder model. The extrinsic information on the useful bits U_n, direct subproduct of the BCJR algorithm, can in turn be used as input for the estimation run on the arithmetic decoder model. The result of the AC soft decoding procedure is a-posteriori probabilities $\mathbb{P}(X_K, N_K|Y)$ on the last state (X_K, N_K) of the symbol clock model (respectively (X_N, K_N) for the bit clock model) corresponding to entire sequences of symbols. These a-posteriori probabilities must then be converted into information on bits U_n by the equation

$$\begin{aligned} P(U_n = i|Y)|_{i=0,1} &= \sum_{(x_K, N_K):U_n=i} \mathbb{P}(x_K, N_K|Y), \\ \text{or} &= \sum_{(x_N, K_N):U_n=i} \mathbb{P}(x_N, K_N|Y), \end{aligned} \tag{11}$$

depending on the model (symbol clock or bit clock) used for the estimation.

VII. Context-based arithmetic coding

Substancial improvements in compression can be obtained by using more sophisticated source models. These models can take the form of conditional probabilities based on contexts which can consist of one or more preceding symbols. Most emerging real coding systems (e.g. JPEG-2000, H.264, MPEG-4 part 10) rely on context-based arithmetic coding. Therefore, the source cannot be considered as an order-1 Markov source as above. The models and algorithms however still apply. This is shown in the following in the particular case of JPEG-2000.

A. Context-based arithmetic coding in EBCOT

A key element of JPEG-2000 is the EBCOT (Embedded Block Coding with Optimized Truncation) algorithm which operates on blocks within the wavelet domain. A block can be seen as an array of integers in sign-magnitude representation and is coded bitplane per bitplane with context-dependent arithmetic codes. Each bitplane is coded in three passes, referred to as sub-bitplanes. The first coding pass is called the significance propagation pass. A bit is coded during this pass if its location is not significant, but at least one of its neighbors is significant. A location is significant if a 1 has already been coded in the current or previous bitplanes. If the bit's location is not yet significant, a combination of zero coding (ZC) and run length coding (RLC) primitives is used. When a bit's location becomes significant (ZC=1), it is immediately followed by the sign coding (SC) primitive, to encode the sign of this new significant location. The purpose of this pass is to visit the bits that are likely to become significant. The second coding pass is the magnitude refinement pass. In this pass, all bits from locations that are already significant are coded with the magnitude refinement (MR) primitive. The third and final pass is the cleanup pass, coding all the remaining bits.

In summary, during each coding pass the decision of processing a bit or not is based on the context of this bit. The context of a bit gathers the significance of that bit's location and the significance and the signs of neighboring locations. When a bit has to be processed, it can be coded via four binary primitives which are then arithmetically coded with the MQ-coder described in [13]. The statistical model used by this arithmetic coder is contextual. A full description of the EBCOT algorithm can be found in [27], [17] and [13].

B. Application of models and algorithms

So far, in the derivation of the models and algorithms, we have considered an order-1 Markov source. Both coder and decoder, in order to respectively approach at best the entropy bound and to have the best estimation accuracy, exploit the stationary $P(s_i)$ and transition probabilities $P(s_i|s_{i-1})$. In the case of context based arithmetic coding, the model considered relies on probability distributions of a binary source given its context. For the JPEG-2000 case, let us note S_k, the k^{th} bit visited in the block and C_k the context associated to this bit. Due to the three passes scheme, the scan order indexed by k is context dependent. The probabilities $P(S_k|C_k)$ are fully specified in the standard.

The estimation on the symbol clock model relies on Eqn. 3, where the source model is exploited by expressing $P(S_1^k)$ as a function of $P(S_1^{k-1})$ and of the transition probability from S_1^{k-1} to S_k. In the case of the contextual model, we have :

$$P(S_1^k) = P(S_1^{k-1}) \cdot P(S_k|C_k), \tag{12}$$

and the estimation scheme holds. Similarly, when considering the bit clock model, the context dependency is introduced in Eqn. 9 via the factorization

$$\mathbb{P}(S_{K_{n-1}+1}^{K_n}|S_1^{K_{n-1}}) = \begin{cases} \prod_{k=K_{n-1}+1}^{K_n} \mathbb{P}(S_k|C_k) \textbf{ if } K_n > K_{n-1} \\ 1 \textbf{ otherwise.} \end{cases} \tag{13}$$

For both clock models, the states (X_k, N_k) or (X_n, K_n) that are estimated do not only contain the state of the arithmetic coder, but also the *context* information related to the three passes coding procedure. Note that the number of symbols to be coded is not known a-priori. The estimation should then be performed on the bit clock model.

