Robust multiplexed codes for compression of heterogeneous data

Hervé Jégou, Christine Guillemot

Robust multiplexed codes for compression of heterogeneous data

Hervé JÉGOU and Christine GUILLEMOT

***Abstract*— Compression systems of real signals (images, video, audio) generate sources of information with different levels of priority which are then encoded with variable length codes (VLC). This paper addresses the issue of robust transmission of such VLC encoded heterogeneous sources over error-prone channels. VLCs are very sensitive to channel noise: when some bits are altered, synchronization losses can occur at the receiver. This paper describes a new family of codes, called multiplexed codes, that confine the de-synchronization phenomenon to low priority data while asymptotically reaching the entropy bound for both (low and high priority) sources. The idea consists in creating fixed length codes for high priority information and in using the inherent redundancy to describe low priority data, hence the name multiplexed codes. Theoretical and simulation results reveal a very high error resilience at almost no cost in compression efficiency.**

***Index Terms*— source coding, variable length codes, data compression, data communication, entropy codes**

I. INTRODUCTION

Entropy coding, producing variable length codewords (VLC), is a core component of any data compression scheme. Unlike fixed length codes (FLCs), variable length codes are designed to exploit the inherent redundancy of a symbol distribution. 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. Soft VLC decoding ideas, exploiting residual source redundancy (the so-called "excess-rate"), have also been shown to reduce the "de-synchronization" effect as well as the residual symbol error rates. These ideas rely on capitalizing on source coder suboptimality, by exploiting inner codeword redundancy and exploiting correlation within the sequence of symbols (inter symbol dependency). Models incorporating both VLC-encoded sources and channel codes (CC) have also been considered [1], [2], [3]. The authors in [1] derive a global stochastic automaton model of the transmitted bitstream by computing the product of the separate models (Markov source (MS), source coder (SC) and channel coder (CC)). The authors in [4] push further the above idea by designing an iterative estimation technique alternating the use of the three models (MS, SC, and CC). The above methods mostly consist in re-augmenting the redundancy of the bitstream, by introducing an error correcting code or dedicated patterns in the chain. Also, the decoding algorithms, often relying on MAP estimators, have a rather high complexity.

The synchronization recovery capability (or self-synchronization property) of a VLC, which represents the error-resiliency of the code, has also been considered as a performance and design criterion of the code in addition to its compression performance. The characterization of the synchronization capability of VLC has thus been extensively studied in the research community [5], [6]. This property is often characterized by an error span [7], also called the mean error propagation length (MEPL) [8]. The authors in [7] have developed a state model for synchronization recovery which is suitable for analyzing the performance of various codes with respect to error recovery. Effort has then also naturally been dedicated to the design of self-synchronizing codes. In [9], the authors show that some Huffman codes, tailored to a given source, may contain a codeword that can serve as a "re-synchronization" point. However, the existence of this codeword depends on the source statistics. Statistically synchronizable codes containing a synchronization sequence are also described in [5] and [10]. Self-synchronizing Huffman codes are also proposed in [11]. The design is governed by a trade-off between redundancy and synchronization recovery capability. E.g., the authors in [12] and [8] search to construct codes with minimum redundancy which allow to have short mean error propagation length (MEPL). In the same spirit, reversible VLCs (RVLC) [13], [14], [3] have been designed specifically to fight against desynchronizations. Variable-to-fixed length codes [15] [16] mainly designed for compression purposes are also well-suited for robust transmission. Both self-synchronizing codes and variable-to-fixed codes only allow for synchronization in terms Levenshtein distance sense [17]. However, strict-sense synchronization [8], i.e. synchronization in the Hamming distance sense, is required by applications, such as image and video compression.

Here, we consider the design of a new family of codes, called "multiplexed codes", that allow to control (even avoid) the "de-synchronization" phenomenon of VLC encoded sources, while still allowing to reach asymptotically the entropy bound and to rely on simple decoding techniques. The principle underlying these codes builds upon the idea that compression systems of real signals generate sources of information with different levels of priority (e.g. texture and motion information for a video signal). This idea underlies unequal error protection (UEP) techniques, which allocate different levels of protection to the different types of information, either to signal frequency bands [18], to bit planes [19], or to quantization layers [20]. In the wake of this idea, we consider two sources, a high priority source and a low priority source referred to as $\mathbf{S_H}$ and $\mathbf{S_L}$ in the sequel. The codes designed are such that the risk of "de-synchronization" is confined to the low priority information. The idea consists in creating a FLC for the $\mathbf{S_H}$ source, and in exploiting the inherent redundancy

to represent or store information of the low priority source $\mathbf{S_L}$. Hence, the source $\mathbf{S_H}$ inherits some of the properties of FLCs such as random access in the data stream and strict-sense synchronization. It is shown that these codes allow to almost reach the entropy bound.

The rest of the paper is organized as follows. Section II introduces the notations we use in the sequel. Section III outlines the principle of multiplexed codes, discusses their compression properties and describes the first coding algorithm in the case of two sources. The approach relies on a mapping of the low priority flow of data into a sequence of k-valued variables, where k depends on the realization of the high priority sequence of symbols. A first mapping, optimal in the sense that it allows to reach the entropy bound is presented. This mapping is based on an Euclidean decomposition of a long integer, which can be computed with a hierarchical (dyadic) approach with a close to linear computational complexity. The computational cost can be further reduced (but this time at the expense of a small excess-rate), by performing the decomposition on a constrained set of k-valued variables. This amounts to consider a constrained partition of the set of fixed length codewords into *equivalence classes* expressed in terms of prime-valued variables. The resulting loss in compression depends on the distance between the respective probability density functions (pdf) of the constrained set of k-valued variable and of the source $\mathbf{S_H}$. A heuristic method for selecting the set of k-valued variables (i.e., the partition) that would best approximate the pdf of the source $\mathbf{S_H}$ is described in section VI. A second class of multiplexed codes for which the cardinalities of the different equivalence classes are constrained to be integer powers of two is also presented. Such codes can be built in a very straightforward manner from any VLC code. The compression performance is then the same as the one achieved by the VLC considered. The impact of channel noise, considering a binary symmetric channel, on the overall source distortion is analyzed in section VII. The choice of the parameters of the algorithm are discussed in section VIII and simulation results are provided in section IX.

II. Problem statement and Notations

Let $\mathbf{S_H} = (S_1, \dots S_t, \dots S_{\mathbf{K_H}})$ be a sequence of source symbols of high priority taking their values in a finite alphabet $\mathcal{A}$ composed of $|\mathcal{A}|$ symbols, $\mathcal{A} = \{a_1, \dots a_i, \dots\}$. Note that, in the following, we reserve capital letters to random variables, and small letters to values of these variables. Bold face characters will be used to denote vectors or sequences. The stationary probability function density of the source $\mathbf{S_H}$ is denoted $\boldsymbol{\mu} = (\mu_1, \dots \mu_i, \dots \mu_{|\mathcal{A}|})$, where μ_i stands for the probability that a symbol of $\mathbf{S_H}$ equals a_i. Let h be the stationary entropy of the source per symbol, given by $h = -\sum_{i=1}^{|\mathcal{A}|} \mu_i log_2(\mu_i)$. Let $\mathbf{S_L}$ be a sequence of source symbols of lower priority taking their values in a finite alphabet. We assume that the realization $\mathbf{s_L}$ of this source has been pre-encoded into a bitstream $\mathbf{b} = (b_1, \dots b_r, \dots b_{K_B})$ with a VLC coder (e.g. Huffman or arithmetic coder). The problem addressed here is the design of a family of joint codes for the two sources $\mathbf{S_H}$ and $\mathbf{S_L}$ that would guarantee the strict-sense synchronization of the high priority source $\mathbf{S_H}$ at almost no cost in terms of compression efficiency.

class $\mathcal{C}_i$	codeword x	symbol a_i	$\lvert\mathcal{C}_i\rvert$	probability μ_i	index q_i
$\mathcal{C}_1$	000	a_1	3	0.40	0
	001				1
	010				2
$\mathcal{C}_2$	011	a_2	2	0.20	0
	100				1
$\mathcal{C}_3$	101	a_3	1	0.20	0
$\mathcal{C}_4$	110	a_4	1	0.10	0
$\mathcal{C}_5$	111	a_5	1	0.10	0

TABLE I
AN EXAMPLE OF MULTIPLEXED CODES ($c = 3$).

III. Multiplexed Codes

A. Principle and definitions

Let us denote $\mathcal{X}$ the set of binary codewords of length c. This set of $|\mathcal{X}| = 2^c$ codewords is partitioned into $|\mathcal{A}|$ subsets, denoted $\mathcal{C}_i$ for $i = 1 \dots |\mathcal{A}|$, called *equivalence classes* associated to symbols a_i of the alphabet $\mathcal{A}$, as shown in Table I. Note that codebook partitioning is sometimes referred to as binning. The condition $c \geq log_2(|\mathcal{A}|)$ is required to have at least 1 codeword per subset. Each *equivalence class* $\mathcal{C}_i$ is a set of $|\mathcal{C}_i|$ codewords. In the following, the integer $|\mathcal{C}_i|$ denotes the cardinality of the *equivalence class* and is such that $\sum_{i=1}^{|\mathcal{A}|} |\mathcal{C}_i| = |\mathcal{X}|$.

A symbol $S_t = a_i$ of the flow $\mathbf{S_H}$ can be encoded with any c-bit codeword x belonging to the *equivalence class* $\mathcal{C}_i$ (see example of Table I). Hence, each codeword S_t can be mapped into a pair $(\mathcal{C}_i, Q_i)$ of two variables denoting respectively the *equivalence class* and the index of the codeword in the *equivalence class* $\mathcal{C}_i$. The variable Q_i is an $|\mathcal{C}_i|$-valued variable, taking its value between 0 and $|\mathcal{C}_i| - 1$ (see Table I) and representing the inherent redundancy of the c-bits FLC. This redundancy will be exploited to describe the lower priority flow of data. Let us denote n_t the cardinality $|\mathcal{C}_i|$ of the equivalence class associated to the realization $S_t = a_i$. Then, to the realization $\mathbf{s_H} = s_1 \dots s_t \dots s_{K_H}$ of the sequence of high priority symbols one can associate a sequence $q_1 \dots q_t \dots q_{K_H}$ of n_t-valued variables that will be used to describe jointly the high and low priority data flows (i.e., $\mathbf{s_H}$ and $\mathbf{s_L}$).

Definition 1: a multiplexed code is defined as a set of pairs (a_i, q_i), where a_i is a symbol value belonging to the alphabet $\mathcal{A}$ (on which the high priority source is quantized) and where q_i is the value of a variable Q_i denoting the index of the codeword in the *equivalence class*.

