Bitstream construction algorithms for transmission of Variable Length Codes over noisy channels

Hervé Jégou, Christine Guillemot

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Bitstream construction algorithms for transmission of Variable Length Codes over noisy channels

Hervé Jégou and Christine Guillemot

N°5357

Novembre 2004

Systèmes communicants

ISSN 0249-6399 ISRN INRIA/RR--5357--FR+ENG

Bitstream construction algorithms for transmission of Variable Length Codes over noisy channels

Hervé Jégou * and Christine Guillemot †

Systèmes communicants
Projet Temics

Rapport de recherche n ° 5357 — Novembre 2004 — 28 pages

Abstract: this paper addresses the issue of robust transmission of sources encoded with Variable Length Codes (VLCs) over error-prone channels. This paper describes bitstream construction methods offering different trade-offs of error-resilience, progressivity and flexibility. In contrast with related algorithms described in the literature, all the proposed methods have a linear complexity as the sequence length increases. The applicability of soft-input soft-output (SISO) and turbo decoding principles to resulting bitstream structures is investigated. In addition to error-resilience, the amenability of the bitstream construction methods to progressive decoding is considered. The VLC code has to be designed so that the symbol *energy* is mainly concentrated on the first bits of the symbol representation (i.e. on the first transitions of the corresponding codetree). Simulation results reveal high performances in terms of symbol error rate (SER) and mean square reconstruction error (MSE). These error-resilience and progressivity properties can be obtained without any penalty in compression efficiency.

Key-words: source coding, robust source coding, variable length codes, entropy codes, data compression, data communication

(Résumé : tsvp)

* prenom.nom@irisa.fr
† prenom.nom@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Algorithmes de construction de train binaire pour la transmission de codes à longueur variable sur canaux bruités

Résumé : cet article décrit des techniques de construction de train binaire offrant différents compromis de résistance aux erreurs, de progressivité et de flexibilité d'utilisation. Contrairement aux algorithmes similaires de la littérature, toutes les méthodes proposées ont un coût de calcul linéaire en fonction de la longueur de la séquence traitée. La possibilité d'appliquer des algorithmes de décodage souple à entrée et sortie souple est considérée. La possibilité d'obtenir des méthodes de construction de train binaire améliorant la progressivité est également considérée. Le code à longueur variable doit être construit de manière à ce que l'énergie du signal soit concentrée sur les premiers bits de la représentation binaire des mots de codes, c'est-à-dire sur les premières transitions de l'arbre binaire de codage. Les simulations montrent les excellentes performances de ce type d'approche pour les critères de taux d'erreur symbole et d'erreur quadratique moyenne. Ces propriétés de résistance aux erreur et de progressivité peuvent être obtenues sans pénalité supplémentaire en compression.

Mots-clé : codage de source, codage de source robuste, codes à longueur variable, codage entropique, compression de données, communications

1 Introduction

Entropy coding, producing VLC, is a core component of any compression scheme. The main drawback of VLCs is their high sensitivity 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 (SER). This phenomenon has motivated studies of the synchronization capability of VLCs as well as the design of codes with better synchronization properties [1][2][3]. Reversible VLCs [4][5][6], have also been designed to fight against de-synchronizations. Soft VLC decoding ideas, exploiting residual source redundancy (the so-called "excess-rate") as well as the inter-symbol dependency, have also been shown to reduce the "de-synchronization" effect [7][8].

For a given number of source symbols, the number of bits produced by a VLC coder is a random variable. The decoding problem is then to properly segment the noisy bitstream into measures on symbols and to estimate the symbols from the noisy sequence of bits (or measurements) that is received. This segmentation problem can be addressed by introducing a priori information in the bitstream, taking often the form of synchronization patterns. This a priori information is then exploited as constraints by the decoding process. One can alternatively, by properly structuring the bitstream, reveal and exploit constraints on some bit positions. This idea is applied in [9] to blocks within an image. A structure of fixed length size slots inherently creates hard synchronization points in the bitstream. The resulting bitstream structure is called EREC (Error-Resilient Entropy Codes). The principle can however be pushed further in order to optimize criteria of resilience, computing complexity and progressivity.

In this paper, given a VLC, we first focus on the design of transmission schemes of the codewords in order to achieve high SER and signal to noise ratio (SNR) performances in presence of transmission errors. The process for constructing the bitstream is regarded as a dynamic bit mapping between an intermediate binary representation of the sequence of symbols and the bitstream to be transmitted on the channel. The intermediate representation is obtained by assigning codewords to the different symbols. The decoder proceeds similarly with a bit mapping which, in presence of transmission noise, may not be the inverse of the mapping realized on the sender side, leading to potential decoder de-synchronization. The mapping can also be regarded as the construction of a new VLC for the entire sequence of symbols. Maximum error-resilience is achieved when the highest number of bit mappings (performed by coder and decoder) are deterministic. Notions of *constant* and *stable* sub-mappings with different synchronization properties in presence of transmission errors are introduced. This general framework leads naturally to several variants for the transmission scheme, exploiting the different mapping properties. By contrast with the EREC algorithm, all the proposed algorithms have a linear complexity as the sequence length increases. The bitstream construction methods presented allow for significant improvements in terms of SER and SNR with respect to classical transmission schemes where the variable length codewords are simply concatenated.

Another design criterion that we consider is the amenability of VLCs and of transmission schemes for progressive decoding. The notion of progressive decoding is very important for

image, video and audio applications. This is among the features that have been targeted in the embedded stream representation allowed in the JPEG2000 standard. For this purpose, an expectation-based decoding procedure is introduced. In order to obtain best progressive SNR performances in presence of transmission errors, the VLC codetrees have to be designed in such a way that most of the symbols *energy* is concentrated on transitions on the codetree corresponding to bits that will be mapped in a deterministic way. Given a VLC tree (e.g. a Huffman codetree [10]), one can build a new codetree, by re-assigning the codewords to the different symbols in order to satisfy at best the above criterion, while maintaining the same mean description length (mdl) for the corresponding source. This leads to codes referred to as *pseudo-lexicographic* codes. The lexicographic order can also be enforced by the mean of the Hu-Tucker algorithm [11]. This algorithm returns the lexicographic code having the smallest mdl. Note that for some sources, Hu-Tucker leads to the same mdl as the corresponding Huffman code.

The rest of the paper is organized as follows. Section 2 introduces the framework of bitstream construction, the notations and definitions used in the sequel. Several bitstream construction methods offering different trade-offs in terms of error-resilience and complexity are described in Section 3. The application of the SISO and the turbo decoding principles to the bitstream resulting from a constant mapping is described in Section 4. The code design is discussed in Section 5 and some choices are advocated. Simulation results are provided and discussed in section 6.

2 Problem Statement and Definitions

2.1 Notations

Let $\mathbf{S} = (S_1, \dots S_t, \dots S_K)$ be a sequence of source symbols taking their values in a finite alphabet $\mathcal{A}$ composed of $|\mathcal{A}|$ symbols, $\mathcal{A} = \{a_1, \dots a_i, \dots a_{|\mathcal{A}|}\}$. Let $\mathcal{C}$ be a binary variable length code designed for this alphabet, according to its stationary probability $\boldsymbol{\mu} = \{\mu_1, \dots \mu_i, \dots \mu_{|\mathcal{A}|}\}$. To each symbol S_t is associated a codeword $\mathcal{C}(S_t) = B_t^1 \dots B_t^{L(S_t)}$ of length $L(S_t)$. The sequence of symbols $\mathbf{S}$ is converted into an intermediate representation,

$$\mathbf{B} = (\mathbf{B_1}; \dots; \mathbf{B_K}), \tag{1}$$

where $\mathbf{B_t}$ is a column vector defined as

$$\mathbf{B_t} = \begin{pmatrix} B_t^1 \\ \vdots \\ B_t^{L(S_t)} \end{pmatrix}. \tag{2}$$

In the sequel, the emitted bitstream is denoted $\mathbf{E} = E_1 \dots E_{K_E}$ and the received sequence of noisy bits is denoted $\hat{\mathbf{E}} = \hat{E}_1 \dots \hat{E}_{K_E}$. Similarly, the intermediate representation on the receiver side is referred to as $\hat{\mathbf{B}}$.

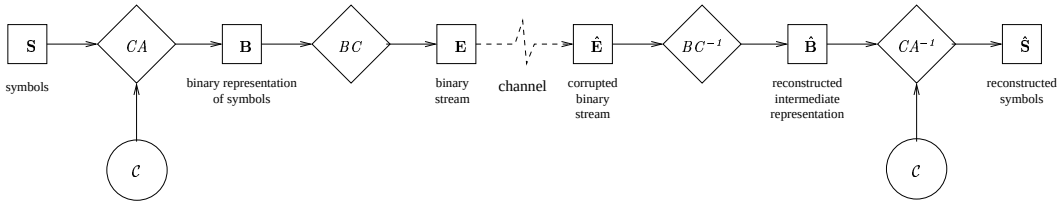


Figure 1: Coding and decoding building blocks.

We consider the general framework depicted in Fig. 1, where the coding process is decomposed into two steps: codeword assignment (CA) and bitstream construction (BC). In classical compression systems, the codewords produced are transmitted *sequentially*, forming a *concatenated* bitstream. Here, we focus on the problem of designing algorithms for constructing bitstreams that will satisfy various properties of resiliency and progressivity. Note that both the sequence length K and the length $K_E = \sum_{t=1}^{K} L(S_t)$ of the constructed bitstream $\mathbf{E}$ are assumed to be known on the decoder side. Note also that we reserve capital letters to represent random variables. Small letters will be used to denote the values or realizations of these variables.

