Entropy coding with variable-length rewriting systems

Hervé Jégou, Christine Guillemot

Entropy coding with variable length re-writing systems

Hervé Jégou and Christine Guillemot

***Abstract*— This paper describes a family of codes for entropy coding of memoryless sources. These codes are defined by sets of production rules of the form $a\overline{l} \rightarrow \overline{b}$, where a is a source symbol and $\overline{l}, \overline{b}$ are sequences of bits. The coding process can be modeled as a finite state machine (FSM). A method to construct codes which preserve the lexicographic order in the binary coded representation is described. For a given constraint of the number of states for the coding process, this method allows the construction of codes with a better compression efficiency than the Hu-Tucker codes. A second method is proposed to construct codes such that the marginal bit probability of the compressed bitstream converges to 0.5 as the sequence length increases. This property is achieved even if the probability distribution function is not known by the encoder.**

***Index Terms*— source coding, joint source/channel codes, finite state machines, variable length codes, transducers, entropy codes, data compression, data communication**

I. Introduction

GRAMMARS are powerful tools which are widely used in computer sciences. Many lossless compression algorithms can be formalized with grammars. Codes explicitly based on grammars have been considered as a means for data compression [1]. These universal codes losslessly encode a sequence in two steps. A first step searches for the production rules. A second step applies these rules to the sequence to be encoded. These codes have mainly been compared with dictionary-based compression algorithms such as LZ77 [2] or [3], which also implicitly use the grammar formalism. All these codes have in common the fact that the set of production rules depends on the data to be encoded.

In this paper, a set of codes based on specific production rules is introduced. In contrast with grammar codes, the set of production rules is fixed for given source properties. These codes are implemented using FSM[1]. FSM have been considered in several contexts, many of them addressed in [5]. They have been shown to be useful in a context of quantization [6][7]. In [8], the authors show that finite-precision arithmetic codes can be implemented using FSM, hence avoiding using any arithmetic operation. Indeed, only table lookup is required by these FSM.

The form of production rules considered here for defining the code is presented in Section II. The sequence of bits generated by a given production rule may be re-written by a subsequent production rule. Hence, the set of productions rules form a non context-free grammar [9]. The corresponding encoding and decoding FSM are said to be *sequential*, which means that state transitions are triggered by a single symbol-input. Since their implementation only makes use of table lookups, the complexity of the encoding and decoding procedures is the same as the one of Huffman codes [10], which are encompassed by the proposed codes. A possible drawback of these codes is that they require backward encoding. However, since most applications deal with block encoding, the forward encoding property is not absolutely required. The decoding and encoding procedures are described in Section III. In Section IV the compression efficiency is analyzed. It is shown with an example that the proposed codes allow for better compression efficiency than Huffman codes applied on a symbol basis, for similar memory and complexity requirements.

Two code construction methods are then described. Both methods lead to codes with the same expected description length (EDL) as the code (e.g., Huffman code) from which they are constructed. The first method constructs a set of production rules preserving the lexicographic order of the original source alphabet in the binary coded representation. This property is of interest for database applications. It allows the processing of comparative queries directly in the binary coded representation, hence avoiding prior decoding of the compressed dictionary for the query itself. Note that the lexicographic variable length codes (VLC) of minimal expected length is obtained with the Hu-Tucker algorithm [11]. For some sources, the Hu-Tucker codes may have the same compression efficiency as Huffman codes, but it is not the case in general. The best lexicographic codes for sequences are generalized Hu-Tucker codes, i.e. Hu-Tucker codes applied on the alphabet of sequences of symbols ordered in a lexicographical manner. However, the number of nodes of the corresponding codetrees is of the order of the number of sequences, and is not tractable in practical systems. The method proposed in Section V constructs lexicographic codes with at least the same compression performance as Huffman codes. The resulting codes provide a better trade-off between compression efficiency and number of states than Hu-Tucker codes.