The corresponding encoder is the function which maps every couple (a_i, q_i) into an c-bits fixed length codeword x as

$$(a_i, q_i) \leftrightarrows x. \tag{1}$$

Example 1: let $\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5\}$ be the alphabet of the source $\mathbf{S_H}$ with the stationary probabilities given by $\mu_1 = 0.4$, $\mu_2 = 0.2$, $\mu_3 = 0.2$, $\mu_4 = 0.1$ and $\mu_5 = 0.1$. Note that this

source has been considered in [7] and [8]. Table I gives an example of partitioning of the set $\mathcal{X}$ into the 5 *equivalence classes* associated to the alphabet symbols. This partitioning reveals 5 $|\mathcal{C}_i|$-valued variables.

Note that, in Table I, the codes are ranked in the lexicographic order. Using this order, each equivalence class can be fully defined by, 1) the first codeword within this equivalence class, 2) the cardinality of this equivalence class. Thus, the lexicographic order permits a compact representation of the multiplexed code table. However, note that the method is not restricted to this order.

B. Conversion of the lower priority bitstream

It appears from above that the design of multiplexed codes relies on the choice of the partition $\mathcal{C} = \{\mathcal{C}_1, \dots \mathcal{C}_{|\mathcal{A}|}\}$. The encoding then proceeds with the conversion of the lower priority bitstream into a flow of variables taking their values in the different sets of $|\mathcal{C}_i|$-valued variables. In order to be multiplexed with symbols of $\mathbf{S_H}$, the lower priority bitstream $\mathbf{b}$ must be mapped into a sequence of $|\mathcal{C}_i|$-valued variables. Let us first consider the mapping of the realization $\mathbf{s_H} = s_1 \dots s_t \dots s_{K_H}$ into the sequence $n_1 \dots n_t \dots n_{K_H}$. The sequence of n_t-valued variables can also be seen as a unique Λ-valued variable γ, where $\Lambda = \prod_{t=1}^{K_H} n_t$. The variable γ hence verifies $0 \leq \gamma < \Lambda$. The quantity Λ denotes the number of different multiplexed sequences of codewords that can be used as a coded representation of the sequence $\mathbf{s_H}$. One of these sequences can be used as a multiplexed coded description of the two sequences $\mathbf{s_H}$ and $\mathbf{s_L}$ [1]. The sequence of multiplexed codewords to be transmitted will then depend on the bitstream representation $\mathbf{b}$ of the source $\mathbf{S_L}$.

The maximum amount of information that can be stored in the Λ-valued variable γ is theoretically $log_2(\Lambda)$ bits. However, since this variable is used to store bits of the bitstream $\mathbf{b}$, only $K'_B = \lfloor log_2(\Lambda) \rfloor$ bits can be stored, leading to an overhead equal to $\frac{log_2(\Lambda)}{K'_B}$. The last[2] K'_B bits of $\mathbf{b}$ are then seen as the binary representation of the integer γ comprised between 0 and $2^{K'_B} - 1$, that can be expressed as

$$\gamma = \sum_{r=1}^{K'_B} b_{(r+K_B-K'_B)}\ 2^{r-1} \tag{2}$$

Note again that the variable γ is an index of the set of sequences that represent $\mathbf{s_H}$. The variable γ must then be expanded into a sequence of pairs (n_t, q_t) that will provide entries in the multiplexed codes table. Let us recall that q_t denotes the index of a codeword in the *equivalence class* associated to the realization s_t of a given symbol S_t. There are different ways to expand the variable γ into the sequence of n_t-valued variables (q_t), where $t = 1, \dots K_H$. One possible method is to use the recursive Euclidean decomposition:

Definition 2: the recursive Euclidean decomposition of a positive integer x by a sequence $\mathbf{y} = (y_1, y_2, \dots, y_K)$ of positive integers is the unique sequence $\mathbf{r} = (r_1, r_2, \dots, r_K) \in \mathbb{N}^{\mathbb{K}}$ such that:

$$\begin{cases} x = r_1 + y_1(\dots(r_t + y_t(\dots(r_{K-1} + y_{K-1}\, r_K)\dots))\dots)) \\ \forall t \in [1..K], 0 \leq r_t \leq y_t - 1 \end{cases} \tag{3}$$

It leads to expand γ as $q_1 + n_1(q_2 + n_2(\dots(q_{K_H-1} + n_{K_H-1}\, q_{K_H})\dots))$. Since each codeword index q_t verifies

$$q_t = \left\lfloor \frac{\gamma}{\prod_{t'=1}^{t-1} n_{t'}} \right\rfloor mod\ n_t, \tag{4}$$

the components q_t can be calculated by Algorithm 1. In section III-C, we will see that these quantities can be computed with a lower complexity. In summary, the encoding proceeds as follows:

1) For $t = 1, \dots K_H$, let n_t be the *cardinality* of the *equivalence class* associated to the realization s_t of the symbol S_t.
2) The integer Λ is computed as $\Lambda = \prod_{t=1}^{K_H} n_t$. The number K'_B of bits of the lower priority bitstream $\mathbf{b}$ that can be jointly encoded with the sequence $\mathbf{S_H}$ is given by $\lfloor log_2(\Lambda) \rfloor$.
3) The K'_B last bits of the low priority bitstream $\mathbf{b}$ are converted into the sequence of pairs (n_t, q_t), $t = 1 \dots K_H$, using Algorithm 1 of the Euclidean decomposition of (3).
4) The sequence of pairs (s_t, q_t) provides the entries in the table of multiplexed codes, hence allows to select the sequence of c-bits fixed length codewords to be transmitted on the channel.
5) If $K'_B \leq K_B$, the $K_B - K'_B$ first bits of the bitstream $\mathbf{b}$ are then concatenated to the sequence of multiplexed codewords. Otherwise, the remaining symbols of $\mathbf{S_H}$ are not multiplexed but sent using other codes on the channel, such as FLCs or Huffman codes.

As all steps are reversible, the decoding proceeds in the reverse order. A schematic diagram of the encoding process, for the particular example proposed hereafter, is depicted in Fig. 1.

Algorithm 1 Conversion of the integer γ into the sequence $\mathbf{q}$

for $t = 1$ to K_H **do**
  $q_t = \gamma'\ modulo\ n_t$
  $\gamma' = \frac{\gamma' - q_t}{n_t}$
end for

t	s_t	n_t	γ	q_t	codeword
1	a_1	3	26	2	001
2	a_4	1	8	0	000
3	a_5	1	8	0	111
4	a_2	2	8	0	011
5	a_3	1	4	0	110
6	a_3	1	4	0	111
7	a_1	3	4	1	000
8	a_2	2	1	1	001

TABLE II

EUCLIDEAN DECOMPOSITION OF THE VARIABLE γ AND CORRESPONDING MULTIPLEXED CODEWORDS.

[1] In fact, only a part of $\mathbf{s_L}$, whose length depends on the multiplexing capacity of the sequence $\mathbf{s_H}$

[2] Any other subset of K'_B bits of $\mathbf{b}$ can be used

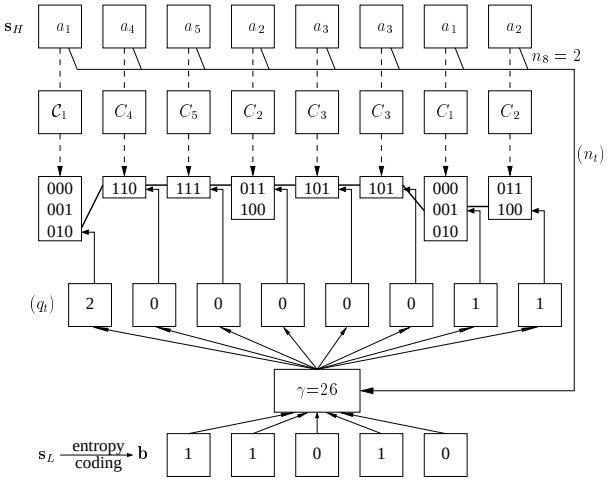

Fig. 1. Link between the source $\mathbf{S_H}$, the low priority source formatted as a state flow $\mathbf{q}$ of $|\mathcal{C}_i|$-valued variables, and multiplexed codewords. Here, the bolded path denotes the chosen codewords.

Example 2: let us consider the source alphabet described in section III and the multiplexed code given in Table I. Considering the sequence of symbols $\mathbf{S_H}$ given in Table II, the algorithm proceeds first with the derivation of the sequence of n_t variables as shown in Table II. This leads to $\Lambda = log_2(36) \approx 2^{5.17}$. Therefore, the last 5 bits of the bitstream $\mathbf{b} = 11010$ are used to process the variable $\gamma = \sum_{r=1}^{5} b_r\, 2^{r-1} = 26$. The Euclidean decomposition of the variable γ according to (1) leads to the sequence of q_t variables (indexes of the codewords in the classes of equivalence associated to the sequence of symbols) given in Table II. The sequence of pairs (s_t, q_t) provides directly the entries in the table of c-bits fixed length codewords, i.e. in the table of multiplexed codes.

C. Hierarchical decomposition of γ

The complexity order of the Euclidean decomposition of the global variable γ is $O(K_H^2)$ if Algorithm 1 is used. This overall complexity can be reduced by considering the following hierarchical decomposition of the variable γ. Note that this reduction in terms of complexity does not induce any overhead or redundancy. The resulting states q_t are identical to the ones obtained by Algorithm 1. For sake of clarity, let us assume that $K_H = 2^l$, with $l \in \mathbb{N}$. The approach can be easily extended to any length. Let $n_{t_1}^{t_2}$ denote the product $\prod_{t=t_1}^{t_2} n_t$. Similarly, $q_{t_1}^{t_2}$ is defined as

$$q_{t_1}^{t_2} \triangleq q_{t_1} + n_{t_1}(q_{t_1+1} + \ldots (q_{t_2-1} + n_{t_2-1} q_{t_2}) \ldots). \quad (5)$$

If the number of terms in (5) is a power of two, i.e if $t_2 - t_1 + 1 = 2^{l'}$, then the entities $n_{t_1}^{t_2}$ and $q_{t_1}^{t_2}$ are respectively decomposed as

$$n_{t_1}^{t_2} = n_{t_1}^{\frac{t_2+t_1-1}{2}}\, n_{\frac{t_2+t_1+1}{2}}^{t_2}, \quad (6)$$

$$q_{t_1}^{t_2} = q_{t_1}^{\frac{t_2+t_1-1}{2}} + n_{t_1}^{\frac{t_2+t_1-1}{2}}\, q_{\frac{t_2+t_1+1}{2}}^{t_2}. \quad (7)$$

Algorithm 2 Conversion of the integer γ into the sequence of states $\mathbf{q}$ using the hierarchical algorithm (for sequences such that $\exists\, l / K_H = 2^l$).

for $j = 1$ to l **do** {Processing of values $n_{t_1}^{t_2}$ involved in further processing}
  for $i = 1$ to 2^{l-j} **do**
    $t_1 = (i-1)\, 2^j + 1$
    $t_2 = i\, 2^j$
    $t_m = (t_1 + t_2 - 1)/2$
    $n_{t_1}^{t_2} = n_{t_1}^{t_m}\, n_{t_m+1}^{t_2}$
  end for
end for

for $j = l$ to 1 by -1 **do** {Processing of values q_{t1}^{t2}}
  for $i = 1$ to 2^{l-j} **do**
    $t_1 = (i-1)\, 2^j + 1$
    $t_2 = i\, 2^j$
    $t_m = (t_1 + t_2 - 1)/2$
    $q_{t_1}^{t_m} = q_{t_1}^{t_2}\ modulo\ n_{t_1}^{t_m}$
    $q_{t_{m+1}}^{t_2} = (q_{t_1}^{t_2} - q_{t_1}^{t_m})/n_{t_1}^{t_m}$
  end for
end for

Notice that $\gamma = q_1^{2^l}$. Hence the state γ is decomposed as

$$\gamma = q_1^{2^{l-1}} + n_1^{2^{l-1}}\, q_{2^{l-1}+1}^{2^l}. \quad (8)$$

Similarly, each term of (8) can be recursively decomposed using (6) and (7). Thus, the conversion of the integer γ into a sequence of states $\mathbf{q}$ can be done iteratively, as described in Algorithm 2. Note that each level j generates the variables required for level $j+1$. The reverse algorithm follows naturally. It is shown in Appendix II that the complexity order of this algorithm is in $O((K_H)^r)$, where $r > 1$ is the complexity exponent of the algorithms used for operations of large integers. Usually, the algorithm used is such that r goes from $r = \frac{log3}{log2} \approx 1.58$ to $r = \frac{log9}{log5} \approx 1.37$, depending on the length of the message [21] [22].