2.2 Notions of constant and stable mapping

The BC algorithms can be regarded as dynamic bit *mappings* between the intermediate representation $\mathbf{B}$ and the bitstream $\mathbf{E}$. These mappings φ are thus defined on the set $\mathcal{I}(\mathbf{b}) = \{(t,l) \ / \ 1 \leq t \leq K, \ 1 \leq l \leq L(s_t)\}$ of tuples (t,l) that parses $\mathbf{b}$ (the realization of $\mathbf{B}$) as

$$\begin{array}{ccc} \mathcal{I}(\mathbf{b}) & \rightarrow & [1..K_E] \\ (t,l) & \mapsto & \varphi(t,l) = n, \end{array} \tag{3}$$

where n stands for a bit position of $\mathbf{E}$. Note that the index l can be regarded as the index of a layer (or a bit-plane) in the coded representation of the symbol. Similarly, the decoder proceeds with a bit mapping between the received bitstream $\hat{\mathbf{E}}$ and an intermediate representation $\hat{\mathbf{B}}$ of the received sequence of codewords. This mapping, referred to as ψ, depends on the noisy realization $\hat{\mathbf{b}}$ of $\hat{\mathbf{B}}$ and is defined as

$$\begin{array}{ccc} [1..K_E] & \rightarrow & \mathcal{I}(\hat{\mathbf{b}}) \\ n & \mapsto & \psi(n) = (t,l), \end{array} \tag{4}$$

where the set $\mathcal{I}(\hat{\mathbf{b}})$, in presence of bit errors, may not be equal to $\mathcal{I}(\mathbf{b})$. The composed function $\pi = \psi \, o \, \varphi$ is a dynamic mapping function from $\mathcal{I}(\mathbf{b})$ into $\mathcal{I}(\hat{\mathbf{b}})$. An element is decoded in the correct position iff $\pi(t,l) = (t,l)$. The error-resilience depends on the capability, in

presence of channel errors, to map a bitstream element n of $\hat{\mathbf{E}}$ to the correct position (t,l) in the intermediate representation $\hat{\mathbf{B}}$ on the receiver side. Hence, it depends on properties of the mapping π, in presence of bit errors, called here "constance" or "stability" properties. Since, for a given code $\mathcal{C}$, the entire mapping π can hardly verify these properties, it will be decomposed into a set of mapping functions, called *sub-mappings*, verifying these properties.

Definition 1: an element index (t,l) is said to be *constant* by $\pi = \psi \, o \, \varphi$, iff $n = \varphi(t,l)$ does not depend on the realization $\mathbf{b}$. Similarly, the bitstream index n is also said to be *constant*.

Let $\mathcal{I}_C$ denote the set of *constant* indexes. The restriction φ_C of φ to the definition set $\mathcal{I}_C$ and its inverse $\psi_C = \varphi_C^{-1}$ are also said to be *constant*. Such constant mappings can not be altered by channel noise: $\forall \hat{\mathbf{b}},\ (t,l) \in \mathcal{I}_C \Rightarrow \pi(t,l) = (t,l)$. Let $h_{\mathcal{C}}^-$ and $h_{\mathcal{C}}^+$ respectively denote the length of the shortest and of the longest codewords of the codetree. From the fact that, for each realization $\mathbf{b}$, $\mathcal{I}_C \subseteq \mathcal{I}(\mathbf{b})$, it follows that if π_C is a constant mapping, then the set $\mathcal{I}_C$ verifies the following property.

Property 1: $\mathcal{I}_C \subseteq [1..K] \times [1..h_{\mathcal{C}}^-]$.

Definition 2: an element index (t,l) is said to be *stable* by π iff $\varphi(t,l)$ only depends on $B_t^1, ...B_t^{l-1}$ and $\forall l'/1 \leq l' < l,\ (t,l')$ is *stable*.

Let $\mathcal{I}_S$ denote the set of *stable* indexes. A stable mapping φ_S can be defined by restricting the mapping φ to the definition set $\mathcal{I}_S$. A stable mapping verifies the following property. Let us assume that the first $l-1$ bits of a symbol s_t have been decoded without any error, i.e. $\hat{b}_t^1 = b_t^1, ...\hat{b}_t^{l-1} = b_t^{l-1}$, and that $(t,l) \in \mathcal{I}_S$. Note that $\hat{s}_t$ is not decoded yet. The decoder can estimate correctly the position $n = \varphi(t,l)$ where a bit element indexed by (t,l) has been mapped by the encoder. The inverse $(t,l) = \psi(n)$ is hence processed without any error. It also means that for the set of stable indexes, the error propagation is restricted to the symbol itself.

2.3 SER Bound

Let us consider the transmission of a VLC encoded source on a Binary Symmetric Channel (BSC) with a bit error rate (BER) p. Provided the mapping is stable, i.e. with no inter-symbol dependence, the probability that a symbol S_t is correctly decoded is given by $\mathbb{P}(\hat{S}_t = s_t | S_t = s_t) = (1-p)^{L(S_t)}$, leading to the following SER bound

$$\begin{aligned} \mathrm{SER}_{\mathrm{bound}}(\mathcal{C}) &= 1 - \sum_{a_i \in \mathcal{A}} \mu_i (1-p)^{L(a_i)} \\ &= p\, h_{\mathcal{C}} + \mathcal{O}(p^2), \end{aligned} \tag{5}$$

where $h_{\mathcal{C}}$ denotes the mdl of the code $\mathcal{C}$. This equation provides a lower bound in terms of SER when transmitting sources encoded with the code $\mathcal{C}$ on a BSC, assuming that simple

hard decoding is used. Note that this bound is lower than the SER that would be achieved with FLCs.

3 Bitstream construction algorithms

In this section, we describe bitstream construction (BC) algorithms offering different trade-offs in terms of error-resilience and complexity.

3.1 Constant MApping (CMA)

Given a code $\mathcal{C}$, the first approach aims at maximizing the cardinal of the definition set of the constant mapping φ_C, i.e. such that $\mathcal{I}_C = [1..K] \times [1..h_{\mathcal{C}}^-]$ (see Property 1). Notice first that a variable length codetree comprises a section of a fixed length equal to the minimum length of a codeword denoted $h_{\mathcal{C}}^-$, followed by a variable length section. A *constant* mapping can thus be defined as the composition of functions $\varphi_C : [1..K] \times [1..h_{\mathcal{C}}^-] \rightarrow [1..Kh_{\mathcal{C}}^-]$ and $\psi_C = \varphi_C^{-1}$ defined such that

$$(t, l) \rightarrow \varphi_C(t, l) = (l-1)K + t \tag{6}$$

The bits that do not belong to the definition set of φ_C can be simply concatenated at the end of the bitstream. The *constant* mapping φ_C defines a set of "hard" synchronization points.

Example 1: let $\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5\}$ be the alphabet of the source $\mathbf{S}_{(1)}$ with the stationary probabilities given by $\mu_1 = 0.4$, $\mu_2 = 0.2$, $\mu_3 = 0.2$, $\mu_4 = 0.1$ and $\mu_5 = 0.1$. This source has been considered by several authors, e.g. [1]. In [12], the 16 optimal codes (in the Huffman sense) available for this source have been named. All these codes have a mdl that is equal to 2.2. With the same notations as in [12], the codes referred to as $\mathcal{C}_5 = \{01, 00, 11, 100, 101\}$ and $\mathcal{C}_7 = \{0, 10, 110, 1110, 1111\}$ have been shown to exhibit interesting self-synchronization properties. The realization $\mathbf{s} = a_1a_4a_5a_2a_3a_3a_1a_2$ leads respectively to the sequence length $K_E = 18$ for code $\mathcal{C}_5$ and to the sequence length $K_E = 20$ for code $\mathcal{C}_7$. The respective intermediate representations associated to the sequence of symbols $\mathbf{s}$ are given in Fig. 2. The CMA algorithm proceeds with the mapping φ of the elements (b_t^l) to, respectively, positions $n = 1..18$ and $n = 1..20$ of the bitstream, as illustrated in Fig. 2. This finally leads to the bitstream $\mathbf{e}$=0110110010001110 01 for $\mathcal{C}_5$ and to the bitstream $\mathbf{e}$=011111101 110111010100 for $\mathcal{C}_7$. Note that the set $\mathcal{I}_C$ of constant indexes associated with $\mathcal{C}_5$ and $\mathcal{C}_7$ are respectively the set $[1..8] \times [1; 2]$ and the set $[1..8] \times [1; 1]$.

In contrast with classical transmission schemes where the codewords are concatenated, error propagation will only take place on the tuples (t, l) which do not belong to $\mathcal{I}_C$. The above mapping amounts to transmit the fixed length section of the codewords bit-plane per bit-plane. Hence, for a Huffman tree, the most frequent symbols will not suffer from de-synchronization. Moreover, the bitstream structure is amenable to soft-decision decoding using Bayesian estimation principles. We will come back on this point in Section 4.

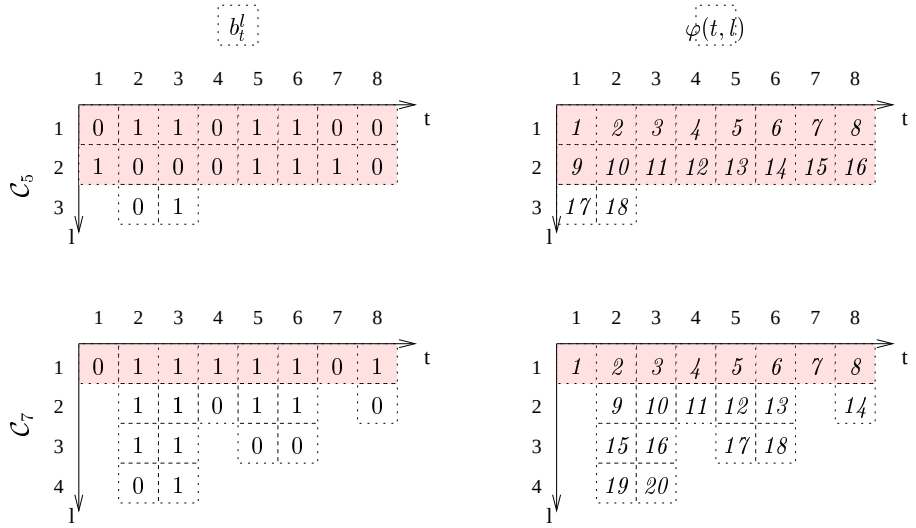


Figure 2: Example of intermediate representation $\mathbf{b}$ and corresponding mapping φ realized by the CMA algorithm. The set $\mathcal{I}_C$ of constant element indexes is highlighted in gray.