The transmission of entropy codes over noisy channels is then considered. Related work includes soft decoding of VLCs [12][13] and joint source-channel decoding of LZ-like source codes [14]. All these approaches aim at exploiting the residual redundancy of the source code. However, very few attention has been dedicated to improve the source code itself. For this purpose, a second construction method is proposed to construct codes, for memoryless stationary sources, such that

[1]In computer sciences, the term *transducer* is used instead of the term FSM. It is worth noting that Shannon used the term *transducer* in [4]. In the coding community, FSMs are also referred to as to *automaton*, although this term is less precise.

the marginal bit probability of the compression bitstream is equal to 0.5. The main advantage of these codes is that this probability is equal to 0.5 even if the actual source probabilities are not known at the encoder. Since channel coders widely assume that 0*s* and 1*s* have the same probability, this property is of interest when compressed bitstreams protected by such encoders are transmitted over noisy channels. Indeed, for a transmission scheme that makes use of a systematic error correcting code, a mismatch in the marginal bit probability leads to a capacity loss [15]. In Table I of [16], the gap to the optimal performance theoretically achievable has been shown to be quite important on Gaussian and Rayleigh channels. This observation has motivated the authors to use non-systematic turbo-codes for highly-biased sources [16]. Here, the problem is addressed by designing codes with a uniform marginal bit probability, leading to improve the soft decoding results of reversible variable length codes (RVLCs) and for which the use of systematic error correcting codes will satisfy the source/channel matching condition.

II. Notation and definitions

In the sequel, random variables are denoted by upper case and the corresponding realizations are denoted by lower case. Sets are denoted by calligraphic characters. The cardinality of a given set $\mathbf{X}$ is denoted $|\mathbf{X}|$. The concatenation of two sequences x and y is denoted xy. The t^{th} element of a sequence x is denoted x_t. The void sequence is denoted ε. This sequence is the neutral element for the concatenation, i.e. $\forall x,\ \varepsilon x = x\varepsilon = x$. The set of sequences of elements of $\mathbf{X}$ with length i is denoted $\mathbf{X}^i$. The length of a given sequence x is denoted $L(x)$. A sequence x is said to be a prefix of a sequence y if and only if $L(x) \leq L(y) \wedge \forall i \in \{1 \dots L(x)\},\ x_i = y_i$. In the following, we write $x \sqsubset y$ if x is a prefix of y, $x \not\sqsubset y$ otherwise. We define $\mathbf{X}^+ = \bigcup_{i=1}^{\infty} \mathbf{X}^i$ and $\mathbf{X}^* = \{\varepsilon\} \cup \mathbf{X}^+$. Hence $\mathbf{X}^*$ denotes the set of sequences composed of elements of $\mathbf{X}$.

Let $\mathbf{S} \in \mathcal{A}^+$ be a sequence of source symbols taking their values in a finite alphabet $\mathcal{A} = \{a_1, \dots, a_i, \dots\}$. The alphabet $\mathcal{A}$ is assumed to be ordered according to a total order $\prec$. Without loss of generality, we assume that $a_1 \prec a_2 \prec \dots \prec a_i \prec \dots \prec a_{|\mathcal{A}|}$. Let us define $\mathcal{B} = \{0, 1\}$. In the sequel, the emitted bitstream is denoted $\mathbf{E} = E_1 \dots E_{L(\mathbf{E})} \in \mathcal{B}^*$ and its realization is denoted $\mathbf{e} = e_1 \dots e_{L(\mathbf{e})}$.

Definition 1: A variable length re-writing system (VLRS) is a set $\mathcal{R} = \bigcup_{i \in \mathcal{A}} \mathcal{R}_i$, where $\mathcal{R}_i$ denotes the set of rules related to a given symbol a_i, with $|R_i| \geq 1$, defined as

$$
\begin{array}{lcll}
r_{1,1}: & a_1\,\overline{l}_{1,1} & \rightarrow & \overline{b}_{1,1} = b_{1,1}^1 \dots b_{1,1}^{L(\overline{b}_{1,1})}, \\
 & & \vdots & \\
r_{i,j}: & a_i\,\overline{l}_{i,j} & \rightarrow & \overline{b}_{i,j} = b_{i,j}^1 \dots b_{i,j}^{L(\overline{b}_{i,j})}, \\
 & & \vdots & \\
r_{|\mathcal{A}|,|\mathcal{R}_{|\mathcal{A}|}|}: & a_{|\mathcal{A}|}\,\overline{l}_{|\mathcal{A}|,|\mathcal{R}_{|\mathcal{A}|}|} & \rightarrow & \overline{b}_{|\mathcal{A}|,|\mathcal{R}_{|\mathcal{A}|}|}
\end{array}
$$

where $\overline{l}_{i,j} \in \mathcal{B}^*,\ \overline{b}_{i,j} \in \mathcal{B}^+$. This set is such that