The estimation algorithm has been integrated in a JPEG-2000 codec [13]. The block partitioning limits spatial error propagation. We have also added a "segmentation symbol" at the end of each bitplane. In the JPEG-2000 standard, this segmentation symbol is used for error detection during decoding. It is a symbol which is arithmetically coded and therefore expected at the

decoding. The estimation process can also use this segmentation symbol as a resynchronization marker (section V-D). We have added the capability to vary the length of this marker and its frequency of appearance between every bitplane (standard), every coding pass, or every stripe.

VIII. Experimentation results

To evaluate the performance of the soft arithmetic decoding procedure, a first set of experiments have been performed on order-1 Gauss-Markov sources, with zero-mean, unit-variance and different correlation factors. The source is quantized with an 8 levels uniform quantizer (3 bits) on the interval $[-3, 3]$. All the simulations have been performed assuming an additive white Gaussian channel with a BPSK modulation. The results are averaged over 100 realizations. The scanning strategy of the different branches on the trellis follows the breadth first strategy [22]. Performances obtained using respectively symbol clock and bit clock models are not identical, however they are very close for a given complexity. Results are presented for the symbol clock model only. In practical applications, the choice between these two approaches may depend on the availability of the values of K and/or N parameters.

The first experiments aimed at comparing the performances in terms of symbol error rates (SER) and signal to noise ratio (SNR) of soft versus hard decoding of arithmetic codes. Arithmetic soft decoding is also evaluated against soft decoding of Huffman codes [9], for comparable bit rates. When the source correlation increases, the compression efficiency of the arithmetic coder increasing, soft synchronization patterns can be added in the arithmetically coded stream up to having comparable rates for both codes.

Fig. 5, 6, 7 plot the residual SER and SNR curves as a function of E_b/N_0, for respectively $\rho = 0.1$, $\rho = 0.5$ and $\rho = 0.9$. The first observation is that soft arithmetic decoding significantly outperforms hard decoding. For sources with low correlation ($\rho = 0.1$), soft arithmetic and Huffman decoding give comparable performances with a slight advantage for Huffman codes. This can be explained easily by the fact that, when the source correlation is low, both codes lead to comparable compression efficiency. Hence, no additional synchronization patterns are added in the arithmetically coded stream, whereas some redundancy remains in the Huffman coded stream. Huffman codes have then a better resynchronization capability. However, the relative performance of arithmetic versus Huffman codes increases with the source correlation. The arithmetic coder performing much better in terms of compression than the Huffman coder, soft synchronization patterns can be added up to a comparable rate.

The second experiment aimed at analyzing the effect of soft synchronization patterns for both codes. Fig. 8 and 9 depict the SER and SNR performances respectively for $\rho = 0.5$ and 0.9. Again, high source correlation is clearly at the advantage of arithmetic codes. For lower source correlation ($\rho = 0.5$), soft arithmetic decoding outperforms soft Huffman decoding for $E_b/N_0 \geq 2$ dB. The compression factor obtained with arithmetic codes being higher, there is more flexibility to add redundancy in an appropriate way, i.e. to specifically handle the resynchronization problem which turns out to be the most crucial for VLC codes. Fig. 8 and 9 also compare soft synchronization with periodic transmission of side information. Soft synchronization outperforms the mechanism based on side information. The length of soft patterns being smaller, their periodicity can be higher for the same amount of redundancy, leading to more frequent decoder re-synchronization. However, the use of side information is much more effective in terms of pruning. When using side information, the estimation is between two and ten times faster.

Fig. 10(a) shows the performance of iterative CC-AC decoding against soft AC decoding with soft synchronization for $\rho = 0.1$. Although the performance of the iterative scheme is rather good, the iterations bring no significant gain. Fig. 10(b) depicts the performance of the iterative CC-AC decoding scheme as a function of the number of iterations for different pruning parameters. One can observe that the pruning plays a significant role in limiting the benefits of the iterations.

The algorithms have been incorporated in a JPEG-2000 coder and decoder. Error resilient markers specified by the standard have been systematically activated. Moreover, information headers and syntax markers have been protected by a systematic convolutional channel code of rate 1/3, defined by the polynomials $F(z) = 1 + z + z^2 + z^4$ and $G(z) = 1 + z^3 + z^4$. When soft arithmetic decoding is performed, soft synchronization patterns are inserted at the end of each stripe. The complexity parameter W is set to 20. The decoding complexity, in terms of both memory usage and CPU time is controlled by the parameter W. As far as memory usage is concerned, $2W$ code-blocks need to be stored in memory when decoding a single code block. Experiments have been carried out with $W = 10$ and $W = 20$. The CPU times for $W = 20$ and $W = 10$ are within a factor of two. The gain obtained for $W = 20$ is illustrated in Fig. 11. The length N of the bitstream depends on the block sizes and on the bitrate. The number of symbols K is not known a priori, hence the termination constraint described in section V-C is not exploited. Fig. 11 provides the PSNR as a function of E_b/N_0 obtained with the image Lena 512x512 compressed at 0.25 bits per pixels. Instantaneous decoding with standard JPEG-2000 parameters provides poor error resilience. Significantly increased performances are achieved with the estimation algorithm described above. Fig. 12 shows the visual quality obtained with soft versus hard decoding. The test image is Lena 512x512 compressed at 0.5 bits per pixels. The channel is supposed to be AWGN with $E_b/N_0 = 5db$. When using hard decoding, due to error propagation, all the information can be lost (Fig. 12(c)).