3.2 Stable MApping (SMA) algorithm

The CMA aims at maximizing the cardinal of the definition set $\mathcal{I}_C$. The error-resilience can be further increased by trying to maximize the number of *stable positions*, i.e., by minimizing the number of inter-symbol dependencies, according to Definition 2. The stability property can be guaranteed for a set $\mathcal{I}_S$ of element indexes (t,l) defined as $\mathcal{I}_S = \{(t,l) \in \mathcal{I}(\mathbf{b})/1 \leq t \leq K, 1 \leq l \leq l_s\} \cup \{(t,l) \in \mathcal{I}(\mathbf{b})/1 \leq t \leq K_s, l = l_s + 1\}$ for l_s and K_s verifying $l_s \times K + K_s \leq K_E$. Maximizing the expectation of $|\mathcal{I}_S|$ amounts to choosing $l_s = \lfloor \frac{K_E}{K} \rfloor$ and $K_s = K_E \ mod \ K$. Let us remind that $\mathcal{I}(\mathbf{b})$ is the definition set of the realization $\mathbf{b}$ of the intermediate representation $\mathbf{B}$. The set $\mathcal{I}_S$ can be seen as the restriction of $\mathcal{I}_F = ([1..K] \times [1..l_s]) \cup ([1..K_s] \times \{l_s + 1\})$, definition set of a mapping independent of $\mathbf{b}$, to $\mathcal{I}(\mathbf{b})$. Note that $|\mathcal{I}_F| = K_E$. On the sender side, the approach is thus straightforward and is illustrated in the example below. The decoder, knowing the values of the parameters l_s and K_s, can similarly compute the restriction of $\mathcal{I}_F$ to $\mathcal{I}(\hat{\mathbf{b}})$ instead of $\mathcal{I}(\mathbf{b})$.

Example 2: considering the source $\mathbf{S}_{(1)}$, the codes and the sequence of symbols of Example 1, the SMA *BC algorithm* leads to map the stable indexes as depicted in Fig. 3, where the notation $\varnothing$ stands for bitstream positions that have not been mapped during this stage. Remaining elements of the intermediate representation, here respectively 1 and 0001 for $\mathcal{C}_5$ and $\mathcal{C}_7$, are inserted in the positions identified by the valuation $\varnothing$, which respectively leads to the bitstream $\mathbf{e} = 01101100\ 10001110\ 10$ for $\mathcal{C}_5$ and to the bitstream $\mathbf{e} = 01111101\ 01101100\ 0111$

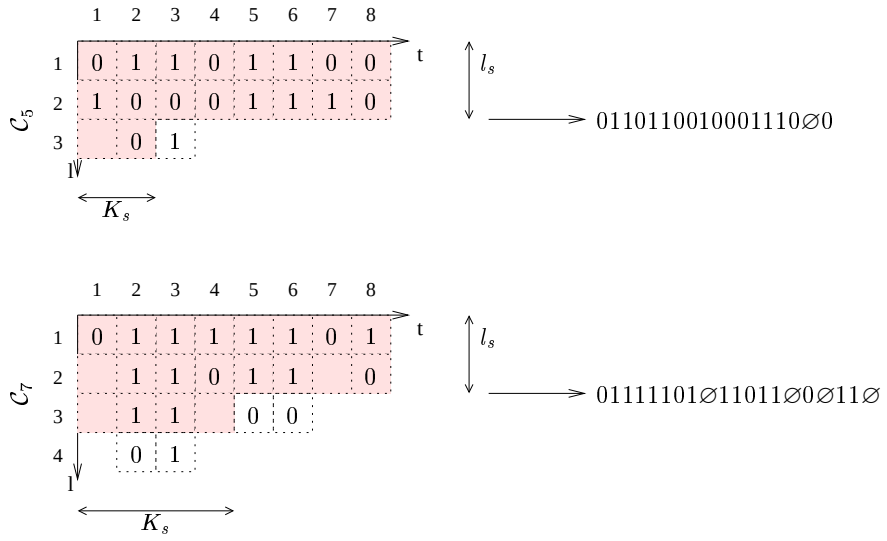


Figure 3: Definition set $\mathcal{I}_S$ (elements in gray) of the stable mapping (SMA algorithm).

for $\mathcal{C}_7$. In other words, the remaining elements of **b** (those that have not been mapped by φ_S), are inserted in positions of **e** that have not been taken as a mapped image of a tuple (t, l) in the *stable* mapping.

At that stage some analogies with the first step of the EREC algorithm [9] can be evidenced. The EREC algorithm structures the bitstream in M slots, the goal being to create hard synchronization points at the beginning of each slot. The EREC algorithm thus leads to the creation of a constant mapping on a definition set $|\mathcal{I}_C| = M\,h^- \leq K\,h^-$. Hence, it appears that, for a number of slots lower than K (the number of symbols), the number of bits mapped in a constant manner is not maximized. This suggests to use a constant mapping on the definition set $[1..K] \times [1..h_{\mathcal{C}}^-]$ and to apply EREC on slots for the remaining bits to be mapped. The corresponding algorithm is called CMA-EREC in the sequel. Note that, if $M{=}K$, CMA-EREC is identical to EREC applied on a symbol basis (which verifies $|\mathcal{I}_C| = K\,h^-$). If $M = 1$, CMA-EREC is identical to CMA. The choice of M has a direct impact on the trade-off resilience-complexity. Higher values of M lead, in average, to increased computing cost.

3.3 Stable MApping construction relying on a stack-based algorithm (SMA-stack)

This section describes an alternative to the above stable mapping algorithms offering the advantage of having a linear complexity ($\mathcal{O}(K)$) with respect to EREC. Let us consider two stacks, $Stack_b$ and $Stack_p$, respectively dedicated to store bit values b_t^l of $\mathbf{b}$ and bitstream positions n of $\mathbf{e}$. These stacks are void when the algorithm starts. Let us consider a structure of the bitstream $\mathbf{e}$ in M slots with $M = K$, i.e., with one slot per symbol s_t (the slots will be also indexed by t). The size of slot t is denoted m_t. There are K_s slots such that $m_t = l_s$ and $K - K_s$ slots such that $m_t = l_s + 1$. For each slot t, the algorithm proceeds as follows:

1. the first $min(m_t, L(s_t))$ bits of the codeword $\mathbf{b_t}$ associated to the symbol s_t are placed sequentially in slot t.

2. if $L(s_t) > m_t$, the remaining bits of $\mathbf{b_t}$ (i.e. $b_t^{m_t+1}...b_t^{L(s_t)}$) are put in the *reverse* order on the top of the stack $Stack_b$.

3. Otherwise, if $L(s_t) < m_t$, some positions of slot t remain unused. These positions are inserted on the top of the stack $Stack_p$.

4. While both stacks are not void, the top bit of $Stack_b$ is retrieved and inserted in the position of $\mathbf{e}$ indexed by the position that is on the top of the position stack $Stack_p$. Both stacks are updated.

After the last step, i.e. once the slot K has been processed, both stacks are void. The decoder proceeds similarly by storing (respectively retrieving) bits in a stack $Stack_b$ depending on the respective values of the codeword lengths $L(s_t)$ and of the slot size m_t. By construction of the slot structure, the number of stable elements is the same for both SMA algorithm and SMA-stack algorithm. The main difference between these algorithms relies in the way remaining elements are mapped. Using the proposed stack-based procedure increases the error resilience of the corresponding bits.

Example 3: let us consider again the source $\mathbf{S}_{(1)}$ of Example 1. Fig. 4 illustrates how the SMA-stack algorithm proceeds with the encoding if the code $\mathcal{C}_7$ is used. In this example, the slot structure has been chosen such that each slot is formed of contiguous bit positions, but it is not mandatory.

3.4 Layered bitstream

The previous BC algorithms allow to decrease the impact of the error propagation induced by the channel errors. Another approach that we consider here is the amenability of the BC framework to progressive decoding. This notion is closely related to the problem of SNR scalability. In section 4.3, we will see that the different bit transitions of a binary codetree convey different amount of energy. In section 5, the approach will be pushed further by expliciting code design algorithms aiming at optimizing an "energetic" criteria. In a context