Example 3: let us consider the decomposition proposed in Example 2. This decomposition can also be processed with the hierarchical algorithm as $n_1^2 = 3$, $n_3^4 = 2$, $n_5^6 = 1$, $n_7^8 = 6$, $n_1^4 = 6$, $n_5^8 = 6$ and $q_1^4 = 26$ mod $n_1^4 = 2$, $q_5^8 = \lfloor \frac{26}{n_1^4} \rfloor = 4$, $q_1^2 = 2$, $q_3^4 = 0$, $q_5^6 = 0$, $q_7^8 = 8$, $q_1 = 1$, $q_2 = 0$, $q_3^4 = 0$, $q_3 = 0$, $q_4 = 0$, $q_5 = 0$, $q_6 = 0$, $q_7 = 1$ and finally $q_8 = 1$.

D. Compression efficiency

Each symbol of the sequence $\mathbf{S_H}$ is associated to a c-bits fixed length codeword, leading to a total length of the multiplexed flow equal to $c \times K_H$. Therefore, the compression rate is determined by the amount of $\mathbf{S_L}$ information that can be jointly coded by the sequence of multiplexed codewords. Let us consider again the realization s_t of a symbol S_t of the sequence $\mathbf{S_H}$. One can associate an n_t-valued variable to the symbol value s_t. The amount of data that can be represented by this variable is $log_2(n_t)$ bits. Since the c bits of the multiplexed codeword code both the symbol value s_t and the index q_t describing the low priority bitstream realization,

the description length for the symbol s_t is theoretically $c - log_2(n_t) = -log_2(\frac{n_t}{|\mathcal{X}|})$ bits. The amount of data, in bits, used to code the sequence $\mathbf{s_H}$ is given by

$$-\sum_{1\leq t\leq K_H} log_2(\frac{n_t}{|\mathcal{X}|}). \quad (9)$$

In order to minimize the mean description length (mdl) $\hat{h}$ of the source $\mathbf{S_H}$, one has then to choose the partitioning $\mathcal{C}$ of the set of $|\mathcal{X}|$ fixed length codewords into *equivalence classes* $\mathcal{C}_i$ of cardinality $|\mathcal{C}_i|,\ i = 1 \dots |\mathcal{A}|$, such that

$$(|\mathcal{C}_1|, \dots |\mathcal{C}_i|, \dots |\mathcal{C}_{|\mathcal{A}|}|) = \text{argmin}\left(-\sum_{i=1}^{|\mathcal{A}|} \mu_i\ log_2(\frac{|\mathcal{C}_i|}{|\mathcal{X}|})\right) = \text{argmin}(\hat{h}). \quad (10)$$

In order to have a rate close to the entropy bound of the source $\mathbf{S_H}$, the quantity $\frac{|\mathcal{C}_i|}{|\mathcal{X}|}$ should be as close as possible to μ_i. We come back in section VI on how to calculate an efficient set of $|\mathcal{C}_i|$ values for a given source pdf.

Example 4: the entropy of the source $\mathbf{S_H}$ introduced in section II is $h = 2.122$. The partition $\mathcal{C}$ given in Table I leads to an mdl of $\mathbf{S_H}$ of $\hat{h} = 2.166$. Note that a Huffman encoder of the source would have led to an mdl equal to 2.2. The choice of c as well as of the partition $\mathcal{C}$ has an impact on the coding rate. For example, for $c = 5$, the partition into classes of respective cardinalities $(|\mathcal{C}_1|, |\mathcal{C}_2|, |\mathcal{C}_3|, |\mathcal{C}_4|, |\mathcal{C}_5|) = (13, 7, 6, 3, 3)$ produces an mdl equal to 2.124.

Property 1: $\forall\, \epsilon > 0, \exists\, c \in \mathbb{N}$ such that $\hat{h} - h \leq \epsilon$

In other words, the mdl of $\mathbf{S_H}$ can be made as close as required to the entropy, by increasing the length c of the codewords. The proof is provided in Appendix I.

IV. Multiplexed codes based on a constrained partition $\mathcal{C}$

The computational cost of the mapping (or multiplexing) algorithm can be further reduced by considering a constrained partition of the set of fixed length codewords. The pre-formatted lower priority bitstream is in this case not seen as the representation of a unique variable γ, but as a sequence of "elementary variables". An elementary variable is defined as a variable which takes its value in a set of small dimension: it is a k-valued variable, k being small.

A. Constrained choice of partition $\mathcal{C}$

The $|\mathcal{C}_i|$-valued variables, $1 \leq i \leq |\mathcal{A}|$, are decomposed into a set of elementary variables as follows:

$$|\mathcal{C}_i| = \prod_{j=1}^{\nu_i} f_j^{\alpha_{i,j}}, \quad (11)$$

where f_j are prime factors. The term f_{ν_i} denotes the highest prime factor in the decomposition of $|\mathcal{C}_i|$. The term $\alpha_{i,j}$ stands for the power of the prime factor f_j in the decomposition of $|\mathcal{C}_i|$. This expansion of $|\mathcal{C}_i|$-valued variables can also be represented as:

$$|\mathcal{C}_i|\text{-valued variable} \rightleftarrows \begin{cases} \alpha_{i,1} \text{ binary variables} \\ \alpha_{i,2} \text{ 3-valued variables} \\ \vdots \\ \alpha_{i,\nu_i}\ f_{\nu_i}\text{-valued variables} \end{cases} \quad (12)$$

In order to limit the encoding complexity, the partitioning of the set of c-bits fixed length codewords must be such that the set $(|\mathcal{C}_i|)_{1\leq i\leq|\mathcal{A}|}$ will not result in prime decompositions into factors greater than a given value f_ν. These additional constraints on the partition $\mathcal{C}$ will induce an approximation of the pdf of the $\mathbf{S_H}$ source. This approximation leads to a slightly higher mdl. We come back on this point in Section VIII-B.

bit 1	bit 2	bit 3	3-valued variable 1	3-valued variable 2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	2
0	1	1	1	0
1	0	0	1	1
1	0	1	1	2
1	1	0	2	0
1	1	1	2	1
			2	2

TABLE III
EXAMPLE OF TRANSFORMATION $\mathcal{T}$ ($u_\mathcal{T} = 3$).

B. Conversion of the low priority bitstream into f_j-valued variables

Consecutive segments of the low priority bitstream are seen as the binary representations of integers to be decomposed this time into a set of binary, 3-valued, ..., j-valued, ..., f_ν-valued variables. Let $\mathcal{T}$ be a transformation that takes $u_\mathcal{T}$ bits and produces a set of f_j-valued variables. Let $v_{\mathcal{T},j}$, $1 \leq j \leq \nu$, be the number of f_j-valued variables produced by a transformation $\mathcal{T}$. The number of possible values that can be taken by the set of f_j-valued variables resulting from the transformation $\mathcal{T}$ applied on $u_\mathcal{T}$ bits of the bitstream $\mathbf{b}$ is given by

$$\Lambda_\mathcal{T} = \prod_{j=1}^{\nu} f_j^{v_{\mathcal{T},j}}. \quad (13)$$

This $\Lambda_\mathcal{T}$-valued variable allows to theoretically store $log_2(\Lambda_\mathcal{T})$ bits. Note that $\Lambda_\mathcal{T}$ may be higher than $2^{u_\mathcal{T}}$. Therefore, there is an overhead per bit given by

$$o_\mathcal{T} = \frac{log_2(\Lambda_\mathcal{T})}{u_\mathcal{T}} - 1. \quad (14)$$

For example, the transformation $\mathcal{T}_{15}$ shown in table III applied on 3 bits produces two 3-valued variables, leading to a number of states increased from 8 to 9.

Table IV enumerates the transformations used for $f_\nu = 5$. If $f_\nu = 3$, only the transformations that do not use 5-valued variables are valid: $\mathcal{T}_0, \mathcal{T}_3, \mathcal{T}_{12}, \mathcal{T}_{15}, \mathcal{T}_{17}$. They are sorted by increasing loss in compression efficiency. Among all the possible sets $\mathbf{v}_\mathcal{T} = (v_{\mathcal{T},1}, \dots v_{\mathcal{T},j}, \dots v_{\mathcal{T},\nu})$ explored under

$\mathcal{T}_z$ identifier	$u_{\mathcal{T}_z}$	$v_{\mathcal{T}_z,1}$	$v_{\mathcal{T}_z,2}$	$v_{\mathcal{T}_z,3}$	$o_{\mathcal{T}_z}$
$\mathcal{T}_0$	1	1	0	0	0.0000
$\mathcal{T}_1$	15	0	8	1	0.0001
$\mathcal{T}_2$	21	0	3	7	0.0004
$\mathcal{T}_3$	19	0	12	0	0.0010
$\mathcal{T}_4$	25	0	7	6	0.0011
$\mathcal{T}_5$	24	0	2	9	0.0028
$\mathcal{T}_6$	14	0	3	4	0.0030
$\mathcal{T}_7$	18	0	7	3	0.0034
$\mathcal{T}_8$	27	0	1	11	0.0047
$\mathcal{T}_9$	17	0	2	6	0.0060
$\mathcal{T}_{10}$	30	0	0	13	0.0062
$\mathcal{T}_{11}$	20	0	1	8	0.0080
$\mathcal{T}_{12}$	11	0	7	0	0.0086
$\mathcal{T}_{13}$	23	0	0	10	0.0095
$\mathcal{T}_{14}$	6	0	1	2	0.0381
$\mathcal{T}_{15}$	3	0	2	0	0.0566
$\mathcal{T}_{16}$	2	0	0	1	0.1610
$\mathcal{T}_{z_{max}} = \mathcal{T}_{17}$	1	0	1	0	0.5850

TABLE IV
TRANSFORMATIONS USED FOR $f_\nu = 5$

Algorithm 3 Choice of the transformations to be used

$z = 0$
while $sum(d_j) > 0$ **do** {while some f_j-valued variables are not computed yet}
  $g_{\mathcal{T}_z} = \lfloor min(\frac{d_j}{v_{\mathcal{T}_z,j}}) \rfloor$ {Calculation of how many transformations $\mathcal{T}_z$ can be used}
  for $\forall j$ between 1 and ν **do** {Update the number of f_j-variables to be transformed}
    $d_j = d_j - g_{\mathcal{T}_z} * v_{\mathcal{T}_z,j}$
    $z = z + 1$ {Try next transformation}
  end for
end while

previous constraints, only those for which the transformations introduce an overhead lower than 1% are kept. Increasing f_ν reduces this overhead.