IX. Conclusion

Bayesian networks are a natural tool to analyze the structure both of dependencies and of constraints between the variables involved in the coding and decoding processes. Estimation algorithms follow quite naturally. In the case of arithmetic codes, the estimation complexity grows exponentially with the number of symbols in the sequence. Pruning strategies are then required to limit the complexity. The algorithms, developed first for order-1 sources, adapts also quite naturally to higher order models such as those considered in context-based arithmetic coding. The use of higher-order models leads to an increased noise sensitivity often resulting in decoder de-synchronization. However, data-dependent bit patterns can be added in the symbol stream, at some known positions, in order to favor the likelihood of synchronized paths. Alternately one can also send periodically coder state-related information. The trade-off between compression and redundancy can then be easily adapted to varying network conditions. The redundancy is not inherent to the structure of the code. The extra information also helps in preventing excessive growth of the trellis. The usage of channel codes can also be considered in order to reduce the bit error rate seen by the estimation algorithm. The latter can then be placed in an iterative decoding structure in the spirit of serially concatenated turbo codes. However, the gain brought by the iterations depends on

the amount of pruning performed. Current work is dedicated to the design of a simplified state model that would avoid pruning hence better exploit the potential of an iterative decoding structure.

References

[1] R. Bauer and J. Hagenauer. Iterative source/channel decoding based on a trellis represent ation for variable length codes. In *Proc. IEEE Intl. Symposium on Information Theory*, page 117, Piscataway, NJ, USA, June 2000.

[2] R. Bauer and J. Hagenauer. Turbo fec/vlc decoding and its application to text compression. In *Proc. Conf. on Information Science and System*, pages WA6.6–WA6.11, Princeton, NJ, USA, Mar. 2000.

[3] C. Boyd, J. G. Cleary, S. A. Irvine, I. Rinsma-Melchert, and I. H. Witten. Integrating error detection into arithmetic coding. *IEEE Trans. on Communications*, 45(1):1–3, Jan. 1997.

[4] J. Chou and K. Ramchandran. Arithmetic coding-based continuous error detection for efficient arq-based image transmission. *IEEE Journal on Selected Areas in Communication*, 18(6):861–867, June 2000.

[5] N. Demir and K. Sayood. Joint source-channel coding for variable length codes. In *Proc. IEEE Data Compression Conf.*, pages 139–148, Snowbird, UT, USA, Mar. 1998.

[6] C. Demiroglu, M. W. Hoffman, and K. Sayood. Joint source channel coding using arithmetic codes and trellis coded modulaion. In *Proc. Data Compression Conf.*, pages 302–311, Snowbird, UT, USA, Mar. 2001.

[7] G. F. Elmasry. Embedding channel coding in arithmetic coding. *IEE Proc.-Communications*, 146(2):73–78, Apr. 1999.

[8] G. F. Elmasry. Joint lossless-source and channel coding using automatic repeat request. *IEEE Trans. on Communications*, 47(7):953–955, July 1999.

[9] A. Guyader, E. Fabre, C. Guillemot, and M. Robert. Joint source-channel turbo decoding of entropy coded sources. *IEEE Journal on Selected Areas in Communication, special issue on the turbo principle : from theory to practice*, 19(9):1680–1696, Sept. 2001.

[10] P. G. Howard and J. S. Vitter. Analysis of arithmetic coding for data compression. *Information Processing and Management*, 28(6):749–763, Nov.-Dec. 1992.

[11] P. G. Howard and J. S. Vitter. *Image and Text Compression*, pages 85–112. Kluwer Academic Publisher, 1992.

[12] P. G. Howard and J. S. Vitter. Arithmetic coding for data compression. *Proc. IEEE*, 82(6):857–865, 1994.

[13] ISO/IEC 15444-1:2000. *JPEG 2000 image coding system – Part 1*, first edition, Dec. 2000.

[14] I. Kozintsev, J. Chou, and K. Ramchandran. Image transmission using arithmetic coding based continuous error detection. In *Proc. Data Compression Conf.*, pages 339–348, Snowbird, UT, USA, Mar. 1998.

[15] W.M. Lam and A.R. Reibman. Self-synchronizing variable-length codes for image transmission. In *Proc. IEEE Intl. Conf. on Accoustic Speech and Signal Processing*, volume 3, pages 477–480, New York, NY, USA, Sept. 1992.