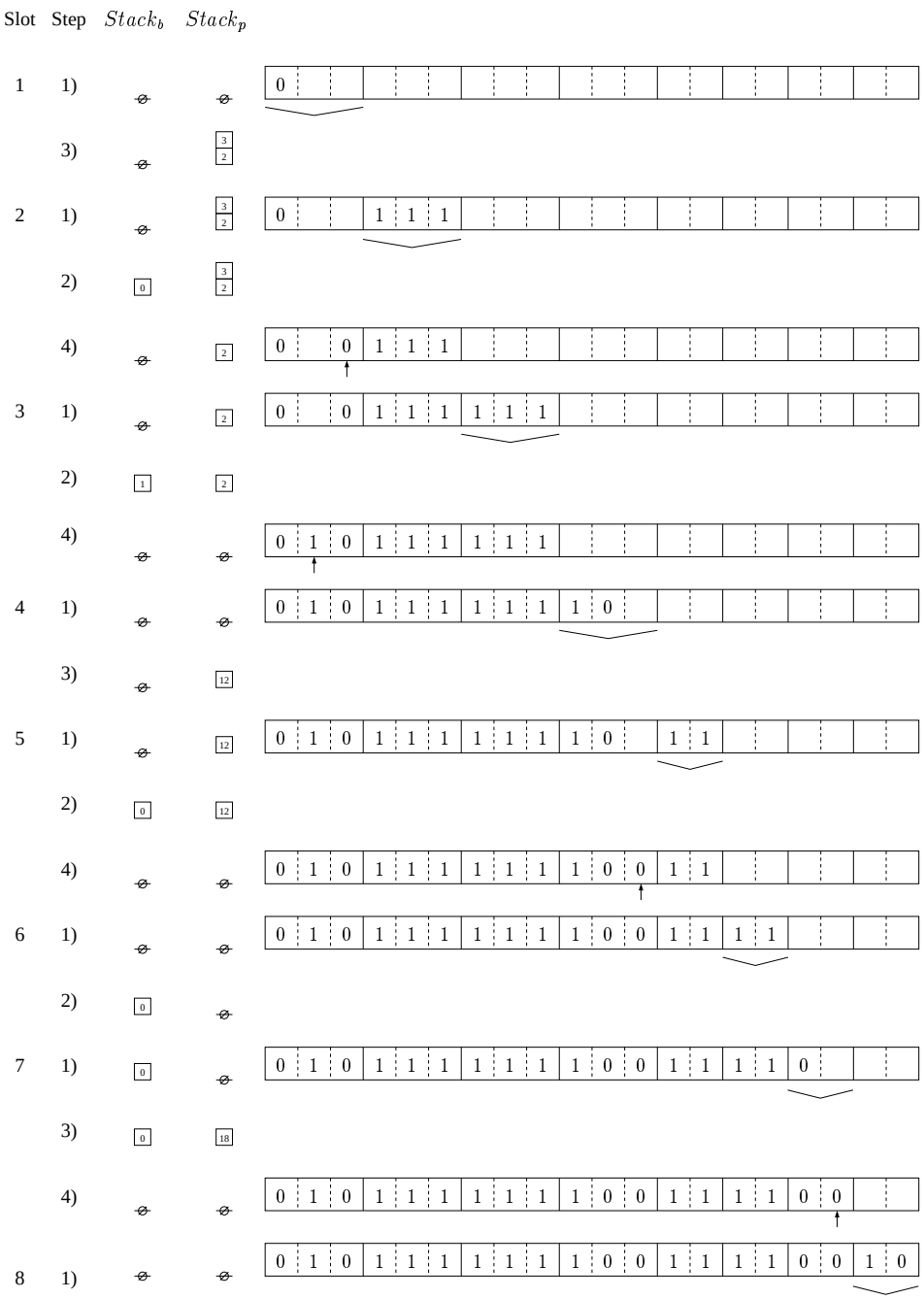


Figure 4: *(Example 3)* encoding of sequence $a_1 a_4 a_5 a_2 a_3 a_3 a_1 a_2$ using code $\mathcal{C}_7$ and algorithm SMA-stack.

of progressive decoding, the bit transitions bearing most of the reconstruction energy should be transmitted first. For example, the VLC codewords can be transmitted in a bit-plane fashion. The idea of transmitting VLCs using a bit-plane method is not new by itself (e.g., it has been considered in [13] in a context of video compression).

The layered decoding considered here is however more general that a problem of bit-plane organization of the bitstream. The approach consists in transmitting the bits forming the bitstream according to a given order $\succ$. Since to each bit transition one can associate an internal node of the codetree, this order is defined between internal nodes of the codetree. This order is the one according to which internal nodes of a VLC codeword are mapped in the bitstream. Thus, if $n_i \succ n_j$, all the bit transitions corresponding to internal node n_i are transmitted before any of the bit transitions corresponding to internal node n_j. The corresponding bits are transmitted so that this order is verified as the bit clock increases. This order induces a partition of the bit transitions into segments of same "priorities". Among a segment, bits are mapped sequentially. Note that this order may not be a total order: some bit transitions corresponding to distinct internal nodes may belong to the same segment. The order $\succ$ must verify the rule

$$n_j \not\succ n_i \text{ iff } n_j \in \mathcal{L}_i, \tag{7}$$

where $\mathcal{L}_i$ denotes the leaves deduced from n_i in the binary codetree corresponding to the VLC code. This rule is required because of the causality relationship between nodes n_i and n_j.

Example 4: let us consider again code $\mathcal{C}_5$. There are four internal nodes: the root /, 0, 1 and 10. These nodes are respectively referred to as n_0, n_1, n_2, n_3. A strict bit-plane [1] approach amounts to define the order $n_0 \succ n_1, n_2, \succ n_3$. Here, nodes n_1 and n_2 are mapped in the same segment. This order allows to ensure that all the bits corresponding to a given "bit-plane" are transmitted before any of the bits corresponding to a deeper "bit-plane". Using this order, the realization $\mathbf{s} = a_1a_4a_5a_2a_3a_3a_1a_2$ of Example 1 is coded within the following bitstream:

$$\underbrace{01101100}_{n_0}\underbrace{1000110}_{n_1 \text{ and } n_2}\underbrace{01}_{n_3}.$$

Another possible order is the order $n_0 \succ n_2 \succ n_3 \succ n_1$. In that case, since the order is total, segments are composed of homogeneous bit transitions, i.e. bit transitions corresponding to the same internal node in the codetree. Then, the realization $\mathbf{s}$ is transmitted as

$$\underbrace{01101100}_{n_0}\underbrace{0011}_{n_2}\underbrace{01}_{n_3}\underbrace{1010}_{n_1}.$$

Note that the CMA algorithm may be defined as the layered bitstream with the order such as $n_0 \succ n_1, n_2, n_3$. Note also that the concatenation of codewords correspond to the less restrictive order between internal nodes, i.e. $\forall (n_i, n_j), n_i \not\succ n_j$.

[1] The bit-plane approach for VLCs is somewhat similar to the one defined in the CABAC [14].

The layered construction bitstream is an efficient mean of enhancing the performances of Unequal Error Protection (UEP) schemes. In the literature, most authors have considered the UEP problem from the channel rates point of view, i.e., by finding the set of channel rates leading to the lowest overall distortion. For this purpose, Rate Compatible Punctunred Codes [15] are generally used. The UEP problem is then regarded as an optimization problem taking as an input the relative source importance for the reconstruction. Here, the UEP problem is seen from the source point of view. It is summarized by the following question: how the bitstream construction impacts the localization of energy so that UEP methods may apply efficiently? Since the bit segments have a different impact on the reconstruction, these segments act as sources of various importance. In the usual framework, concatenated bits corresponding to different internal nodes are not distinguished. Using the layered approach, one can differentiate the bits according to their impact on the reconstruction and subsequently applies the appropriate protection. The application of UEP techniques can hence be performed straightforwardly. The codetree itself has clearly an impact on the energy repartition, and has to be optimized in terms of the amount of source reconstruction energy beared the different bit transitions. The code design is treated in Section 5.

4 Decoding algorithms

4.1 Applying the soft decoding principles (CMA algorithm)

Trellis-based soft-decision decoding techniques making use of Bayesian estimators can be used to further improve the decoding SER and SNR performances. Assuming that the sequence $\mathbf{S}$ can be modeled as a Markov process, Maximum a Posteriori (MAP), MPM (Maximum of Posterior Marginals[2]) or Minimum of Mean Square Error (MMSE) estimators, using e.g. the BCJR algorithm [16], can be run on the trellis representation of the source model [17]. This section describes how the BCJR algorithm can be applied for decoding a bitstream resulting from a constant mapping (CMA algorithm).

Let us consider a symbol-clock trellis representation of the product model of the Markov source with the coder model [18]. For a given symbol index (or symbol clock instant) t, the state variable on the trellis is defined by the pair (S_t, N'_t) where N'_t denotes the number of bits *used to encode the first t symbols*. The value n'_t taken by the random variable N'_t is thus given by $n'_t = \sum_{t'=1}^{t} l(s_{t'})$. Notice that, in the case of classical transmission schemes where the codewords are simply concatenated, $n'_t{=}n$, where n is the current bit position in the bitstream $\mathbf{e}$.

The BCJR algorithm proceeds with the calculation of the probabilities $\mathbb{P}(S_t = a_i|\hat{e}_1; \ldots \hat{e}_{K_E})$, knowing the Markov source transitions probabilities $\mathbb{P}(S_t = a_i|S_{t-1} = a_{i'})$, and the channel transition probabilities $\mathbb{P}(\hat{E}_n = \hat{e}_n|E_n = e_n)$, assumed to follow a discrete memoryless channel (DMC) model. Using similar notations as in [16], the estimation proceeds with forward

[2] also referred to as symbol-MAP in the literature.

and backward recursive computations of the quantities

$$\alpha_t(a_i, n'_t) = \mathbb{P}(S_t = a_i; N'_t = n'_t; (\hat{e}_{\varphi(t',l)})), \quad (8)$$

where $(\hat{e}_{\varphi(t',l)})$ denotes the sequence of received bits in bitstream positions $n = \varphi(t', l)$, with $1 \leq t' \leq t$ and $1 \leq l \leq L(s_{t'})$, and

$$\beta_t(a_i, n'_t) = \mathbb{P}((\hat{e}_{\varphi(t',l)})|S_t = a_i; N'_t = n'_t), \quad (9)$$

where $(\hat{e}_{\varphi(t',l)})$ denotes the sequence of received bits in bitstream positions $n = \varphi(t', l)$, with $t+1 \leq t' \leq K$ and $1 \leq l \leq L(s_{t'})$. The recursive computation of the quantities $\alpha_t(a_i, n'_t)$ and $\beta_t(a_i, n'_t)$ requires to calculate

$$\begin{aligned} \gamma_t(a_j, n'_{t-1}, a_i, n'_t) &= \mathbb{P}(S_t = a_i; N'_t = n'_t; (\hat{e}_{\varphi(t,l)})_{1 \leq l \leq L(a_i)}|S_{t-1} = a_j; N'_{t-1} = n'_{t-1}) \quad (10) \\ &= \delta_{n_t - n_{t-1} - L(a_i)} \, \mathbb{P}(S_t = a_i|S_{t-1} = a_j) \prod_{l=1}^{L(a_i)} \mathbb{P}(\hat{e}_{\varphi(t,l)}|b_t^l). \end{aligned}$$

In the case of a simple concatenation of codewords, $\varphi(t, l) = n'_{t-1} + l$. When using a constant mapping, we have

$$\begin{aligned} \varphi(t, l) &= (l-1)\,K + t && \text{if } l \leq h_{\mathcal{C}}^- \\ &= (K-t)\,h_{\mathcal{C}}^- + n'_{t-1} + l && \text{otherwise} \end{aligned} \quad (11)$$

The product $\lambda_t(a_i, n') = \alpha_t(a_i, n')\,\beta_t(a_i, n')$ leads naturally to the posterior marginals $\mathbb{P}(S_t, N'_t|\hat{e}_1; \ldots \hat{e}_{K_E})$ and in turn to the MPM and MMSE estimates of the symbols S_t.