C. Optimal set of transformations

Let again $\mathbf{s_h}$ be the realization of a source $\mathbf{S_H}$ mapped into a sequence of n_t-valued variables. Each n_t-valued variable is decomposed into a set of f_j-valued variables, each one being used $\alpha_{t,j}$ times. Let d_j be the total number of f_j-valued variables involved in the expansion of the entire sequence of n_t-valued variables. Let $g_{\mathcal{T}_z}$ be the number of transformations $\mathcal{T}_z,\ z = 0, \dots z_{max}$ necessary to convert the low priority bitstream into the f_j-valued variables. There are several possible sets $\mathbf{g} = (g_{\mathcal{T}_1}, \dots g_{\mathcal{T}_z}, \dots g_{\mathcal{T}_{z_{max}}})$. The optimum set of transformations is the one which will minimize the total overhead, given by

$$O = \sum_{z=0}^{z_{max}} g_{\mathcal{T}_z}\ u_{\mathcal{T}_z}\ o_{\mathcal{T}_z}. \tag{15}$$

The choice of $\mathbf{g}$ is an optimization problem. Knowing the transformation set $(\mathbf{T}_z)_{0 \le z \le z_{max}}$ and $\mathbf{d} = (d_1, \dots d_j, \dots d_\nu)$, the optimal vector $\mathbf{g}$ is estimated by Algorithm 3.

t	1	2	3	4	5	6	7	8	
s_t	a_1	a_4	a_5	a_2	a_3	a_3	a_1	a_2	
n_t	3	1	1	2	1	1	3	2	
$\alpha_{t,1}$	0	0	0	1	0	0	0	1	$d_1 = 2$
$\alpha_{t,2}$	1	0	0	0	0	0	1	0	$d_2 = 2$

TABLE V
CALCULATION OF THE NUMBER OF AVAILABLE f_j-VALUED VARIABLES.

D. Encoding procedure

Assuming that the low priority source s_L has been pre-encoded using efficient VLCs, the encoding of the two sequences s_H and s_L using fast multiplexed codes proceeds as follows:

1) For each prime integer f_j, $1 \le j \le \nu$, the number d_j of f_j-valued variables resulting from the expansion of the sequence of n_t-valued variables associated to s_H is calculated.
2) The number $g_{\mathcal{T}_z}$ of transformations $\mathcal{T}_z$ needed to convert the low priority bitstream $\mathbf{b}$ into a sequence of f_j-valued variables is searched with the procedure described in Section IV-C. The transformations $\mathcal{T}_z$ are then applied on consecutive segments of $u_{\mathcal{T}_z}$ bits of the bitstream $\mathbf{b}$, leading to the generation of sequences of f_j-valued variables.
3) The n_t-valued variable associated to the symbol realization s_t has to be in bijection with several elementary f_j-valued variables generated in the previous step. Then, at this point, the encoder has to choose the value q_t taken by this n_t-valued variable, such that the Euclidean decomposition of q_t leads to the given sequence of f_j-valued variables. This is done using the Euclidean multiplication.
4) The resulting pair (s_t, q_t) provides entries in the multiplexed codewords table to select the codeword to be transmitted.

The decoding algorithm is made in the reverse order. As a consequence, it is not necessary to receive the whole sequence of codewords to decode the information of high priority $\mathbf{s_H}$: it is directly read in the multiplexed codes tables. Moreover, random access is also possible for the flow $\mathbf{s_H}$.

Example 5: let us consider the sequence of 8 high priority symbols given in Table V, with the corresponding mapping into the sequence of n_t-valued variables according to the multiplexed codes given in Table I. The sequence of n_t-valued variables is decomposed into a sequence of d_1 binary and d_2 3-valued variables with respective powers $\alpha_{t,1}$ and $\alpha_{t,2}$ (cf. table V). In a second step, one has to convert the low priority bitstream into the binary and 3-valued variables. It appears that both transformations $\mathcal{T}_0$ and $\mathcal{T}_{15}$ have to be used (see section IV-C) 2 times and 1 time, respectively. Thus, 5 bits can be multiplexed with the realization $\mathbf{s_H}$ of $\mathbf{S_H}$. The transformation $\mathcal{T}_0$ being identity, the first 3 bits of the bitstream $\mathbf{b}$ are multiplexed unchanged. For each transformation $\mathcal{T}$, the $u_\mathcal{T}$ input bits are seen as the binary representation of an integer

t	1	2	3	4	5	6	7	8
s_t	a_1	a_4	a_5	a_2	a_3	a_3	a_1	a_2
binary variables				1				1
3-valued variables	0						2	
q_t	0	0	0	1			2	1
codeword	000	110	111	100	101	101	010	100

TABLE VI

CONSTRUCTION OF THE SEQUENCE OF MULTIPLEXED CODEWORDS.

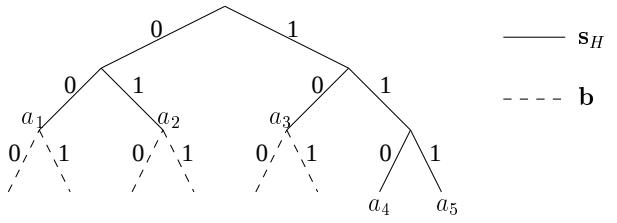

Fig. 2. Codetree describing both $\mathbf{S_H}$ and $\mathbf{B}$

class $\mathcal{C}_i$	codeword x	symbol a_i	$\lvert\mathcal{C}_i\rvert$	probability μ_i	$\overline{U_i}$
$\mathcal{C}_5$	000	a_5	1	0.10	$\varnothing$
$\mathcal{C}_1$	001	a_1	2	0.40	0
	010				1
$\mathcal{C}_2$	011	a_2	2	0.20	0
	100				1
$\mathcal{C}_3$	101	a_3	2	0.20	1
	110				0
$\mathcal{C}_4$	111	a_4	1	0.10	$\varnothing$

TABLE VII

A BINARY MULTIPLEXED CODES ($c = 3$, MDL $= 2.2$).

$\gamma_{\mathcal{T}}$ as

$\mathcal{T}$		Input bits	$\rightarrow$	$\gamma_{\mathcal{T}}$	$\rightarrow$	binary	3-valued
$\mathcal{T}_0$	:	1	$\rightarrow$	1	$\rightarrow$	1	
$\mathcal{T}_0$	:	1	$\rightarrow$	1	$\rightarrow$	1	
$\mathcal{T}_{15}$	:	010	$\rightarrow$	2	$\rightarrow$		02

and leading to the sequences 11 of binary variables and 02 of 3-valued variables. The value q_t taken by the n_t-valued variable associated to the symbol realization s_t and to the realization of the low priority bitstream $\mathbf{b}$ is obtained from the sequences of binary ($j = 1$) and 3-valued ($j = 2$) variables. In the example above, the state q_t is generated using either a bit for equivalence class $\mathcal{C}_2$, either a 3-valued variable for equivalence class $\mathcal{C}_1$. The resulting pair (s_t, q_t) provides entries in the multiplexed codes table (see Table I), leading to the selection of multiplexed codes given in Table VI.

V. BINARY MULTIPLEXED CODES

The partition of the FLC into several equivalence classes can be chosen such that the storage capacity is an integer number of bits. It means that, for each equivalence class $\mathcal{C}_i$, a bijection exists between the interval of indexes $[0, |\mathcal{C}_i|-1]$ and a varying integer number of bits, hence the following definition:

Definition 3: a binary multiplexed code is a multiplexed code such that $\forall i \in [1..|\mathcal{A}|],\ log_2(|\mathcal{C}_i|) \in \mathbb{N}$.

Then, to each symbol a_i of the source $\mathbf{S_H}$, one can associate an equivalence class $\mathcal{C}_i$, hence a set of fixed length codewords, which is in turn indexed by a sequence $\overline{U_i} = U_i^1 \dots U_i^{log_2|\mathcal{C}_i|}$ of $log_2(|\mathcal{C}_i|)$ bits. A codeword representing jointly a symbol of the source $\mathbf{S_H}$ and a segment of the low priority bitstream $\mathbf{b}$ can thus be selected without requiring any conversion of the bitstream $\mathbf{b}$.

Example 6: let again consider the source $\mathbf{s_H} = a_1a_4a_5a_2a_3a_3a_1a_2$ and $\mathbf{b} = 110100$ a pre-encoded bitstream. Let us also consider the binary multiplexed codes of Table VII. The number of bits that can be multiplexed with each symbol realization of the sequence $\mathbf{S_H}$ is processed, which leads to segment $\mathbf{b}$ into $(\overline{u_1}, \dots \overline{u_t}, \dots \overline{u_{K_H}}) = (1, \varnothing, \varnothing, 1, 0, 1, 0, 0)$, where notation $\varnothing$ indicates that the corresponding equivalence class is of cardinality 1 and is consequently not indexed by any bit. Then, for each high priority symbol index t, the couple $(s_t, \overline{u_t})$ indexes a codeword in the multiplexed table. The resulting multiplexed bitstream is 010 111 000 100 110 101 001 011.

Instead of designing a FLC codebook for the high priority source $\mathbf{S_H}$, one can alternatively consider a variable length codetree. This VLC codetree can then be completed to form a binary FLC codetree (see Fig. 2). The symbols of the alphabet of the source $\mathbf{S_H}$ will then be associated either to leaves of the FLC or to intermediate nodes. When they correspond to a node, all the corresponding leaves will constitute the equivalence class associated to the given symbol of $\mathbf{S_H}$. The cardinalities of the equivalence classes are the integer powers of 2. Hence, one can create a multiplexed code such that codewords of the same equivalence class have the same prefix. The suffix $\overline{U_i}$ of the codewords can then be used to "store" the bits of the bitstream $\mathbf{b}$ (see Fig. 2).