[16] G. G. Langdon. An introduction to arithmetic coding. *IBM J. Res. Develop.*, 28(2):135–149, Mar. 1984.

[17] M.W. Marcellin, M.J. Gormish, A. Bilgin, and M.P. Boliek. An overview of jpeg-2000. In *Proc. Data Compression Conf.*, pages 523–541, Snowbird, UT, USA, Mar. 2000.

[18] D. J. Miller and M. Park. A sequence-based approximate MMSE decoder for source coding over noisy channels using discrete hidden markov models. *IEEE Trans. on Communications*, 46(2):222–231, Feb. 1998.

[19] A.H. Murad and T.E. Fuja. Joint source-channel decoding of variable length encoded sources. In *Proc. Information Theory Workshop*, pages 94–95, New York, NY, USA, June 1998.

[20] M. Park and D.J. Miller. Decoding entropy-coded symbols over noisy channels by MAP sequence estimation for asynchronous HMMs. In *Proc. Conf. on Information Sciences and Systems*, pages 477–482, Princeton, NJ, USA, May 1998.

[21] R. Pasco. *Source coding algorithms for fast data compression*. PhD thesis, Dept. of Electrical Engineering, Stanford Univ., Stanford Calif., 1976.

[22] B. D. Pettijohn, M. W. Hoffman, and K. Sayood. Joint source/channel coding using arithmetic codes. *IEEE Trans. on Communications*, 49(5):826–836, May 2001.

[23] J. J. Rissanen. Generalized kraft inequality and arithmetic coding. *IBM J. Res. Develop.*, 20:198–203, May 1976.

[24] J. J. Rissanen. Arithmetic codings as number representations. *Acta Polytech. Scand. Math.*, 31:44–51, Dec. 1979.

[25] I. Sodagar, B. B. Chai, and J. Wus. A new error resilience technique for image compression using arithmetic coding. In *IEEE Intl. Conf. on Accoustics, Speech and Signal Processing*, pages 2127–2130, Istanbul, Turkey, June 2000.

[26] K. P. Subbalakshmi and J. Vaisey. On the joint source-channel decoding of variable-length encoded sources: The bsc case. *IEEE Trans. on Communications*, 49(12):2052–2055, Dec. 2001.

[27] D. Taubman. High performance scalable image compression with ebcot. *IEEE Trans. on Image Processing*, 9(7):1158–1170, July 2000.

[28] J.H. Rabinowitz T.J. Ferguson. Self-synchronizing huffman codes. *IEEE Trans. on Information Theory*, IT-30(4):687–693, July 1984.

[29] J. Wen and J. D. Villasenor. Reversible variable length codes for efficient and robust image and video coding. In *Proc. IEEE Data Compression Conf.*, pages 471–480, Snowbird, UT, USA, Apr. 1998.

[30] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.

[31] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding revisited. *ACM Trans. on Information Systems*, 16(3):256–294, July 1998.

[32] M. Wada Y. Takishima and H. Murakami. Reversible variable length codes. *IEEE Trans. on Communications*, 43(4):158–162, Apr. 1995.

Thomas Guionnet received the B.S. degree from the university of Newcastle Upon Tyne, U.K. in computer science in 1997. He obtained the engineer degree in computer science and image processing and the PhD degree from the University of Rennes 1, France, respectively in 1999 and 2003. He is currently research engineer at INRIA and is involved in the french national project RNRT VIP and in the JPEG 2000 Part 11 - JPWL ad-hoc group. His research interests include image processing, coding and joint source and channel coding.

Christine Guillemot is currently 'Directeur de Recherche' at INRIA, in charge of a research group dealing with image modelling, processing and video communication. She holds a PhD degree from ENST (cole Nationale Suprieure des Tlcommunications) Paris. Her research interests are signal and image processing, coding and joint source and channel coding. From 1985 to October 1997, she has been with FRANCE TELECOM/ CNET, at CCETT, where she has been involved in various projects in the domain of coding for TV, HDTV and multimedia applications. From January 1990 to mid 1991, she has worked at Bellcore, NJ, USA. She serves as associated editor for IEEE Trans. on Image Processing and is a member of the IEEE IMDSP committee.

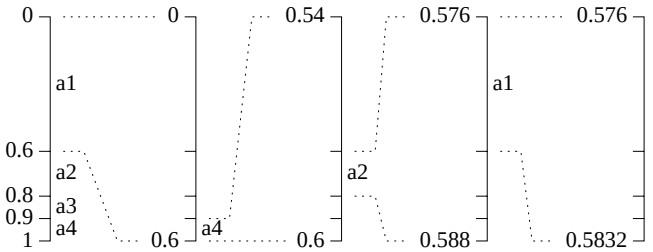

Fig. 1. Arithmetic coding principle.