For the CMA algorithm, the fact that both the bit clock and the symbol clock are deduced from given state of the trellis is required to compute the entities $\gamma_t(a_j, n'_{t-1}, a_i, n'_t)$: it allows to regard the value of φ as a function of the state, as shown in Eqn. 11. This condition is verified by the bit/symbol trellis [6]. However, this property is not verified by some trellis of the literature, such as the trellis proposed by Balakirsky [19]. Thus, such a bit-level trellis can not be used to decode CMA-encoded bitstreams.

4.2 Turbo-decoding

The soft-decoding approach described above is used in a turbo-like scheme. For this purpose, extrinsic information is computed on bits. This amounts to compute the bit marginal probability bit $\mathbb{P}(e_t = 0 \vee 1|\hat{e}_1; \ldots \hat{e}_{K_E})$ instead of the symbol marginal probability. The SISO VLC then acts as the inner code, the outer code being a Recursive Systematic Convolutional Code (RSCC) of constraint length 5. In the last iteration only, the symbol per symbol output distribution $\mathbb{P}(S_t = a_i|\hat{e}_1; \ldots \hat{e}_{K_E})$ is processed instead of the bit marginal probability, in order to compute either the MPM either the MMSE estimators.

4.3 MMSE progressive decoding

The above approach aims at reducing the SER. However, it does not take into account MSE performances in a context of progressive decoding. Progressive decoding of VLC can be realized by considering an expectation-based approach as follows. Notice that VLC codewords can be decoded progressively by regarding the bit generated by the transitions at a given level of the codetree as a bit-plane or a layer.

Let us assume that the l first bits of a codeword have been received without error. They correspond to an internal node n_j of the codetree. Let $\mathcal{L}_j$ and $\tilde{\mu_j} = \sum_{n_i \in \mathcal{L}_j} \mu_i$ respectively denote the leaves deduced from n_j and the probability associated to the node n_j. Then the optimal reconstruction value $\tilde{a}_j$ is given by

$$\tilde{a}_j = \frac{1}{\tilde{\mu_j}} \sum_{n_i \in \mathcal{L}_j} \mu_i \, a_i. \tag{12}$$

The corresponding mean square error (MSE), referred to as Δ_j, is given by the variance of the source knowing the first bits, i.e., by

$$\Delta_j = \frac{1}{\tilde{\mu_j}} \sum_{a_i \in \mathcal{L}_j} \mu_i \, (a_i - \tilde{a}_j)^2. \tag{13}$$

Let us consider the codetree modeling the decoding process. The reception of one bit will trigger the transition from a parent node n_j to children nodes $n_{j'}$ and $n_{j''}$ depending on the bit realization. The corresponding reconstruction MSE is then decreased as $\Delta_j - \Delta_{j'}$ or $\Delta_j - \Delta_{j''}$ depending on the value of the bit received. This amounts to an MMSE decoding on the corresponding node of the codetree. Given a node n_j, the expectation δ_j of the MSE decrease for the corresponding transition T_j is given by

$$\delta_j = \Delta_j - \frac{\tilde{\mu}_{j'} \, \Delta_{j'} + \tilde{\mu}_{j''} \, \Delta_{j''}}{\tilde{\mu}_{j'} + \tilde{\mu}_{j''}}. \tag{14}$$

The term δ_j can be seen as an amount of signal *energy*. If all the bits are used for the reconstruction, the MSE equals 0, which leads to $\mathrm{var}(\mathbf{S}) = \Delta_{\mathrm{root}} = \sum_{n_j} \tilde{\mu_j} \, \delta_j$, which can also be deduced from Eqn. 14. The total amount δ_l^* of reconstruction *energy* corresponding to a given layer l of a VLC codetree can then be calculated as the weighted sum of *energies* given by transitions corresponding to the given layer:

$$\delta_l^* = \sum_{T_j \text{ in layer } l} \tilde{\mu}_j \, \delta_j. \tag{15}$$

Remark: Note that the MMSE estimation can be further improved by applying a BCJR algorithm on the truncated bitstream, setting the transitions on the trellis that correspond to the non-received bits to their posterior marginals or to an approximated value of $\frac{1}{2}$.

Note also that, if the quantities K and K_E are both known on the receiver side, error propagation can be detected if the termination constraints are not verified. Here, by termination constraints, we mean that the K_E bits of $\mathbf{E}$ must lead to decode K symbols of $\mathbf{S}$.

In the case where the termination constraint is not verified, it may be better to restrict the expectation-based decoding to the bits that cannot be de-synchronized (i.e., bits mapped with a constant or stable sub-mapping).

5 Code design

As previously suggested, the code should be optimized for the following reasons:

1. In order to maximize the SNR performance in presence of transmission errors, the code $\mathcal{C}$ should be such that it concentrates most of the *energy* on the bits (or codetree transitions) that will not suffer from de-synchronization. In particular, if the bits that concentrate most of the energy correspond to the first bit transition of the binary codetree, the concept of *most significant* bits can also apply for VLC codewords.

2. Similarly, the progressivity depends on the amount of energy transported by the first transmitted bits. That is why the code design should be such that few bits gather most of the reconstruction energy, and these bits should be transmitted first. For this purpose, we will assume that the layered approach proposed in section 3.4 will be used.

3. Finally, a better energy concentration enhances the performances of the UEP techniques: since these techniques exploit the fact that different sources (here segments of bits) have various priorities, the code should be designed to enhance the heterogeneity of the bit transitions in terms of reconstruction energy.

In this section, we will regard the code optimization problem as a simplified problem consisting in maximizing the values δ_l^* for the first codetree transitions. This problem can be addressed by optimizing Eqn. 15, either with the Binary Switching Algorithm [20] or with a simulated annealing algorithm (e.g., [21]). The optimization has to be processed jointly for every layer. Hence, this multi-criteria optimization requires that some weights are provided to each layer l of Eqn. 15. The weights associated to bit transition depend on the application. In the following, we propose two simplified approaches, led by the consideration that the lexicographic order, separating the smaller values from the greater values, -in general- concentrates most of the energy in the first layers. Obviously, a lexicographic order is relevant only for a scalar alphabet. The first approach consists in finding an optimal code (in the Huffman sense) that aims at verifying a lexicographic order. The second approach consists in using Hu-Tucker [11] codes to enforce the lexicographic order.

5.1 Pseudo-lexicographic (p-lex) codes

Let us consider a classical VLC (e.g. using a Huffman code), associating a codeword of length l_i to each symbol a_i. One can re-assign the different symbols a_i of the source alphabet to the VLC codewords in order to try to best satisfy the above criteria. In this part, the re-assignment is performed under the constraint that the lengths of the codewords associated to the different symbols are not affected, in order to preserve the compression performance

Huffman				*p-lex*				Hu-Tucker			
node	$\mathbb{P}(n_j)$	$\tilde{a_j}$	δ_j	node	$\mathbb{P}(n_j)$	$\tilde{a_j}$	δ_j	node	$\mathbb{P}(n_j)$	$\tilde{a_j}$	δ_j
$\varnothing$	1	0.000	0.022	$\varnothing$	1	0.000	0.297	$\varnothing$	1	0.000	0.626
0	0.434	+0.170	0.477	0	0.566	−0.4776	0.013	0	0.5	−0.791	0.228
1	0.566	−0.131	0.224	1	0.434	+0.6220	0.119	1	0.5	+0.791	0.228
00	*0.160*	*+1.074*		00	0.292	−0.5903	0.285	00	0.226	−1.317	0.145
01	*0.274*	*−0.358*		*01*	*0.274*	*−0.358*		*01*	*0.274*	*−0.358*	
10	*0.274*	*+0.358*		*10*	*0.274*	*+0.358*		*10*	*0.274*	*+0.358*	
11	0.292	−0.590	0.285	*11*	*0.160*	*+1.074*		11	0.226	+1.317	0.145
110	0.131	0.000	2.294	*000*	*0.160*	*−1.074*		000	0.066	−1.911	0.072
111	*0.160*	*−1.074*		001	0.131	0.000	2.294	*001*	*0.160*	*−1.074*	
								110	*0.160*	*+1.074*	
								111	0.066	+1.911	0.072
1100	*0.055*	*+1.791*		0010	0.077	−1.281	0.654	*0000*	*0.011*	*−2.511*	
1101	0.077	−1.281	0.654	*0011*	*0.055*	*+1.791*		*0001*	*0.055*	*−1.791*	
								1110	*0.055*	*+1.791*	
								1111	*0.011*	*+2.511*	
11010	0.022	0.000	6.306	*00100*	*0.055*	*−1.791*					
11011	*0.055*	*−1.791*		00101	0.022	0.000	6.306				
110100	*0.011*	*−2.511*		*001010*	*0.011*	*−2.511*					
110101	*0.011*	*+2.511*		*001011*	*0.011*	*+2.511*					

Table 1: Definition of Huffman, *p-lex* Huffman and Hu-Tucker codes for a quantized Gaussian source. Leaves (e.g., alphabet symbols) are in italic.

of the code. A new codetree, referred to as a *pseudo-lexicographic* (*p-lex*) VLC codetree, can be constructed as follows. Starting with the layer $l = h_{\mathcal{C}}^{+}$, the nodes (including leaves) of depth l in the codetree are sorted according to their expectation value given in Eqn. 12. Pairs of nodes are grouped according to the resulting order. The expectation values corresponding to the parent nodes (at depth $l-1$) are in turn computed. The procedure continues until the codetree is fully constructed. Grouping together nodes having close expectation values in general contributes to increase the energy or information carried on the first transitions on the codetree.