1) The set $\bigcup_{i=1}^{|\mathcal{A}|} \bigcup_{j=1}^{|\mathcal{R}_i|} \{\overline{b}_{i,j}\}$ forms a prefix code (i.e. no codeword is the prefix of another codeword, see Chapter 5 of [17]) ;
2) $\forall j,\ \bigcup_{i=1}^{|\mathcal{R}_i|} \{\overline{l}_{i,j}\}$ is the set $\{\varepsilon\}$ or forms a full prefix code (i.e, such that the Kraft sum is equal to 1) ;
3) $\forall i, i'/i' \neq i, \forall j, j',\ \overline{b}_{i,j} = \overline{l}_{i',j'}$ or $\overline{b}_{i,j} \not\sqsubset \overline{l}_{i',j'}$.

These production rules can be regarded as successive transformations of a sequence $\mathbf{s}$ of symbols into a sequence $\mathbf{e}$ of bits. These rules are assumed to be reversible: inverting the direction of the arrow allows the recovery of a given sequence $\mathbf{s}$ from the corresponding bitstream $\mathbf{e}$. Note that a given production rule absorbs a symbol (a_i) and some bits ($\overline{l}_{i,j}$) to generate a given sequence of bits ($\overline{b}_{i,j}$).

The restricting assumptions proposed in Definition 1 are motivated as follows. Condition 1 ensures that the code is uniquely decodable without any ambiguity in the forward direction. Condition 2 ensures that, for any possible realization $\overline{x}$ of the sequence of bits following the symbol $S_t = a_i$ to be encoded, a production rule of the form $a_i\overline{y} \rightarrow \overline{b}$, such that $\overline{y} \sqsubset \overline{x}$, can be chosen to encode the realization of S_t. The encoding initialization step also ensures that the number of bits following the last symbol is sufficient to trigger at least one rule.

Note that from Condition 2 we deduce that $\forall i,\ |\mathcal{R}_i| \geq 1$. Hence, at least one production rule will be assigned to each symbol. Note that if $|\mathcal{R}_i| = 1$ we have $\overline{l}_{i,1} = \varepsilon$. More generally, a VLRS is a fixed-to-variable (F-to-V) length code if $\forall i\ |\mathcal{R}_i| = 1$ and $\overline{l}_i = \{\varepsilon\}$. Condition 3 is introduced to restrict the analysis to VLRS verifying the following condition: the knowledge of the rule applied for symbol s_t is sufficient to select the rule for the encoding of symbol s_{t-1}. This condition, which is not strictly required to define a valid encoding/decoding system, is used by the proof of Property 6 in Section IV.

Note that the usual fixed-to-variable length codes such as Huffman or Shannon codes, are covered by Definition 1. They correspond to the subset of VLRSs such that $\forall i\ |\mathcal{R}_i| = 1$.

Example 1: The code $\mathcal{C}_1 = \{0, 10, 11\}$ can be regarded as the VLRS defined as

$$
\mathcal{C}_1 = \left\{ \begin{array}{lcll} r_{1,1}: & a_1 & \rightarrow & 0 \\ r_{2,1}: & a_2 & \rightarrow & 10 \\ r_{3,1}: & a_3 & \rightarrow & 11. \end{array} \right.
$$

for which $l_{i,j} = \varepsilon$.

Definition 1 does not warranty that the system defines a valid prefix code. For example, a rule $r_{i,j}$ where $\overline{b}_{i,j} \sqsubset \overline{l}_{i,j}$ is not valid. In this paper, we will only consider VLRS defining valid prefix codes. For this purpose, it will also be assumed that the encoding process is ergodic. We will come back on this point in section IV. The suffix-constrained codes introduced in [18] form a subset of VLRS and are now characterized as follows.