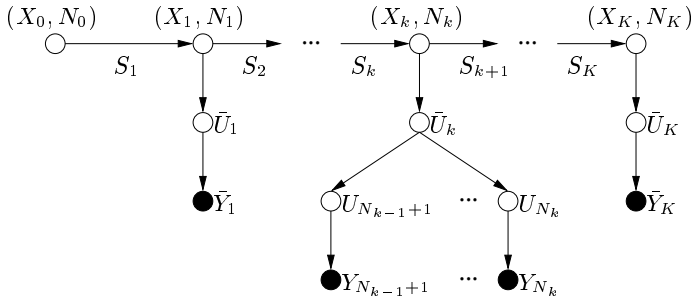

Fig. 2. Symbol clock model of the arithmetic coder ($N_k \geq N_{k-1}$). White and black dots represent respectively the hidden and observed variables.

1. Init (X_{k+1}, N_{k+1}) : Let $S_k = a_j$ and $S_{k+1} = a_i$.
- Set $\boldsymbol{l_{k+1}} = \boldsymbol{l_k} + (\boldsymbol{h_k} - \boldsymbol{l_k})\mathbb{P}_{cum}(a_{i-1}|S_k = a_j)$. If $k = 0$, $\mathbb{P}_{cum}(a_{i-1})$ is used.
- Set $\boldsymbol{h_{k+1}} = \boldsymbol{l_k} + (\boldsymbol{h_k} - \boldsymbol{l_k})\mathbb{P}_{cum}(a_i|S_k = a_j)$. If $k = 0$, $\mathbb{P}_{cum}(a_i)$ is used.
- Set $\boldsymbol{nbScaling_{k+1}} = \boldsymbol{nbScaling_k}$.
- Set $N_{k+1} = N_k$.

2. While one of the three scalings is possible, update (X_{k+1}, N_{k+1}) :
- If scaling 1 ($\boldsymbol{h_{k+1}} < 0.5$) :
  - Set $\boldsymbol{l_{k+1}} = 2\boldsymbol{l_{k+1}}$.
  - Set $\boldsymbol{h_{k+1}} = 2\boldsymbol{h_{k+1}}$.
  - Emit a bit 0.
  - Emit $\boldsymbol{nbScaling_{k+1}}$ bits 1.
  - Set $N_{k+1} = N_{k+1} + 1 + \boldsymbol{nbScaling_{k+1}}$.
  - Set $\boldsymbol{nbScaling_{k+1}} = 0$.
- If scaling 2 ($0.5 \leq \boldsymbol{l_{k+1}}$) :
  - Set $\boldsymbol{l_{k+1}} = 2(\boldsymbol{l_{k+1}} - 0.5)$.
  - Set $\boldsymbol{h_{k+1}} = 2(\boldsymbol{h_{k+1}} - 0.5)$.
  - Emit a bit 1.
  - Emit $\boldsymbol{nbScaling_{k+1}}$ bits 0.
  - Set $N_{k+1} = N_{k+1} + 1 + \boldsymbol{nbScaling_{k+1}}$.
  - Set $\boldsymbol{nbScaling_{k+1}} = 0$.
- If scaling 3 ($0.25 \leq \boldsymbol{l_{k+1}} < 0.5 \leq \boldsymbol{h_{k+1}} < 0.75$) :
  - Set $\boldsymbol{l_{k+1}} = 2(\boldsymbol{l_{k+1}} - 0.25)$.
  - Set $\boldsymbol{h_{k+1}} = 2(\boldsymbol{h_{k+1}} - 0.25)$.
  - Set $\boldsymbol{nbScaling_{k+1}} = \boldsymbol{nbScaling_{k+1}} + 1$.

3. End : (X_{k+1}, N_{k+1}) is up-to-date, $N_{k+1} - N_k$ bits have been emitted.

TABLE I

OPERATIONS TRIGGERED BY EACH TRANSITION OF THE ARITHMETIC CODER.

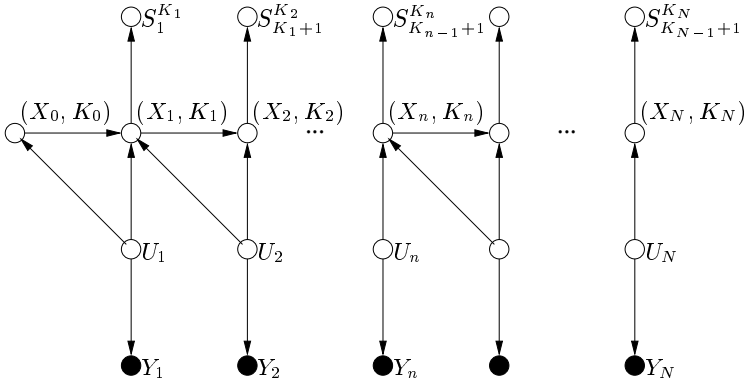

Fig. 3. Bit clock model of the arithmetic decoder. White and black dots represent respectively the hidden and observed variables.