Definition 4: a Binary multiplexed code derived from a VLC is a multiplexed code constructed such that (1) every prefix of a codeword is the representation of a symbol in a VLC tree, (2) every suffix corresponds to the multiplexed data of the low priority bitstream.

Example 7: considering the same sequences as in example 6, the VLC-based multiplexed code depicted in Fig. 2 leads to the multiplexed bitstream below.

s_t	a_1	a_4	a_5	a_2	a_3	a_3	a_1	a_2
prefix	00.	110	111	01.	10.	10.	00.	01.
$\overline{u_t}$	..1	...	...	..1	..0	..1	..0	..0
codeword	001	110	111	010	100	101	000	010

The compression efficiency for the source $\mathbf{S_H}$ is obviously given by the efficiency of the VLC considered. Similarly, since the low priority source before multiplexing has been pre-encoded, the corresponding compression efficiency depends on the VLC code considered for this source. The low priority source (e.g high frequencies of a wavelet decomposition) can be pre-encoded with an arithmetic coder.

VI. Probability distribution function approximation

As already mentioned above, the mdl $\hat{h}$ of the source $\mathbf{S_H}$ is function of the sizes $|\mathcal{C}_i|$ of the equivalence classes $\mathcal{C}_i, i = 1 \dots |\mathcal{A}|$. This section describes an algorithm that aims at approximating, under constraints, the pdf of the source $\mathbf{S_H}$. The probability distribution function estimator $\hat{\boldsymbol{\mu}}$ is defined by

$$\hat{\boldsymbol{\mu}} = (\hat{\mu}_1, \dots \hat{\mu}_i, \dots \hat{\mu}_{|\mathcal{A}|}) = \left(\frac{|\mathcal{C}_1|}{|\mathcal{X}|}, \dots \frac{|\mathcal{C}_i|}{|\mathcal{X}|}, \dots \frac{|\mathcal{C}_{|\mathcal{A}|}|}{|\mathcal{X}|} \right), \tag{16}$$

where $\sum_{i=1}^{|\mathcal{A}|} |\mathcal{C}_i| \leq |\mathcal{X}|$. Let $\boldsymbol{\eta} = \{\eta_1 = 1, \dots \eta_k, \dots\}$ be the set of possible values $|\mathcal{C}_i|$. A higher number of valid values for $|\mathcal{C}_i|$ leads indeed to a better approximation of the pdf of the source $\mathbf{S_H}$. The probability μ_i of a symbol cannot be lower than $\frac{1}{|\mathcal{X}|}$. The alphabet $\mathcal{A}$ is partitioned into two subsets $\mathcal{A}_m$ and $\mathcal{A}_M$, defined respectively by

$$a_i \in \mathcal{A}_m \text{ iff } \mu_i < \frac{1}{|\mathcal{X}|} \text{ and } a_i \in \mathcal{A}_M \text{ iff } \mu_i \geq \frac{1}{|\mathcal{X}|}. \tag{17}$$

Let $\tilde{\boldsymbol{\mu}}$ be the probability distribution function of the set of symbols $a_i \in \mathcal{A}_M$. Each is given by

$$\tilde{\mu}_i = \frac{\mu_i}{\sum_{a_i \in \mathcal{A}_\mathcal{M}} \mu_i} \tag{18}$$

The algorithm aiming at the best partition of the set of codewords proceeds as follows:

1) For each $a_i \in \mathcal{A}_m$, $|\mathcal{C}_i|$ is set to 1.
2) The probability density function $\tilde{\boldsymbol{\mu}}$ of symbols belonging to $\mathcal{A}_M$ is calculated. Since $|\mathcal{A}_m|$ codewords have already been affected to equivalence classes in the first step, there is still $|\mathcal{X}| - |\mathcal{A}_m|$ codewords to be assigned to equivalence classes. The expression (10) can then be written as
$(|\mathcal{C}_1|, \dots |\mathcal{C}_i|, \dots |\mathcal{C}_{|\mathcal{A}|}|) =$
$$\text{argmin} \left(log_2(|\mathcal{X}|) \sum_{a_i \in \mathcal{A}_m} \mu_i - \sum_{a_i \in \mathcal{A}_M} \mu_i \ log_2(\frac{|\mathcal{C}_i|}{|\mathcal{X}|}) \right). \tag{19}$$
Since the first part of expression (19) is a constant and $|\mathcal{C}_i| = 1$ when $a_i \in \mathcal{A}_m$, the optimization is performed for symbols of $\mathcal{A}_M$ only, and is expressed as
$$\begin{aligned}
(|\mathcal{C}_i|)_{i \in \mathcal{A}_\mathcal{M}} &= \text{argmin} \left(- \textstyle\sum_{a_i \in \mathcal{A}_M} \mu_i \ log_2(\frac{|\mathcal{C}_i|}{|\mathcal{X}|}) \right) \\
&= \text{argmin} \Big(- \textstyle\sum_{a_i \in \mathcal{A}_M} \frac{\mu_i}{\sum_{i \in \mathcal{A}_\mathcal{M}} \mu_i} \ (log_2(\frac{|\mathcal{C}_i|}{|\mathcal{X}| - |\mathcal{A}_m|}) \\
&\qquad + log_2(\frac{|\mathcal{X}| - |\mathcal{A}_m|}{|\mathcal{X}|})) \Big) \\
&= \text{argmin} \Big(- \textstyle\sum_{a_i \in \mathcal{A}_M} \tilde{\mu}_i \ log_2(\frac{|\mathcal{C}_i|}{|\mathcal{X}| - |\mathcal{A}_m|}) \\
&\qquad - \textstyle\sum_{a_i \in \mathcal{A}_M} \tilde{\mu}_i \ log_2(\frac{|\mathcal{X}| - |\mathcal{A}_m|}{|\mathcal{X}|}) \Big) \\
&= \text{argmin} \left(- \textstyle\sum_{a_i \in \mathcal{A}_M} \tilde{\mu}_i \ (log_2(\frac{|\mathcal{C}_i|}{|\mathcal{X}| - |\mathcal{A}_m|}) \right).
\end{aligned} \tag{20}$$
As $\sum_{a_i \in \mathcal{A}_M} \tilde{\mu}_i = 1$, expression (20) can be seen as the expression of the mdl when the index i is constrained to be such that $a_i \in \mathcal{A}_M$. Consequently, assuming that $\frac{|\mathcal{C}_i|}{|\mathcal{X}| - |\mathcal{A}_m|}$ can take any value of interval $[0, 1]$, the optimum is given by:
$$\forall a_i \in \mathcal{A}_M, \ \frac{|\mathcal{C}_i|}{|\mathcal{X}| - |\mathcal{A}_m|} = \tilde{\mu}_i. \tag{21}$$
In the following, the index i is supposed to be chosen such that $a_i \in \mathcal{A}_M$.
3) For each $a_i \in \mathcal{A}_M$, $|\mathcal{C}_i|$ is set to the highest value in $\boldsymbol{\eta}$ such that $|\mathcal{C}_i| \leq \tilde{\mu}_i (|\mathcal{X}| - |\mathcal{A}_m|)$. However, at this point, some cardinalities may be equal to zero. In that case, the corresponding symbols are assumed to belong to $\mathcal{A}_m$ instead of $\mathcal{A}_M$ and the algorithm goes back to step 1. The remaining number of codewords to be assigned to equivalence classes is then given by $\sigma = |\mathcal{X}| - \sum_{i=1}^{|\mathcal{A}|} |\mathcal{C}_i| \geq 0$.
4) As a consequence, it is possible to increase some values of $|\mathcal{C}_i|$. For each index i between 1 and $|\mathcal{A}|$, if the value η_{k+1} is used instead of η_k, the mdl $\hat{h}$ decreases of a positive value $\mu_i \, log_2(\frac{\eta_{k+1}}{\eta_k})$. The decreasing mdl is induced by the assignment of $\eta_{k+1} - \eta_k$ pairs $(a_i, |\mathcal{C}_i|)$ to classes of equivalence and, in average per codeword, is given by $\gamma_i = \frac{\mu_i \, log_2(\frac{\eta_{k+1}}{\eta_k})}{\eta_{k+1} - \eta_k}$. These pairs are then sorted by decreasing values of γ_i. The way to proceed is to choose the pair $(a_i, |\mathcal{C}_i|)$ with the highest associated value γ_i. For a given variable $|\mathcal{C}_i|$ set in the previous steps, it is possible to re-set it to η_{k+1} instead of η_k only if $\eta_{k+1} - \eta_k \leq \sigma$, i.e. if there is a sufficient number of codewords still to be assigned. If it is not the case, the pair $(a_i, |\mathcal{C}_i|)$ is ignored in the next iteration of the algorithm. The procedure continues with the treatment of the pair with the next value γ_i. If $\eta_{k+1} - \eta_k \leq \sigma$, then $|\mathcal{C}_i| = \eta_{k+1}$, the value σ is set to $\sigma - \eta_{k+1} + \eta_k$, and the variable γ_i is updated for this symbol.

VII. Error handling on a binary symmetric channel

In this section, we analyze the impact of channel errors on the distortion of the source $\mathbf{S_H}$. The part of the flow $\mathbf{b}$ that is not multiplexed is supposed to be lost if an error occurs on the channel. We consider a binary symmetric channel (BSC) with a bit error rate p. The pdf $\boldsymbol{\mu}$ of $\mathbf{S_H}$ being known, it is possible to calculate the reconstruction accuracy in terms of Symbol Error Rate (SER) or of Mean Square Error (MSE). Since each state q_t calculated from $\mathbf{b}$ can be seen as a uniform random variable, a symbol a_i of $\mathcal{A}$ has the same probability to be encoded with any codeword of its equivalence class $\mathcal{C}_i$. Hence, for a given codeword $x \in \mathcal{C}_i$,

$$P(x) = \frac{\mu_i}{|\mathcal{C}_i|}. \tag{22}$$

The probability to receive the symbol y, if x has been emitted, is function of the Hamming distance $H(x, y)$ between the codewords x and y, and is given by $P(y/x) = p^{H(x,y)} \ (1-p)^{c - H(x,y)}$. The mean distortion induced by the channel noise is then given by

$$\mathbb{E}(\Delta_{\mathbf{S_H}}) = \sum_{x \in \mathcal{X}} P(x) \sum_{y \in \mathcal{X}} P(y/x) \ \Delta(x, y), \tag{23}$$

where $\Delta(x, y)$ denotes the distortion of the source $\mathbf{S_H}$ induced by the reception of y instead of x. Let respectively a_i and $a_{i'}$ be the symbols decoded from x and y and $\Delta(a_i, a_{i'})$ the corresponding distortion measure. E.g., $\Delta(a_i, a_{i'})$ may be,

respectively, the Kronecker operator or the quadratic error, depending on the error-resilience measure (respectively the SER or the MSE). This leads to

$$\mathbb{E}(\Delta_{\mathbf{S_H}}) = \sum_{a_i \in \mathcal{A}} \mu_i \sum_{a_{i'} \in \mathcal{A}} \Delta(a_i, a_{i'}) \underbrace{\frac{1}{|\mathcal{C}_i|} \sum_{x \in \mathcal{C}_i} \sum_{y \in \mathcal{C}_{i'}} P(y/x)}_{P(a_{i'}/a_i)}, \tag{24}$$

where $P(a_{i'}/a_i)$ is the probability to decode the symbol $a_{i'}$ if a_i has been transmitted. Therefore, the choice of the partition $\mathcal{C}$ has a major impact on the final distortion. However, the number of partitions for a given set of $|\mathcal{C}_i|$-values is very high, and is given by $\frac{|\mathcal{X}|!}{\prod_{i=1}^{|\mathcal{A}|} |\mathcal{C}_i|!}$. For example, the number of partitions such that $|\mathcal{C}_1| = 3, |\mathcal{C}_2| = 2, |\mathcal{C}_3| = 1, |\mathcal{C}_4| = 1$, $|\mathcal{C}_5| = 1$, is $3\,360$.