Definition 2: A suffix-constrained code is a VLRS such that $\forall i, j,\ \overline{l}_{i,j}$ is suffix of $\overline{b}_{i,j}$.

By construction, a production rule of a suffix-constrained code

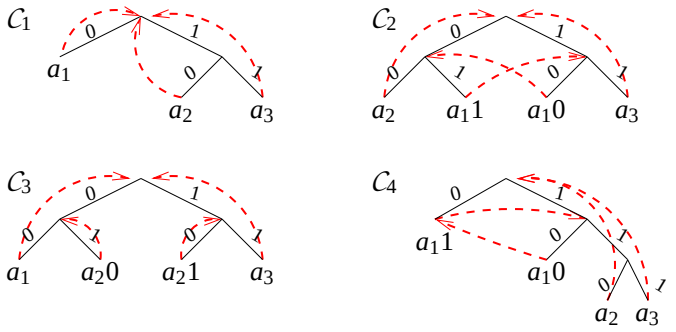

Fig. 1. Tree and FSM representation of $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$ and $\mathcal{C}_4$. The transitions triggered by the production rules are depicted by arrows.

generates the same bits as those it has absorbed. Hence, the following property:

Property 1: A bit generated by a production rule of a suffix-constrained code will not be modified by a subsequent production rule.

Example 2: The following VLRS $\mathcal{C}_2$ is a suffix-constrained code defined as:

$$\mathcal{C}_2 = \begin{cases} r_{1,1}: & a_1 0 & \rightarrow & 10 \\ r_{1,2}: & a_1 1 & \rightarrow & 01 \\ r_{2,1}: & a_2 & \rightarrow & 00 \\ r_{3,1}: & a_3 & \rightarrow & 11. \end{cases}$$

Note that Code $\mathcal{C}_2$ can not be encoded in the forward direction[2]. We will come back on this point in Section III. The lexicographic code $\mathcal{C}_3$ defined as

$$\mathcal{C}_3 = \begin{cases} r_{1,1}: & a_1 & \rightarrow & 00 \\ r_{2,1}: & a_2 0 & \rightarrow & 01 \\ r_{2,2}: & a_2 1 & \rightarrow & 10 \\ r_{3,1}: & a_3 & \rightarrow & 11 \end{cases}$$

and the code $\mathcal{C}_4$ defined as

$$\mathcal{C}_4 = \begin{cases} r_{1,1}: & a_1 1 & \rightarrow & 0 \\ r_{1,2}: & a_1 0 & \rightarrow & 10 \\ r_{2,1}: & a_2 & \rightarrow & 110 \\ r_{3,1}: & a_3 & \rightarrow & 111 \end{cases}$$

will also be considered in the sequel. This code $\mathcal{C}_4$ allows encoding the symbol a_1 with less than 1 bit. Note that these codes are not suffix-constrained codes.

VLRS can also be represented using trees, as depicted in Fig. 1. The tree structure corresponds to the one of the prefix code defined by $\bigcup_{i=1}^{|\mathcal{A}|}\bigcup_{j=1}^{|\mathcal{R}_i|}\{\overline{b}_{i,j}\}$. Leaves correspond to both the symbols $\{a_i\}_i$ and the sequences of bits $\{\overline{l}_{i,j}\}_{i,j}$.

III. Encoding and Decoding FSMs

On the encoder side, the production rules transform the sequence $\mathbf{s}$ into the sequence $\mathbf{e}$ of bits. Any segment of the sequence (composed of symbols *and* bits) can be rewritten if there exists a rule having this segment as an input (this input is composed of one symbol *and* a variable number of bits). In general, the set of rules defining a VLRS does not allow encoding the sequence $\mathbf{S}$ in the forward direction with a sequential FSM of reasonable dimension. Therefore, the encoding must be processed backward. To initiate the encoding process, a specific rule is used to encode the last symbol of the sequence. Indeed the last symbol may not be sufficient to trigger a production rule by itself. For this reason, some termination bits are concatenated to the sequence of symbols. The sequence of termination bits and its realization are denoted $\mathbf{U}$ and $\mathbf{u}$ respectively. They can be arbitrarily defined at the condition that the termination bit(s) do(es) not trigger a production rule by itself. Hence, the choice $\mathbf{u} = 0$ is valid for the codes $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ but should not be used for code $\mathcal{C}_4$, since 0 triggers the rule $r_{1,1}$.