1. Init (X_{n+1}, K_{n+1}) : Let $S_{K_n} = a_j$.
- Set $K_{n+1} = K_n$.
- Set $\boldsymbol{l_{K_{n+1}}} = \boldsymbol{l_{K_n}}$ and $\boldsymbol{h_{K_{n+1}}} = \boldsymbol{h_{K_n}}$.
- If $u_{n+1} = 0$
  Set $\boldsymbol{l_{n+1}} = \boldsymbol{l_n}$ and $\boldsymbol{h_{n+1}} = \frac{1}{2}(\boldsymbol{l_n} + \boldsymbol{h_n})$.

Else
  Set $\boldsymbol{l_{n+1}} = \frac{1}{2}(\boldsymbol{l_n} + \boldsymbol{h_n})$ and $\boldsymbol{h_{n+1}} = \boldsymbol{h_n}$.

2. Search $a_i \in \mathcal{A}$ such that
- $L = \boldsymbol{l_{K_{n+1}}} + (\boldsymbol{h_{K_{n+1}}} - \boldsymbol{l_{K_{n+1}}})P_{cum}(a_{i-1}|S_{K_{n+1}})$ (if $K_{n+1} = 0$, $P_{cum}(a_{i-1})$ is used),
- $H = \boldsymbol{l_{K_{n+1}}} + (\boldsymbol{h_{K_{n+1}}} - \boldsymbol{l_{K_{n+1}}})P_{cum}(a_i|S_{K_{n+1}})$ (if $K_{n+1} = 0$, $P_{cum}(a_i)$ is used),
- $L \leq l_{n+1} < h_{n+1} < H$ (condition for the symbol a_i to be decoded),

3. If a_i exists go to 4, else go to 5.
4. Update (X_{n+1}, K_{n+1}) :
- $K_{n+1} = K_{n+1} + 1$,
- $S_{K_{n+1}} = a_i$,
- $\boldsymbol{l_{K_{n+1}}} = L$,
- $\boldsymbol{h_{K_{n+1}}} = H$,
- Scale the intervals $[\boldsymbol{l_{n+1}}, \boldsymbol{h_{n+1}}]$ and $[\boldsymbol{l_{K_{n+1}}}, \boldsymbol{h_{K_{n+1}}}]$ if necessary,
- Go to 2.

5. End : (X_{n+1}, K_{n+1}) is up-to-date, $K_{n+1} - K_n$ symbols have been decoded.

TABLE II

OPERATIONS TRIGGERED BY EACH TRANSITION OF THE ARITHMETIC DECODER.

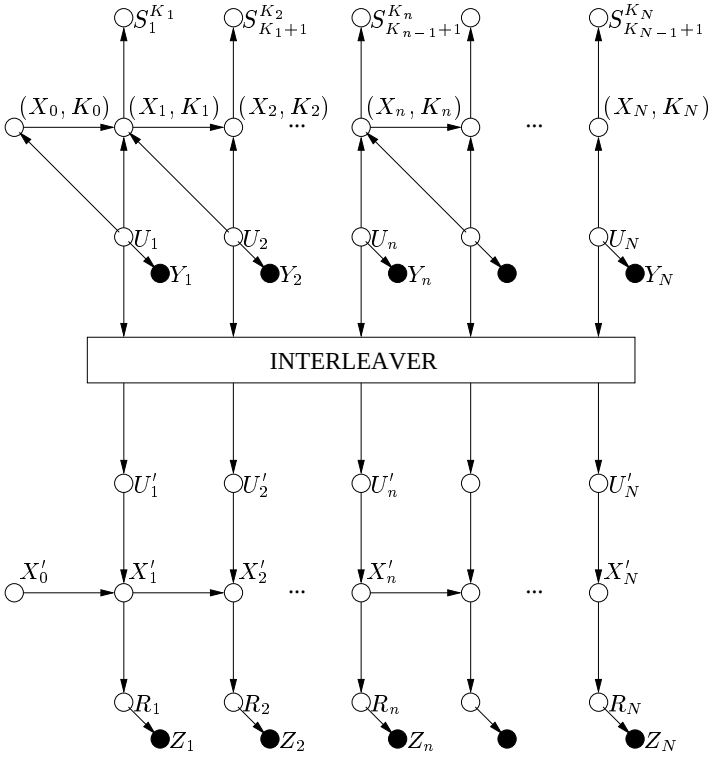

Fig. 4. Graphical representation of dependencies in the joint arithmetic-channel coding chain.