Example 8: let us consider again the example of the source alphabet given in section III and the corresponding multiplexed code of codeword length $c = 6$. Let us define the alphabet values as $a_1 = 1$, $a_2 = 2$, $a_2 = 3$, $a_3 = 4$, $a_4 = 5$. We consider the two following distortion measures: the SER (in that case $\Delta(a_i, a_{i'}) = 1$ if $a_i = a_{i'}$, 0 otherwise) and the quadratic error ($\Delta(a_i, a_{i'}) = (a_i - a_{i'})^2$). For a BSC of BER equal to 0.01, the summation in (24) leads to a SER of 0.0279 and an MSE of 0.0814 for a multiplexed code when a lexicographic index assignment is used. If the index assignment is optimized using a simulated annealing algorithm (e.g, [23]), then the distortion is 0.0243 for the SER. However, we did not observe any improvement for the MSE criteria using this method (for this source).

For multiplexed codes derived from a VLC, the analysis of the SER simplifies as follows. One can remark that a given symbol a_i is in error if and only if the prefix that identifies the equivalence class is in error. Hence, the probability that the symbol a_i is in error is given by $p^{c-log_2(|\mathcal{C}_i|)}$. This leads to the following expression of the SER:

$$\text{SER}_{\mathbf{S_H}} = 1 - \sum_{a_i \in \mathcal{A}} \mu_i \, (1-p)^{c-log_2(|\mathcal{C}_i|)}. \tag{25}$$

For low BER, the approximation $(1-p)^n = 1 - n\,p + O(p^2)$ leads to

$$\text{SER}_{\mathbf{S_H}} = p \sum_{a_i \in \mathcal{A}} \mu_i \, (c - log_2(|\mathcal{C}_i)|) + O(p^2), \tag{26}$$

where we recognize the expression of the mdl of the VLC code from which the multiplexed code has been derived: SER $\approx$ $p \times$ mdl. This SER is lower than for FLCs ($p \times mdl \leq p \times c$), but the error-resilience is not uniformly better: symbols with higher probabilities of occurrence have a lower probability to be erroneous whereas symbols with lower probabilities have a higher probability to be erroneous.

Note on the error-resilience of the low priority bitstream: in the general case and from equation (3), it appears that any error occurring on a multiplexed codeword at symbol clock t engenders, at least, an error on the quantity n_t or on the state q_t, or on possibly both. Consequently, the remaining states $q_{t+1} \cdots q_{K_H}$ are corrupted. Thus, the conversion of $\mathbf{S_L}$ into a sequence of state is a step that is sensitive to bit errors. As a first approximation, the expectation of the position of the first impaired bit on the BSC is given by $(c - \text{mdl}_{\mathbf{S_H}})/(c \times p)$ (instead of $1/p$ if $\mathbf{S_L}$ is not multiplexed).

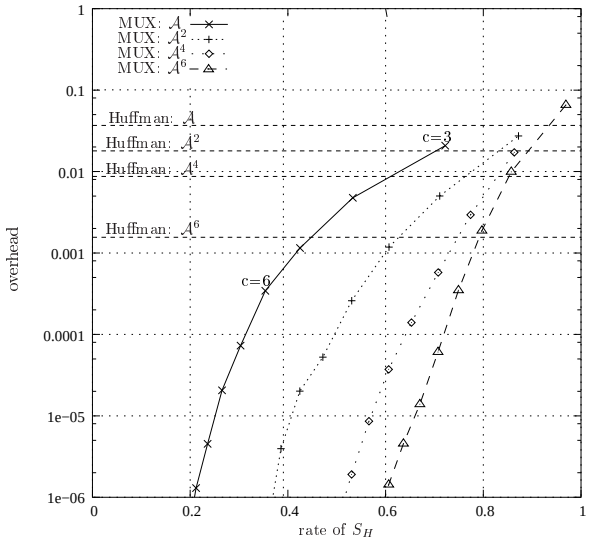

Fig. 3. Compression efficiency *vs* Ratio $\frac{\mathbf{S_H}}{\mathbf{S_H}+\mathbf{S_L}}$. The source is the one given in Example 1. Multiplexed codes designed for the product alphabets $\{a_1, a_2, a_3, a_4, a_5\}^n$ with $n = 1, 2, 4, 6$ have been considered. Compression efficiencies of corresponding Generalized Huffman codes are also depicted.

In binary multiplexed codes, the bits of $\mathbf{S_L}$ are directly indexed by codewords. It follows that either a bit error may lead to drop or to insert some bits, either the number of decoded bits is correct but these bits are erroneous. Consequently, the bitstream is not fully desynchronized in terms of Levenshtein distance. This fact has been validated by simulations results (see section IX).

VIII. Discussion and Parameters choice

A. Redundancy in terms of the ratio $\mathbf{S_H}$ versus $\mathbf{S_H}$+$\mathbf{S_L}$

We have seen above that c must be high enough so that one can find a partition that will closely approximate the pdf of the source $\mathbf{S_H}$. On the other hand, a high value for the parameter c will result in a decreased rate of synchronous data. Thus a compromise has to be found between reaching the entropy (compression efficiency) and the rate of synchronous data in the multiplexed flow (error-resilience). Fig. 3 shows the overhead (or redundancy) in terms of the ratio between the source $\mathbf{S_H}$ mdl and the total amount of transmitted bits ($\mathbf{S_H}$+$\mathbf{S_L}$). It can be seen that, if we consider multiplexed codes for joint encoding of motion and texture information in a video stream, for which realistic ratios are around 15%, the redundancy of the code is negligible (around 10^{-6}).

Fig. 3 shows the performance of multiplexed codes versus Huffman codes when both codes are designed for product alphabets. One can observe in Fig. 3 that when considering a ratio of S_H versus the total number of symbols of 60%, the overhead remains rather low (10^{-3}). To achieve the same overhead with a Huffman code, one has to take the symbols by groups of 6 symbols (alphabet $\mathcal{A}^6$). Fig. 3 also shows that using product alphabets (here $\mathcal{A}^2$, $\mathcal{A}^4$ and $\mathcal{A}^6$) allows

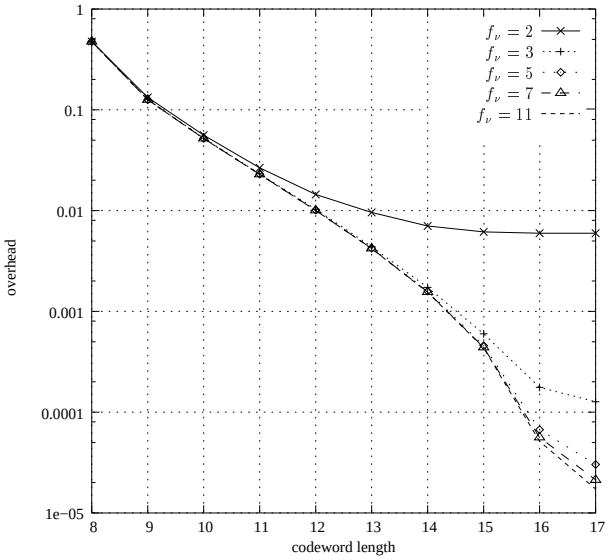

Fig. 4. Closeness to entropy of fast-computable Multiplexed codes (Gaussian source).

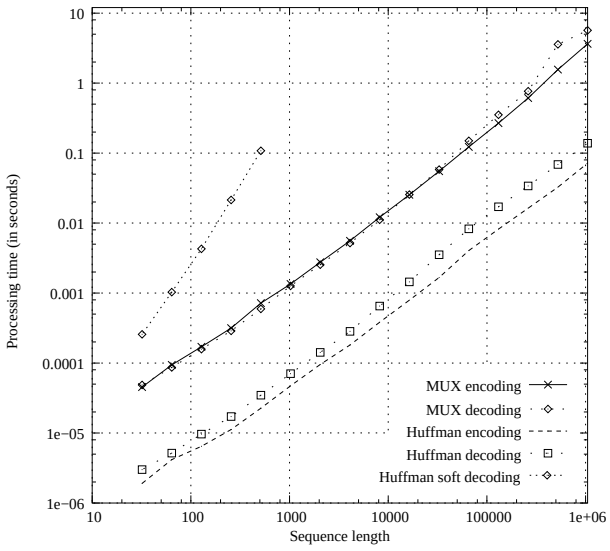

Fig. 5. Encoding and decoding processing durations of Multiplexed codes (c=3) based on Algorithm 2. Source of Example 1 has still been considered. Corresponding durations are provided for encoding and decoding (hard and soft) of Huffman codes.

to find a better trade-off between compression efficiency and the ratio of the high priority source versus the overall number of symbols, however at the cost of larger codetrees or tables. Note that using product alphabets preserves the strict-sense synchronization of the multiplexed codes. However, this hard synchronization is in this case guaranteed every n symbols, where n is the power of the alphabet.

B. Pdf approximation inherent to codes with constrained partitions

Let us consider the codes based on a constrained partition presented in Section IV-A. The ability to closely approximate the pdf of the source $\mathbf{S_H}$ (hence the closeness to entropy) depends also on the parameter ν. The choice of the parameters c and ν has hence a strong impact on the performance, in terms of compression and error-resilience, of the codes. Let h be the entropy per symbol of the source and $\hat{h}$ the mean codeword length produced by the algorithm. The overhead rate is given by $\frac{\hat{h}-h}{h}$. As c bits are always used to represent a symbol of the source $\mathbf{S_H}$, the rate of high priority data in the final multiplexed c-bits codewords stream is given by $\frac{h}{c}$. Fig. 4 shows the influence of the parameters c and ν on the rate $\frac{\hat{h}-h}{h}$. For a source of cardinality $|\mathcal{A}| = 256$, the value $f_\nu = 5$ leads to a small overhead when the codeword length is $c = 14$. Notice that the number of available transformations and the number of flows of f_j-valued variables increases with ν. Therefore, the computational cost of the algorithm increases with ν.