Property 2: Transmitting the termination bit(s) $\mathbf{U}$ is not required for suffix-constrained codes.

Proof: Let us assume without loss of generality that the termination constraint $\mathbf{U}$ is arbitrarily chosen to be a sequence of 0. From Property 1, we deduce that $\mathbf{U}$ is the suffix of the intermediate re-written terms. Therefore $\mathbf{U}$ is also the suffix of the emitted bitstream $\mathbf{E}$. Hence, these bits are deterministically known on the decoder side and need not be transmitted. ■

Example 3: Let $\mathbf{s}_1 = a_1a_2a_2a_3a_2a_1a_1a_1$ be a sequence of symbols taking their values in the alphabet $\mathcal{A}_1 = \{a_1, a_2, a_3\}$. This sequence is encoded with the code $\mathcal{C}_2$. Since the last symbol is a_1, no rule applies directly. Therefore, the termination bit $\mathbf{u} = 0$ is concatenated to this sequence in order to initiate the encoding. The encoding then proceeds as follows:

$$\begin{aligned} r_{1,1}: \quad \mathbf{s}_1\mathbf{u} &= a_1a_2a_2a_3a_2a_1a_1\underline{a_1 0} \\ r_{1,2}: \quad &= a_1a_2a_2a_3a_2a_1\underline{a_1 1}0 \\ r_{1,1}: \quad &= a_1a_2a_2a_3a_2\underline{a_1 0}10 \\ r_{2,1}: \quad &= a_1a_2a_2a_3\underline{a_2}1010 \\ r_{3,1}: \quad &= a_1a_2a_2\underline{a_3}001010 \\ r_{2,1}: \quad &= a_1a_2\underline{a_2}11001010 \\ r_{2,1}: \quad &= a_1\underline{a_2}0011001010 \\ r_{1,1}: \quad &= \underline{a_1 0}0011001010 \\ \mathbf{e}_1\mathbf{u} &= 10011001010 \end{aligned}$$

Example 3 illustrates Property 2. The termination bit $\mathbf{u} = 0$ is not modified by the successive applications of the production rules. This bit is deterministically known on the decoder side. Hence, the suffix does not need to be transmitted. Since these termination bits may be required in the general case of VLRS, it will be assumed in the following that they are transmitted to the decoder. In the following example, the termination bit must be $\mathbf{u} = 1$.

Example 4: Let us now consider the sequence $\mathbf{s}'_1 = a_1a_1a_1a_1a_1$. This sequence is encoded with code $\mathcal{C}_4$ as

$$\begin{aligned} r_{1,1}: \quad \mathbf{s}'_1\mathbf{u}' &= a_1a_1a_1a_1\underline{a_1 1} \\ r_{1,2}: \quad &= a_1a_1a_1\underline{a_1 0} \\ r_{1,1}: \quad &= a_1a_1\underline{a_1 1}0 \\ r_{1,2}: \quad &= a_1\underline{a_1 0}00 \\ r_{1,1}: \quad &= \underline{a_1 1}00 \\ \mathbf{e}'_1 &= 000 \end{aligned}$$

[2]Theoretically, it can. However, the dimension of the encoding FSM with respect to the number of states is not tractable.