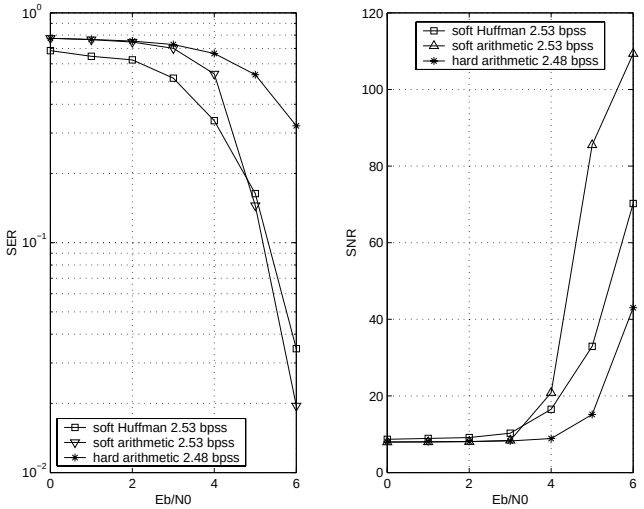

Fig. 5. SER and SNR performances of (a) soft arithmetic decoding, (b) hard arithmetic decoding and (c) soft huffman decoding ($\rho = 0.1$, 200 symbols, 100 channel realizations).

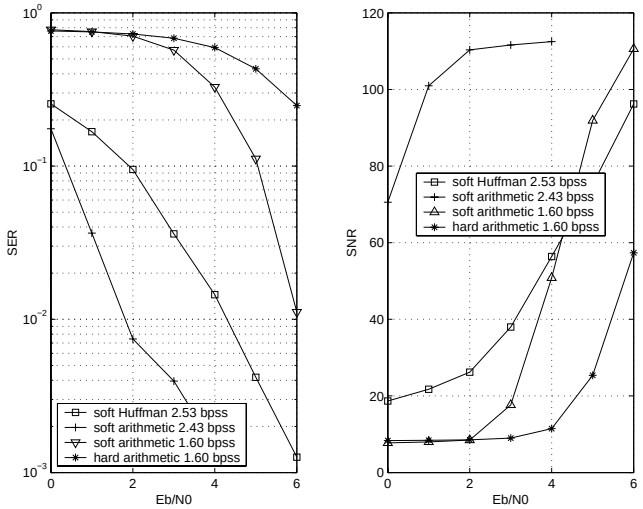

Fig. 7. SER and SNR performances of (a) soft arithmetic decoding (b) hard arithmetic decoding and (c) soft huffman decoding ($\rho = 0.9$, 200 symbols, 100 channel realizations).

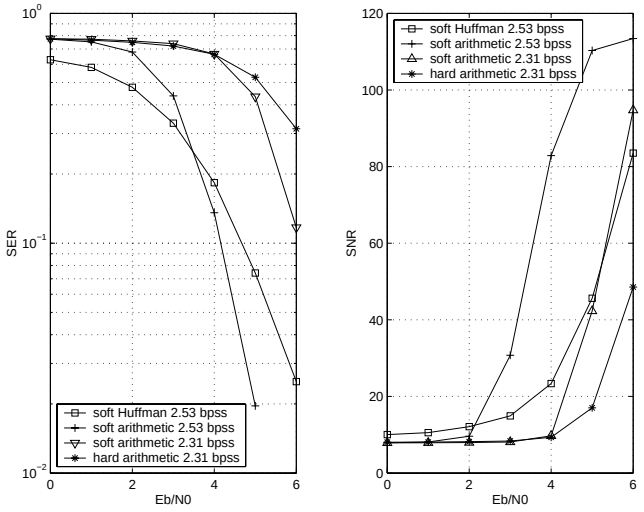

Fig. 6. SER and SNR performances of (a) soft arithmetic decoding, (b) hard arithmetic decoding and (c) soft huffman decoding ($\rho = 0.5$, 200 symbols, 100 channel realizations).

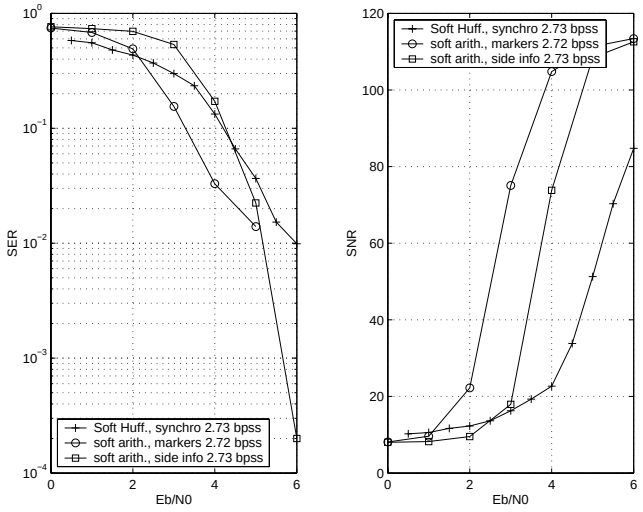

Fig. 8. SER and SNR performances of (a) soft arithmetic decoding with synchronization markers, (b) with side information and (c) soft huffman decoding with synchronization ($\rho = 0.5$).