The problem of finding good cardinalities for binary multiplexed codes is equivalent to the problem of finding good VLC binary codetrees. Thus, choosing the codewords length is equivalent to minimizing (10) by assigning the cardinalities $|\mathcal{C}_i|$, under the constraints $log_2(|\mathcal{C}_i|) \in \mathbb{N}$ and $\sum_{a_i \in \mathcal{A}} |\mathcal{C}_i| = 2^c$. It is well known that the algorithm that will provide the best approximation of the source $\mathbf{S_H}$ pdf is the Huffman algorithm [24]. However, in order to meet the requirements in terms of the ratio of synchronous data, one may limit the length of the longest codeword [25], which amounts to limit the parameter c. For this purpose, the algorithm proposed in [26] may be used.

C. Elements of encoding and decoding complexity

The conversion of the low priority source into a sequence of states is the step that has the highest computing cost. For binary multiplexed codes, we have seen in section V that this conversion is straightforward. In that case, the subsequent computing costs required for encoding and decoding are very similar to the ones involved in encoding and decoding VLCs. The encoding and decoding of multiplexed codes based on constrained partitions (see Section IV) has a higher computing cost but the order of complexity remains the same: the number of operations involved in the conversion of the low priority source is linear as the length of the sequence to encode increases. In section III-C, we have shown that the complexity of the general case is sub-quadratic, but above linear complexity. The exact complexity depends on the algorithms used for large integer multiplications and divisions.

Fig. 5 depicts quantitative results of encoding/decoding complexity using an implementation of Algorithm 2 based on the GNU Multiple Precision Arithmetic Library [27]. The computing cost, expressed in seconds, is compared with a typical implementation of Huffman codes. It is shown that the complexity of multiplexed codes, as the sequence length increases, remains tractable for most applications. Multiplexed Codes complexity has also been compared against the complexity obtained with soft decoding of VLCs based on a bit/symbol trellis [3] and on the BCJR algorithm [28]. Fig. 5 shows that this solution has a rather higher complexity than multiplexed codes. This results from the fact that the corre-

sponding trellis has a number of states which is quadratic as the length of the sequence increases. Note that many authors have proposed methods to decrease the complexity of VLCs, using pruning techniques or suboptimal trellis, e.g. [29], but then the estimation is not optimal.

IX. Simulation results

The multiplexed code are all the most beneficial when applied to sources of different levels of priority. They intrinsically support unequal error protection of the corresponding sources. This is going to be illustrated below with a simple image coding system.

However, these codes can also be applied in the case, there is a unique source. They allow to guarantee hard synchronization for at least part of the sequence of symbols to be transmitted. To illustrate the advantage in such a situation, the performance of the multiplexed codes referred to as MUX codes in the sequel and on the figures) is first assessed in terms of SER (Hamming distance) and Levenshtein distance by considering the simple theoretical source given in Example 1 [7]. The performance of the MUX code is compared to the performance achieved with the code $VLC = \{01, 00, 11, 100, 101\}$ referred to as $C5$ in [8]. We have chosen to use this code as a basis for comparisons since, among the 16 optimal codes described in [8], this is the code which leads to the best performance in terms of Levenshtein distance. For the experiments reported for the theoretical source, we have considered sequences of 1000 symbols. The sequence of 1000 symbols is divided into two segments. The first and second segments are assumed to be the sources of high and low priority respectively. The second segment of the sequence of symbols is thus coded with this code referred to as VLC on the figures. The sequence is divided in such a way that the multiplexing capacity of the high priority source is greater or equal to the length of the bitstream representation of the low priority source. The multiplexed code used in this first experiment is the lexicographic binary multiplexed code such that $|\mathcal{C}_1| = 4$, $|\mathcal{C}_2| = 1$, $|\mathcal{C}_3| = 1$, $|\mathcal{C}_4| = 1$ and $|\mathcal{C}_5| = 1$. This code is derived from the VLC code $\{0, 100, 101, 110, 111\}$.

This code leads to a ratio between the high priority symbols to the overall number of symbols equal to 0.72. The mdl measured is close to the theoretical value of 2.2 with both the MUX and the VLC ($C5$) codes. The results have been averaged over 20000 simulations. For both codes, we have respectively measured the overall performances in terms of normalized Levenshtein distance and SER. Fig. 6 and Fig. 7 show the performance in terms of normalized Levenshtein distance and SER obtained on the entire source with both codes (curves VLC:S; MUX:S). These figures also show the normalized Levenshtein distance and SER obtained on the high (MUX:S_H) and low (MUX:S_L) priority symbols separately. It can be observed that the Levenshtein distance values averaged on the entire sequence are very close with both codes. However, Fig. 7 shows the advantage of multiplexed codes if we consider strict-sense synchronization, i.e. the SER or Hamming distance. It can also be observed that significantly lower normalized Levenshtein distance and SER values are

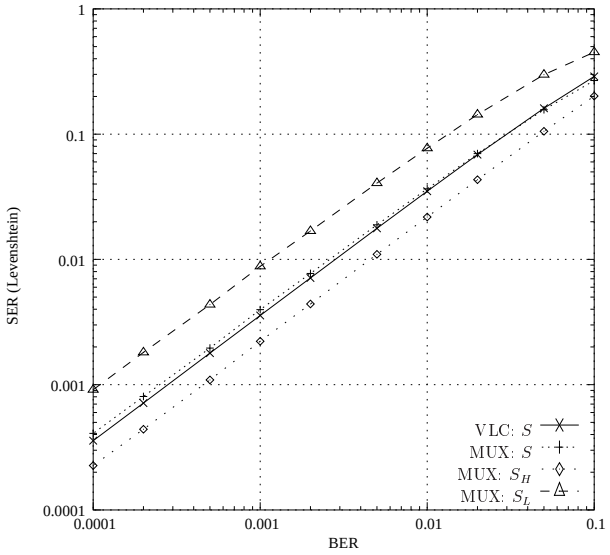

Fig. 6. Performances in terms of normalized Levenshtein distance of multiplexed codes *vs* Huffman codes.

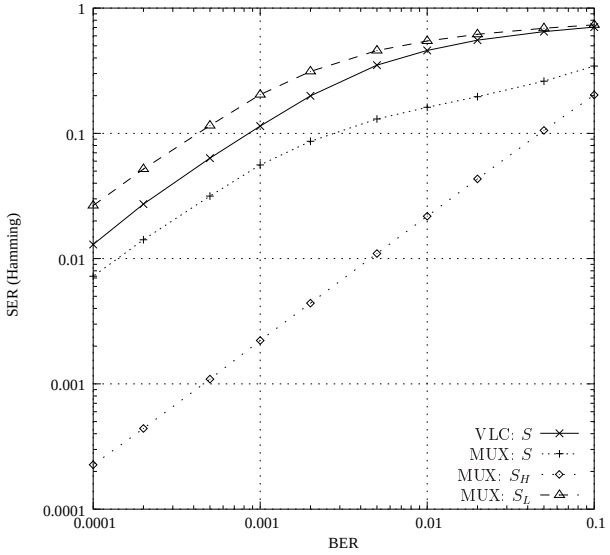

Fig. 7. SER performances of multiplexed codes *vs* Huffman codes.

obtained for the symbols to be considered as high priority symbols. One can also verify on the curves that the SER approximation given in section VII ($\mathrm{SER}_{\mathbf{S_H}} \approx 2.2 \times \mathrm{BER}$) is valid. The benefit of this inherent unequal error protection feature is better illustrated below with a concrete image coding example.

The multiplexed codes have then been experimented with real sources to show the benefits of both the inherent unequal error protection feature and of the hard synchronization property guaranteed for the high priority symbols. We have considered a simple image coding system where the image is first decomposed into 16 subbands using a two-stage wavelet transform. The low and high frequency subbands have been quantized respectively on 7 and 4 bits. The high frequencies have been encoded with an Huffman algorithm, and

then multiplexed into the low frequencies using lexicographic multiplexed codes of parameters $c = 16$ and $f_\nu = 5$. A synchronization marker has been used every four lines for both Huffman codes and the low priority part of the multiplexed codes. The bit rates obtained with Huffman and multiplexed codes are quite similar and given respectively by 1.713 bit per pixel (*bpp*) and 1.712 *bpp*. When considering FLCs, the average bit rate obtained is 6.187 *bpp*. These FLCs are based on the lexicographic index assignment. Fig. 8 depicts the PSNR and visual qualities obtained with the three codes. This figure evidences significant improvements both in PSNR and visual quality when using multiplexed codes even in comparison with FLCs and for a similar compression factor as the one obtained with Huffman codes.

X. Conclusion

This paper describes a new family of codes called "multiplexed codes". These codes avoid the "de-synchronization" phenomenon for symbols considered to be of high priority, and allow thus to confine error propagation to symbols assumed to be of lower priority. The low priority symbols can be pre-encoded with any efficient VLC. A FLC is created for the high priority source, its inherent redundancy being exploited to represent information from the low priority stream. These codes are shown to reach asymptotically the entropy bound for both (low and high priority) sources. Several methods of multiplexing of the two sources are described. Sub-classes of multiplexed codes relying on constrained partitions of the FLC codebook are also introduced. The codes called binary multiplexed codes can be constructed from classical VLC. They thus inherit the compression properties of the VLC considered and allow for multiplexing with very low complexity. Theoretical and simulation results, using both theoretical and real sources, show that these codes are error-resilient at almost no cost in terms of compression efficiency. Another key advantage is that they allow to make use of simple decoding techniques. They hence appear to be excellent alternatives to reversible variable length codes (which suffer from some penalty in terms of compression efficiency, while not avoiding completely error propagation despite a decoding in two passes) or to classical VLCs for which robust decoding makes often use of computationally expensive Bayesian estimation techniques.

XI. Acknowledgment

The authors would like to thank the anonymous reviewers for their very constructive and helpful comments.

References

[1] 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.

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

[3] 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.

[4] 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.

[5] V. Wei and R. Scholtz, "On the characterization of statistically synchronizable codes," *IEEE Trans. Inform. Theory*, vol. 26, pp. 733–735, Nov. 1980.

[6] R. Capocelli, L. Gargano, and U. Vaccaro, "On the characterization of statistically synchronizable variable length codes," *IEEE Trans. Inform. Theory*, vol. 34, pp. 817–825, Jul. 1988.

[7] 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.

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

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

[10] R. Capocelli, A. Santis, L. Gargano, and U. Vaccaro, "On the construction of statistically synchronizable variable length codes," *IEEE Trans. Inform. Theory*, vol. 38, pp. 407–414, Mar. 1992.

[11] 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.

[12] Y. Takashima, M. Wada, and H. Murakami, "Error states and synchronization recovery for variable length codes," *IEEE Trans. Commun.*, vol. 42, pp. 783–792, Feb./Mar./Apr. 1994.

[13] 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.

[14] 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

[15] B. Tunstall, "Synthesis of noiseless compression codes," *Ph.D Dissertation, Georgia Institute of Technology, Atlanta*, 1967.