Example 5: Let us consider a Gaussian source of zero-mean and standard deviation 1, uniformly quantized on 8 cells partitioning the interval $[-3,+3]$. The subsequent discrete source is referred to as $\mathbf{S}_{(2)}$ in the sequel. Probabilities and reconstruction values associated to source $\mathbf{S}_{(2)}$ are respectively given by

$$\mathcal{A} = \{-2.5112, -1.7914, -1.0738, -0.3578, 0.3578, 1.0738, 1.7914, 2.5112\}$$

and

$$\boldsymbol{\mu} = \{0.01091, 0.05473, 0.16025, 0.27411, 0.27411, 0.16025, 0.05473, 0.01091\}.$$

The Huffman algorithm leads to construct the code $\{110100, 11011, 111, 01, 10, 00, 1100, 110101\}$ detailed in table 1. The *p-lex* algorithm proceeds as follows:

Bit transition level	nodes	selected aggregation	expectation	probability
6	$\{a_1, a_8\}$	$(a_1, a_8) \rightarrow n_7$	$\mathbb{E}(n_7) = 0.000$	$P(n_7) = 0.022$
5	$\{n_7, a_2\}$	$(a_2, n_7) \rightarrow n_6$	$\mathbb{E}(n_6) = -1.281$	$P(n_6) = 0.077$
4	$\{n_6, a_7\}$	$(n_6, a_7) \rightarrow n_5$	$\mathbb{E}(n_5) = 0.000$	$P(n_5) = 0.131$
3	$\{n_5, a_3\}$	$(a_3, n_5) \rightarrow n_4$	$\mathbb{E}(n_4) = -0.590$	$P(n_4) = 0.292$
2	$\{n_4, a_4, a_5, a_6\}$	$(n_4, a_4) \rightarrow n_3$	$\mathbb{E}(n_3) = -0.478$	$P(n_3) = 0.566$
2	$\{a_5, a_6\}$	$(a_5, a_6) \rightarrow n_2$	$\mathbb{E}(n_2) = +0.622$	$P(n_2) = 0.434$
1	$\{n_2, n_3\}$	$(n_2, n_3) \rightarrow n_1$	$\mathbb{E}(n_1) = 0.000$	$P(n_1) = 1.000$

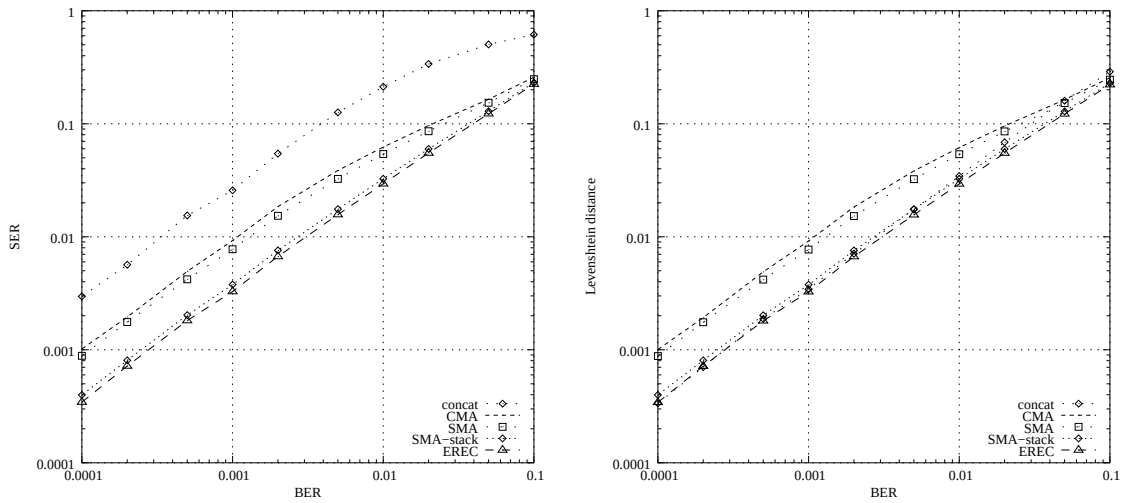
The reconstruction values obtained with this code are given in table 1. Note that both the Huffman code and the code constructed with the *p-lex* algorithm have a mdl of 2.521, while the source entropy is 2.471. The corresponding bit transition energies δ_j are also depicted. The reconstruction of symbols using the first bit only is improved by 1.57 dB (MSE is equal to 0.631 for the p-lex code instead of 0.906 for the Huffman code).

5.2 Hu-Tucker codes

For a given source, it may occur that the previous procedure leads to a code that preserves the lexicographic order in the binary domain. For example, it is well known that if the probability distribution function is a monotone function of symbol values, then it is possible to find a lexicographic code with the same compression efficiency as Huffman codes. But in the general case, it is not possible. In this section, Hu-Tucker [11] codes are used to enforce the lexicographic order to be preserved in the bit domain. The resulting codes may be sub-optimal, with the mdl falling into the interval $[h, h+2[$, where h denotes the entropy of the source. Thus, for the source $\mathbf{S}_{(2)}$, the mdl of the corresponding Hu-Tucker code is 2.583, which corresponds to a penalty in terms of mdl of 0.112 bit per symbol, against 0.050 for the Huffman code. The counterpart is that these codes have strong progressivity features: the energy is concentrated on the first bit transitions (see 1). Thus, for source $\mathbf{S}_{(2)}$, the reconstruction with the first bit only offers an improvement of 4.76 dB over Huffman codes.

6 Simulation results

The performances of the different codes and *BC* algorithms have been evaluated in terms of SER, SNR and Levenshtein distance with Source $\mathbf{S}_{(1)}$ and Source $\mathbf{S}_{(2)}$ (quantized Gaussian source), introduced in Example 1 and Example 5 respectively. Let us recall that the Levenshtein distance [22] between two sequences is the minimum number of operations (e.g., symbol modifications, insertions and deletions) required to transform one sequence into the other. Unless the number of simulations is explicitely specified, the results shown are averaged over 100000 channel realizations and over sequences of 100 symbols. Since the source realizations are distinct, the number of emitted bits K_E is variable. Most of the algorithms that have been used to produce these simulation results are available on the web site [23].



(a) (b)

Figure 5: SER (a) and Levenshtein distance (b) performances of the different BC schemes for source $\mathbf{S}_{(1)}$ of Example 1 (code $\mathcal{C}_5$)

6.1 Error-resilience of algorithms CMA, SMA and SMA-stack for p-lex Huffman and Hu-Tucker codes

The first set of experiments aimed at evaluating the performances in terms of SER, Levenshtein distance and SNR obtained by the different BC algorithms CMA, SMA and SMA-stack. Fig. 5 and Fig. 6 respectively show for Source $\mathbf{S}_{(1)}$ and Source $\mathbf{S}_{(2)}$ the SER obtained with the algorithms CMA, SMA and SMA-stack in comparison with the concatenated scheme and a solution based on EREC [9] applied on a symbol (or codeword) basis, for channel error rates going from 10^{-4} to 10^{-1}. In Fig. 5, the results in terms of SER and normalized distance have been obtained with code $\mathcal{C}_5$ (cf. Example 1). Fig. 6 depicts the results obtained for a Huffman code optimized using the codetree optimization described in Section 5.1. This code is given in table 1.

In both figures, it appears that the concatenated scheme can be advantageously replaced by the different BC algorithms described above. In particular, the SER performance of the SMA-stack algorithm approaches the one obtained with the EREC algorithm applied on a symbol basis (which itself already outperforms EREC applied on blocks of symbols), for a quite lower computing cost. Similarly, it can be observed in Fig. 7 that the best SNR values are obtained with Hu-Tucker codes used jointly with the EREC algorithm. It can also be observed that the SMA-stack algorithm leads to very similar error-resilience performances. The results confirm that error propagation affects a smaller amount of reconstruction energy.
Remarks:

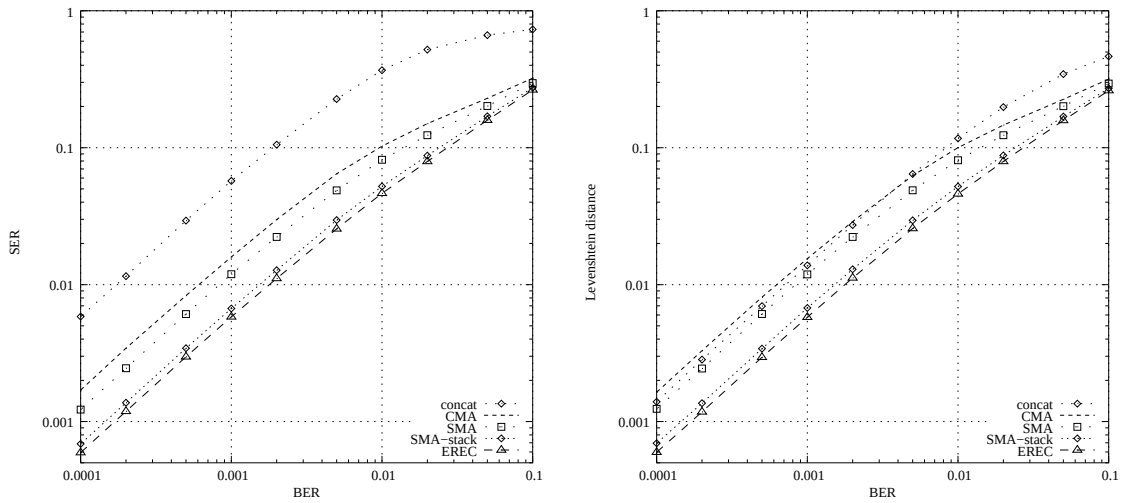
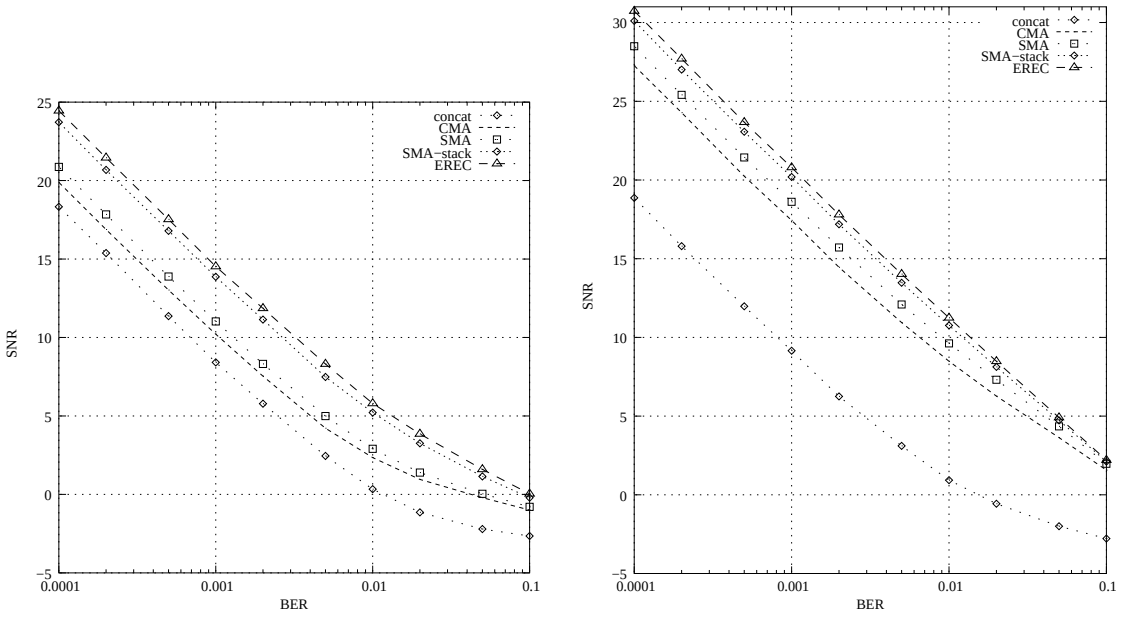