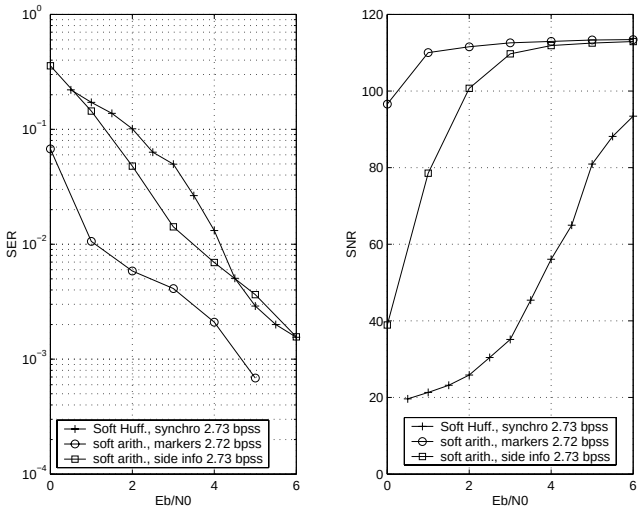

Fig. 9. SER and SNR performances of (a) soft arithmetic decoding with synchronization markers, (b) with side information and (c) soft huffman decoding with synchronization ($\rho = 0.9$).

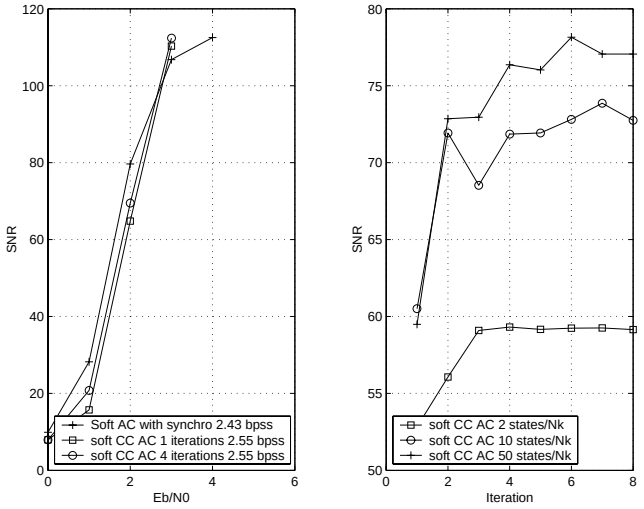

Fig. 10. SNR performances of iterative soft arithmetic decoding (a) in comparison with soft synchronization ($K = 200$), (b) as a function of the number of iterations for different pruning parameters ($w = 2, 10, 50$, $K = 20$).

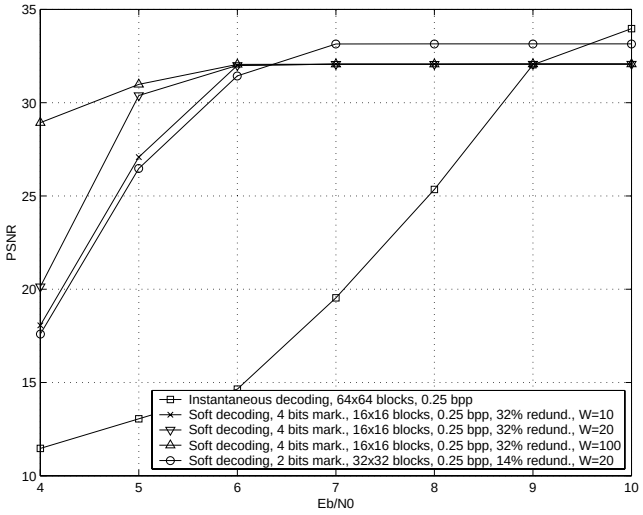

Fig. 11. PSNR performance for Lena 512x512 coded at 0.25 bpp with JPEG-2000.

(a) Original.

(b) JPEG-2000 encoded; no channel error; PSNR = 37.41 dB.

(c) JPEG-2000 encoded, AWGN channel (E_b/N_0 = 5 dB), PSNR = 11.25 dB.

(d) JPEG-2000 encoded and soft decoded, AWGN channel (E_b/N_0 = 5 dB), PSNR = 33.26 dB.

Fig. 12. Impact of channel errors on the visual quality (0.5 bpp).