[16] S. Savari and R. Gallager, "Generalized tunstall codes for sources with memory," *IEEE Trans. Inform. Theory*, vol. 43, no. 2, pp. 658–668, Mar. 1997.

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

[18] N. Tanabe and N. Farvardin, "Subband image coding using entropy-coded quantization over noisy channels," *IEEE J. Select. Areas Commun.*, vol. 10, no. 5, pp. 926–942, June 1992.

[19] M. J. Ruf and J. W. Modestino, "Rate-distortion performance for joint source and channel coding of images," *Proc. Intl. Conf. Image Processing, ICIP*, vol. 2, pp. 77–80, Oct. 1995.

[20] G. Cheung and A. Zakhor, "Joint source-channel coding of scalable video over noisy channels," in *Proc. Intl. Conf. Image Processing, ICIP*, 1996, pp. 767–770.

[21] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Soviet Physics Doklady*, January 1963, *http://cr.yp.to/bib/1963/karatsuba.html*.

[22] D. Zuras, "More on squaring and multiplying large integers," *IEEE Trans. Comput.*, vol. 43, pp. 899–908, 1994.

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

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

[25] D. C. V. Voorhis, "Constructing codes with bounded codeword lengths," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 288–290, 1974.

[26] L. L. Larmore and D. S. Hirschberg, "A fast algorithm for optimal length-limited huffman codes," *J. Assoc. Computing Machinery*, vol. 37, pp. 464–473, 1990.

[27] "The gmp library," *http://www.swox.com/gmp/*.

[28] 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.

[29] C. Weidmann, "Reduced-complexity soft-in-soft-out decoding of variable length codes," in *Proc. Intl. Conf. Inform. Theory, ISIT*, July 2003, yokohama, Japan.

Appendix I
Proof of property 1

Let $\epsilon > 0$. The problem consists in finding a partition $\mathcal{C}$ that verifies the property. Let c_h be the integer defined as

$c_h = \lceil -log_2(\min_{i \in \mathcal{A}}(\mu_i)) \rceil$. By definition,

$$\forall\, i \in \mathcal{A},\ \mu_i \geq 2^{-c_h}. \tag{27}$$

For all $c \in \mathbb{N}$ such that $c \geq c_h$, we choose $|\mathcal{C}_i| = \lfloor \mu_i |\mathcal{X}| \rfloor$. From $\sum_{i \in \mathcal{A}} \mu_i = 1$ we get $\sum_{i \in \mathcal{A}} \lfloor \mu_i |\mathcal{X}| \rfloor \leq |\mathcal{X}| \sum_{i \in \mathcal{A}} \mu_i = |\mathcal{X}|$. Moreover, $\forall i,\ |\mathcal{C}_i| \geq 1$. Therefore this choice of $(|\mathcal{C}_i|)_{1 \leq i \leq |\mathcal{A}|}$ is valid. By construction, $\mu_i |\mathcal{X}| - 1 \leq |\mathcal{C}_i| \leq \mu_i |\mathcal{X}|$, hence,

$$1 \leq \mu_i \frac{|\mathcal{X}|}{|\mathcal{C}_i|} \leq 1 + \frac{1}{\mu_i |\mathcal{X}| - 1}. \tag{28}$$

The difference δ_h between the mdl and the entropy is given by

$$\begin{aligned} \delta_h = \hat{h} - h &= -\sum_{i \in \mathcal{A}} \mu_i\ log_2(\frac{|\mathcal{C}_i|}{|\mathcal{X}|}) + \sum_{i \in \mathcal{A}} \mu_i\ log_2(\mu_i) \\ &= \sum_{i \in \mathcal{A}} \mu_i\ log_2(\frac{\mu_i |\mathcal{X}|}{|\mathcal{C}_i|}). \end{aligned} \tag{29}$$

Therefore, from (28), it can be seen easily that δ_h verifies

$$\begin{array}{rcl} \delta_h & \leq & \sum_{i \in \mathcal{A}} \mu_i\ log_2(1 + \frac{1}{\mu_i |\mathcal{X}| - 1}) \\ \Rightarrow\ \delta_h & \leq & \sum_{i \in \mathcal{A}} \frac{\mu_i}{\mu_i |\mathcal{X}| - 1} = \sum_{i \in \mathcal{A}} \frac{1}{|\mathcal{X}| - \frac{1}{\mu_i}}. \end{array} \tag{30}$$

From equations (27) and (30) we get

$$\delta_h \leq \sum_{i \in \mathcal{A}} \frac{1}{|\mathcal{X}| - 2^{c_h}} = \frac{|\mathcal{A}|}{|\mathcal{X}| - 2^{c_h}}. \tag{31}$$

Thus, using $c = \lceil log_2(\frac{|\mathcal{A}|}{\epsilon} + 2^{c_h}) \rceil$ and $\forall i,\ |\mathcal{C}_i| = \lfloor \mu_i |\mathcal{X}| \rfloor$, the inequality $\delta_h \leq \epsilon$ is verified.

Notice that, for any code for which the inequality $c \geq c_h$ is true, this proof provides an upper bound for the mdl. □

Appendix II
Complexity of the hierarchical decomposition of the variable γ

The complexity of the hierarchical decomposition of the variable γ strongly depends on the complexity of the algorithms used for the multiplication, division and modulo operations. Karatsuba [21] has shown that these operations have a subquadractic complexity. For example, the multiplication can be done in $O(N^{\frac{log3}{log2}})$ with the Karatsuba algorithm. For implementation purpose, efficient algorithms are described in the GNU Multiple precision computing library [27]. Now, let us assume that the complexity of long integer operations are given by $C(K)$, where K is the block size of the largest variable involved. Then, processing the level j in Algorithm 2 has the complexity $2^{l-j}\, C(2^j) = K_H\, 2^{-j} C(2^j)$. Assuming that $C(K) = O(K^r)$ with $r > 1$, the overall complexity can be written as

$$K_H \sum_{j=1}^{l} 2^{-j}\, O((2^j)^r). \tag{32}$$

Hence, from $C(K) = O(K^r)$, we deduce that $\exists A > 0\ \exists B > 0\ /\ \forall K > 0,\ C(K) < A + B\, K^r$. This leads to the following inegality

$$K_H \sum_{j=1}^{l} 2^{-j} C(2^j) < K_H \sum_{j=1}^{l} 2^{-j} (A + B\, (2^j)^r) \tag{33}$$

$$< K_H A \sum_{j=1}^{l} 2^{-j} + K_H\, B \sum_{j=1}^{l} 2^{j(r-1)} \tag{34}$$

$$< K_H\, A + K_H\, B\, 2^{(l+1)(r-1)}. \tag{35}$$

Since we have

$$2^{(l+1)(r-1)} = (K_H)^{r-1}\, 2^{r-1}, \tag{36}$$

it appears that the complexity of processing the level l and the overall complexity are of the same order, i.e $O((K_H)^r)$.

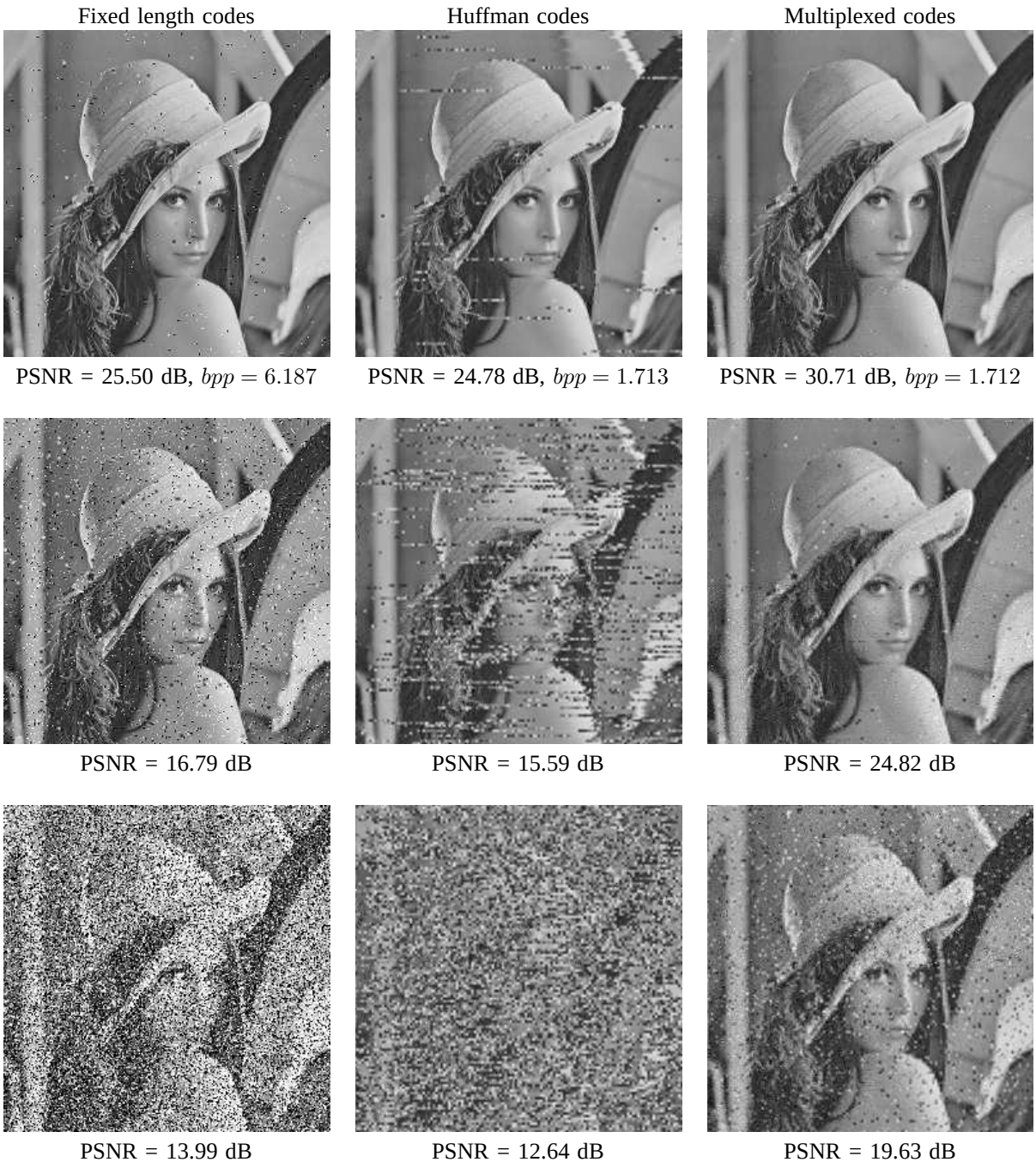

Fig. 8. PSNR performance and visual quality obtained respectively with FLCs, Huffman codes and multiplexed codes. The channel bit error rates are 0.0005 (top images), 0.005 (middle images) and 0.05 (bottom images).