Figure 6: SER (a) and Levenshtein distance (b) performances of the different BC schemes with a quantized gaussian source (source $\mathbf{S}_{(2)}$ encoded with Huffman codes).

- for Source $\mathbf{S}_{(2)}$, Huffman and *p-lex* Huffman codes lead to the same error-resilience performances: the amount of energy conveyed by the bits mapped during the constant stage is identical for both codes. This fact can be deduced from table 1. However, for a large variety of sources, the *p-lex* Huffman codes lead to better results.
- The layered bitstream construction has not been included in this comparison: the layered bistream construction offers improved error-resilience in a context of UEP. Simulation results depicted in Fig. 5, Fig. 6 and Fig. 7 assume that no channel code has been used together with the source code.

6.2 Progressivity performances of CMA and layered algorithms and impact of the code design

The second set of experiments aimed at comparing the amenability to progressive decoding (using expectation-based decoding) of the CMA and layered solutions with Huffman, p-lex and Hu-Tucker codes with respect to a concatenated scheme. These codes are also compared against lexicographic FLCs. for which the first bit transitions notably gather most of the energy. For the layered approach, nodes have been sorted according to the value of δ_j, under the constraint of Eqn. 7. Let us recall that the values of δ_j are provided in table 1. The corresponding orders between nodes (identified by the path in the codetree) are given hereafter. The corresponding values of δ_j are also given.



(a) (b)

Figure 7: SNR performances of the different *BC* schemes (with pseudo-lexicographic Huffman) for *p*-lex Huffman codes (a) and Hu-Tucker codes (b).

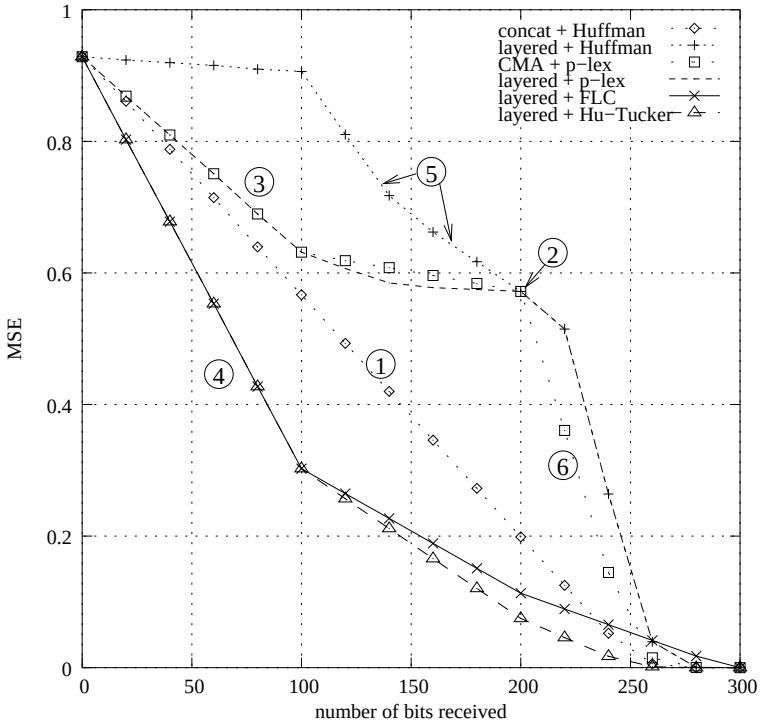


Figure 8: Progressive MSE performances/Energy Repartition profiles of the different codes and *BC* schemes without transmission noise.

Huffman	$\varnothing$	$\prec$	0	$\prec$	1	$\prec$	11	$\prec$	110	$\prec$	1101	$\prec$	11010
	0.022		0.477		0.224		0.285		2.294		0.654		6.306
p-lex Huffman	$\varnothing$	$\prec$	1	$\prec$	0	$\prec$	11	$\prec$	110	$\prec$	1101	$\prec$	11010
	0.297		0.119		0.013		0.285		2.294		0.654		6.306
Hu-Tucker	$\varnothing$	$\prec$	0	$\prec$	1	$\prec$	00	$\prec$	11	$\prec$	000	$\prec$	111
	0.626		0.228		0.228		0.145		0.145		0.072		0.072
FLC	$\varnothing$	$\prec$	0	$\prec$	1	$\prec$	01	$\prec$	10	$\prec$	00	$\prec$	11
	0.626		0.190		0.190		0.119		0.119		0.072		0.072

The choice of the order has a major impact on the progressivity. The proposed orders are the ones that provide the best progressivity performances. Fig. 8 shows the respective MSE in terms of the number of bits received for different approaches. The performance obtained is better than the one obtained with a bit-plane FLC transmission, but with higher compression efficiency. The following points, identified on Fig. 8 with a number, are of interests:

1. The concatenation, which does not differentiate the bits, leads to have the MSE performance that linearly decreases as the number of received bits increases.

2. Several curves converge to this point. This is due to the fact that the number of bits received at this point corresponds to the fixed length portions of Huffman and p-lex Huffman codes, and that these codes have the same energy concentration properties on the next bit transitions (see table 1).

3. For this source distribution, using the *p-lex* code instead of the usual Huffman code amounts to transfer some *energy* from layer 2 (bits 101 to 200) to layer 1 (bits 1 to 100)

4. For both the FLC and the Hu-Tucker code, the first bit transition is a bit that separates the negative values from the positive values in a symmetrical alphabet.

5. Here, the segment of bits going from bit 101 to bit 200 for the Huffman code with layered BC appears to be separated into two segments, corresponding respectively to bit transitions of node 0 and node 1. The choice of the order $1 \succ 0$ instead of $0 \succ 1$ leads to swap the two segments, which is a poorer choice.

6. For the Huffman and p-lex Huffman codes, the bit transitions corresponding to nodes with a high depth in the codetree bear a lot of reconstruction energy. As a result, the concatenation of codewords offers in general better performances than the layered transmission.

To conclude, this figure evidences the efficiency of Hu-Tucker codes transmitted with a layered BC scheme with respect to classical VLCs and transmission schemes. Let us underline again that the performance of UEP schemes strongly depend on the "energy repartition", which is another interpretation of Fig. 8. Note that the performance gap between p-lex Huffman codes and Hu-Tucker codes strongly depends on the source statistical distribution.

6.3 Error-resilience with CMA

The third set of experiments aimed at evaluating the efficiency, in terms of SER, of the CMA BC algorithm with an MPM decoder. The results are compared against those obtained with hard decoding and with MPM decoding of the same sequences but with concatenated codewords. The MPM decoder proceeds as described in Section 4.1 (no pruning is done here). For this set of experiment, we have considered a quantized Gauss-Markov source with a correlation factor $\rho = 0.5$, and sequences of 100 symbols. Fig. 9 shows the significant gain (SER divided by a factor close to 2) obtained with the CMA structure with respect to the concatenated bitstream structure for the same decoding complexity (the number of states in the trellis is strictly the same for both approaches).