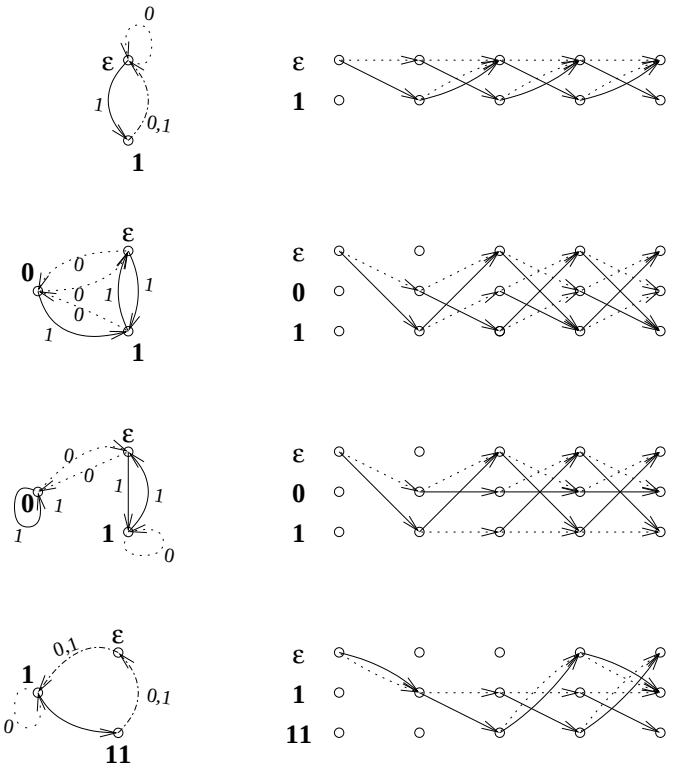

Fig. 2. Decoding FSMs corresponding to the codes $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$ and $\mathcal{C}_4$ and corresponding decoding trellises. Transitions corresponding to 0s and 1s are represented with dotted and solid lines respectively.

Note that the sequence is encoded with less than 1 bit per symbol. The decoding is processed forward using the reverse rules. The encoding and decoding algorithms are implemented using FSMs. These FSMs are used to catch the memory of the encoding and decoding processes. They have to be constructed so that the states include the knowledge of the bits that may be re-written or used to select the next production rule. The transitions on the FSM representing the encoding process are triggered by symbols. The internal states of the encoding FSM are given by the variable length segments of bits $\{\overline{l}_{i,j}\}$. This FSM may be simplified if a variable length bit segment $\overline{l}_{i,j}$ is a prefix of another segment $\overline{l}_{i',j'}$ (in that case, according to Definition 1, we have $i \neq i'$). For the code $\mathcal{C}_1$, we have $\forall i,j,\ \overline{l}_{i,j} = \varepsilon$. Therefore, there is only one internal state $\{\varepsilon\}$ for the encoding FSM corresponding to the code $\mathcal{C}_1$. The sets of states of encoding FSMs of codes $\mathcal{C}_2$, $\mathcal{C}_3$ and $\mathcal{C}_4$ are identical and are equal to $\{\mathbf{0}, \mathbf{1}\}$. Note that the termination sequence $\mathbf{u}$ allows uniquely identifying the starting state of the encoding FSM.

The states of the decoding FSMs correspond to bit segments that have already been decoded, but which are not sufficient to identify a symbol. For VLCs such as Huffman codes, these internal states correspond to the internal nodes of the codetree.

Example 5: The sets of states for the decoding FSMs corresponding to the codes $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$ and $\mathcal{C}_4$ are respectively $\{\varepsilon, \mathbf{1}\}$, $\{\varepsilon, \mathbf{0}, \mathbf{1}\}$, $\{\varepsilon, \mathbf{0}, \mathbf{1}\}$ and $\{\varepsilon, \mathbf{1}, \mathbf{11}\}$.

Since the bit segments, that are not sufficient to identify a symbol, correspond to the set of strict[3] prefixes of the set of codewords $\bigcup_{i,j} \overline{b}_{i,j}$, we deduce the following property.

Property 3: If the prefix code $\bigcup_{i,j} \overline{b}_{i,j}$ is a full prefix code, then the number of states of the decoding FSM is equal to $|\mathcal{R}| - 1$.

Proof: Since the number of internal nodes of a full prefix code is also equal to the number of strict prefixes of the code, the property straightforwardly stems from the classical property that the number of internal nodes of a full prefix code composed of n symbols is equal to $n - 1$. ■

According to Property 3, the number of production rules is directly linked to the number of states. Consequently, it is of interest to keep the number of rules as low as possible. For this purpose, the following property may be used to reduce the number of production rules without modifying the function defined by the FSM.