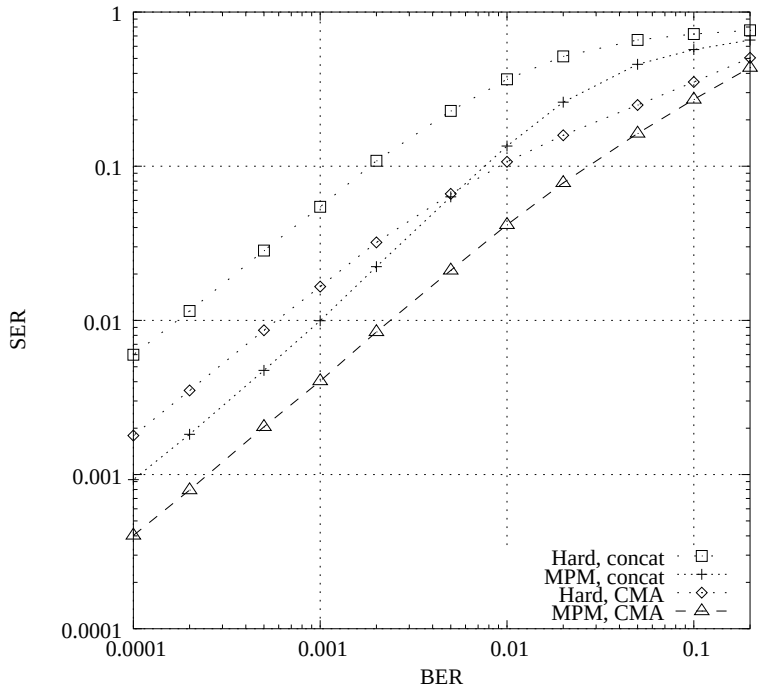
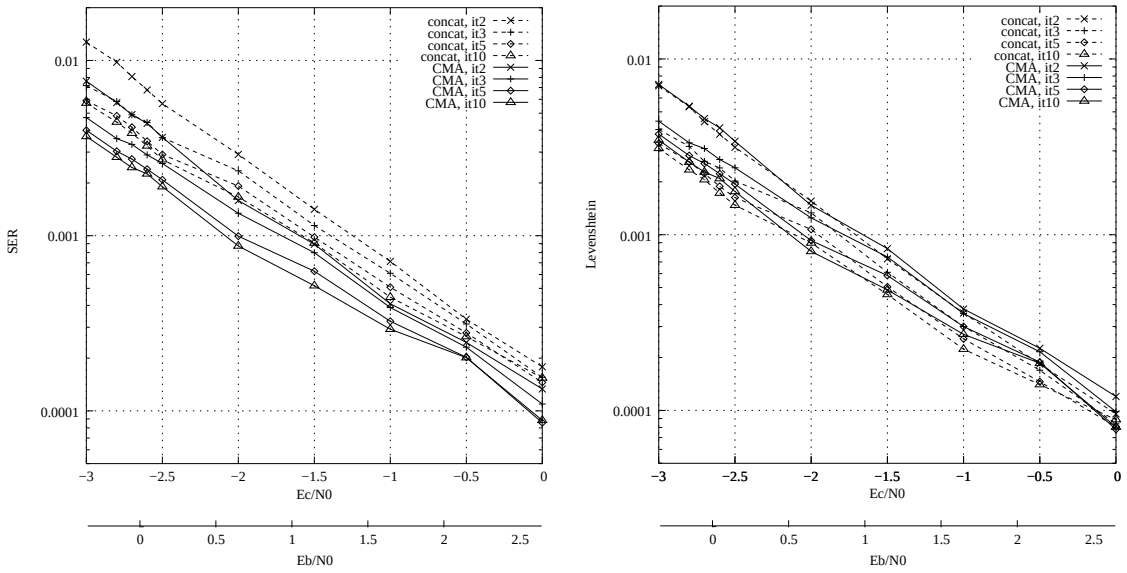


Figure 9: SER obtained by MPM decoding of a bitstream constructed by the CMA algorithm in comparison with those obtained with a concatenated bitstream structure (Gauss-Markov source with correlation $\rho = 0.5$).



(a) (b)

Figure 10: SER (a) and Levenshtein distance (b) obtained by two turbo-decoding schemes, both of them composed of a source coder and a RSCC: 1) a classical joint-source channel turbo-decoding, where the VLC codewords are concatenated ; 2) a joint-source channel where the VLC codewords are encoded/decoded with the CMA bitstream construction algorithm.

6.4 Turbo-decoding performances of CMA

The last set of experiments aimed at comparing the amenability to turbo-decoding (using expectation-based decoding) of the CMA solution with respect to a concatenated scheme, both using Huffman codes. The decoding algorithm used for this simulation has been described in section 4.1 and section 4.2. This set of experiments has been processed for a first-order Markov source governed by its transition probabilities. This source takes its value in an alphabet composed of three symbols. The matrix of transition probabilities is defined as

$$\mathbb{P}(S_t/S_{t-1}) = \begin{bmatrix} 0.94 & 0.18 & 0.18 \\ 0.03 & 0.712 & 0.108 \\ 0.03 & 0.108 & 0.712 \end{bmatrix}.$$

This source has been proposed by Murad and Fuja [7]. The channel considered here is an AWGN channel. Results have been averaged over 10000 realizations of sequences of length 100. Fig. 10 shows the SER performances corresponding to these simulations. This figure evidences the clear advantage in this context of the turbo-VLC based on the CMA bitstream construction with respect to classical turbo-VLC scheme based on concatenation for the SER measure. The improvement, expressed in terms of $Eb/N0$, is around 0.4dB. This gain is

BC Algorithm	Linear complexity	Mandatory parameters	Soft Decoding Algorithms	Main purpose
Concatenation	YES	K or K_E	Bit/symbol trellis [6] bit-level trellis [19], ...	
CMA	YES	K	Bit/symbol trellis	Error-resilience
SMA	YES	K and K_E	?	Error-resilience
SMA-stack	YES	K and K_E	?	Error-resilience
Layered	YES	K	?	Progressivity
EREC	NO	K and K_E	?	Error-resilience

Table 2: *BC* algorithms features.

maintained as the number of iterations grows, and for the same complexity. Since no pruning is done there, algorithms based on suboptimal strategies (e.g., Balakirsky trellis [19]) would lead to poorer results. For the Levenshtein distance measure, the concatenation leads to sligthy better results. Since in most applications, it is crucial to consider the SER rather than the Levenshtein distance, the advantage of the concatenation over the BC algorithm in this context must obviously be tempered.

7 Conclusion

In this paper, we have introduced a bitstream construction framework for the transmission of VLC-encoded sources over noisy channels. Several practical algorithms with different trade-offs in terms of error-resilience, progressivity and feasibility of soft decoding have been described. The features of these algorithms are summarized in Table. 2. Unlike the EREC algorithm, the complexity of the proposed algorithms evolves linearly with the length of the sequence. For the CMA algorithm, the corresponding coding processes can be easily modeled under the form of stochastic automata which are then amenable for running MAP estimation and soft-decision decoding techniques. Together with an MPM decoder, the CMA algorithm has been shown to increase the error-resiliency performance in terms of SER in comparison with a similar decoder for a concatenated bitstream, and this at no cost in terms of complexity. The approach has also been shown to offer improved SER performances in a turbo-VLC setup. For the other bitstream construction algorithms, i.e. SMA, EREC and SMA-stack, the design of efficient tractable soft and turbo decoding algorithms is a challenging problem. The code design has been shown to have a very high impact on the

error-resilience, the progressivity and the amenability to enhance the UEP schemes. Hu-Tucker codes have been shown to be a good choice for these purposes. As a perspective, this framework could hopefully help in optimizing the source binarization step of modern image and video coders (EBCOT,CABAC).

References

[1] J. Maxted and J. Robinson, "Error recovery for variables length codes," *IEEE Trans. Inform. Theory*, vol. IT-31, no. 6, pp. 794–801, Nov. 1985.

[2] T. Ferguson and J. H. Rabinowitz, "Self-synchronizing huffman codes," *IEEE Trans. Inform. Theory*, vol. IT-30, no. 4, pp. 687–693, July 1984.

[3] W. Lam and A. Reibman, "Self-synchronizing variable-length codes for image transmission," in *Proc. Intl. Conf. Acc. Speech Signal Processing, ICASSP*, vol. Mar., Sept. 1992, pp. 477–480.

[4] Y. Takishima, M. Wada, and H. Murakami, "Reversible variable length codes," *IEEE Trans. Commun.*, vol. 43, no. 2/3/4, pp. 158–162, Feb. 1995.

[5] J. Wen and J. D. Villasenor, "Reversible variable length codes for efficient and robust image and video coding," in *Proc. Data Compression Conf., DCC*, Apr. 1998, pp. 471–480. [Online]. Available: citeseer.nj.nec.com/wen98reversible.html

[6] R. Bauer and J. Hagenauer, "Iterative source-channel decoding using reversible variable length codes," in *Proc. Data Compression Conf., DCC*, Mar. 2000, pp. 93–102.

[7] A. Murad and T. Fuja, "Joint source-channel decoding of variable length encoded sources," in *Proc. Inform. Theory Workshop, ITW*, June 1998, pp. 94–95.

[8] N. Demir and K. Sayood, "Joint source-channel coding for variable length codes," in *Proc. Data Compression Conf., DCC*, Mar. 1998, pp. 139–148.

[9] D. Redmill and N. Kingsbury, "The erec: An error resilient technique for coding variable-length blocks of data," *IEEE Trans. Image Processing*, vol. 5, pp. 565–574, Apr. 1996.

[10] D. Huffman, "A method for the construction of minimum redundancy codes," in *Proc. of the IRE*, vol. 40, 1952, pp. 1098–1101.

[11] T. C. Hu and A. C. Tucker, "Optimal computer search trees and variable length alphabetic codes," *SIAM J. Appl. Math.*, vol. 21, pp. 514–532, 1971.

[12] G. Zhou and Z. Zhang, "Synchronization recovery of variable length codes," *IEEE Trans. Inform. Theory*, vol. 48, no. 1, pp. 219–227, Jan. 2002.

[13] F. Ling, W. Li, and H. Sun, "Bitplane coding of dct coefficients for image and video compression," *Visual Communications and Image Processing*, vol. 3653, pp. 500–508, 1999.

[14] "Advanced video coding, final draft international standard, document jvt-g050r1," ITU-T Rec. H.264 ISO/IEC 11496-10, Tech. Rep., May 2003.

[15] J. Hagenauer, "Rate-compatible punctured convolutional codes (rcpc codes) and their applications," *IEEE Trans. Commun.*, vol. 36, no. Apr., pp. 389–400, Apr. 1988.

[16] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, pp. 284–287, Mar. 1974.

[17] R. Bauer and J. Hagenauer, "Turbo fec/vlc decoding and its application to text compression," in *Proc. Conf. Inform. Theory and Systems*, Mar. 2000, pp. WA6.6–WA6.11.

[18] A. Guyader, E. Fabre, C. Guillemot, and M. Robert, "Joint source-channel turbo decoding of entropy coded sources," *IEEE J. Select. Areas Commun.*, vol. 19, no. 9, pp. 1680–1696, Sept. 2001.

[19] V. B. Balakirsky, "Joint source-channel coding with variable length codes," in *Proc. Intl. Conf. Inform. Theory, ISIT*, 1997, p.419.

[20] K. Zeger and A. Gersho, "Pseudo-gray coding," *IEEE Trans. Commun.*, vol. 38, no. 12, pp. 2147–2158, Dec. 1990.

[21] N. Farvardin, "A study of vector quantization for noisy channels," *IEEE Trans. Inform. Theory*, vol. 36, no. 4, pp. 799–809, July 1990.

[22] Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," *Soviet Physics Doklady*, vol. 10, pp. 707–710, 1966.

[23] [Online]. Available: http://www.irisa.fr/temics/Equipe/Jegou/src/source_code.php

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399