Property 4: Let $r_{i,j}$ and $r_{i,j'}$ be two production rules such that

$$\begin{cases} l_{i,j}^1 \dots l_{i,j}^{L(\overline{l}_{i,j})-1} &= l_{i,j'}^1 \dots l_{i,j'}^{L(\overline{l}_{i,j'})-1} \\ b_{i,j}^1 \dots b_{i,j}^{L(\overline{b}_{i,j})-1} &= b_{i,j'}^1 \dots b_{i,j'}^{L(\overline{b}_{i,j'})-1} \\ l_{i,j}^{L(\overline{l}_{i,j})} &= b_{i,j}^{L(\overline{b}_{i,j})}, \end{cases}$$

then these rules can be merged into a single rule

$$a_i \overline{l}_{i,j}^1 \dots \overline{l}_{i,j}^{L(\overline{l}_{i,j})-1} \rightarrow \overline{b}_{i,j'}^{L(\overline{b}_{i,j'})-1}$$

which is equivalent to the set of two rules.

The graphical representations of the decoding FSMs may be deduced from the tree representations given in Fig. 1. These FSMs and the corresponding trellises are depicted in Fig. 2. For sake of clarity, the symbols generated by the bit transitions are not shown. However, note that the set of generated symbol(s) must also be associated with each bit transition. For the codes $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$, at most 1 symbol is associated with each bit transition. It is not the case for the code $\mathcal{C}_4$, where the transition starting from decoding state $\mathbf{1}$ triggered by the bit 0 generates the symbol a_1 twice. As shown in Example 4 and demonstrated in Section IV, this transition allows encoding long sequences of a_1 with less than 1 bit, at the expense of a higher encoding cost for the symbols a_2 and a_3.

IV. Compression efficiency

This section analyzes the compression efficiency of VLRSs for a memoryless stationary source. First, the asymptotic EDL is given when the sequence length increases to infinity. This quantity is denoted EDL_∞ in the following. Next, we discuss the compression efficiency for sequences of finite length.

Let us assume that $\mathbf{S}$ is an independent identically distributed (i.i.d.) source characterized by its probability mass function (PMF) on $\mathcal{A}$:

$$\boldsymbol{\mu} = \{\mathbb{P}(a_1), \dots, \mathbb{P}(a_i), \dots\}.$$

[3]A codeword x is said to be a strict prefix of y if $x \sqsubset y$ and $x \neq y$.

Let

$$\delta(r_{i,j}) \triangleq L(\overline{b}_{i,j}) - L(\overline{l}_{i,j}) \quad (1)$$

denote the number of bits generated by a given production rule $r_{i,j}$.

Property 5: Let $\mathcal{C}$ be a VLRS such that $\forall i,\ \forall j,j'\ \delta(r_{i,j}) = \delta(r_{i,j'})$. Then we have

$$\mathrm{EDL}_\infty(\mathcal{C}) = \sum_{a_i \in \mathcal{A}} \mathbb{P}(a_i)\delta(r_{i,1}). \quad (2)$$

Proof: Let $S_t = a_i$ be the symbol to be encoded at instant t. This symbol is encoded using a rule $R_t = r_{i,j}$. From the assumption $\forall j,j'\ \delta(r_{i,j}) = \delta(r_{i,j'})$, we deduce that the number of bits generated by the rule r_t does not depend on j, hence $\delta(r_{i,j}) = \delta(r_{i,1})$. Then we have $\mathbb{E}(\delta(R_t)) = \sum_{a_i \in \mathcal{A}} \mathbb{P}(a_i)\delta(r_{i,1})$. Note that this expectation does not depend on the instant t. Therefore, the expectation of the bitstream length $L(\mathbf{E})$ is given by $L(\mathbf{S})\cdot\mathbb{E}(\delta(R_t))+\mathbb{E}(L(\mathbf{U}))$. The expectation $\mathbb{E}(L(\mathbf{U}))$ of the length of the termination constraint is bounded by the quantity $\max_{i,j} l_{i,j}$, which does not depend on the length $L(\mathbf{S})$. Consequently,

$$\mathrm{EDL}_\infty(\mathcal{C}) = \lim_{L(\mathbf{S})\to\infty} \frac{1}{L(\mathbf{S})}\left(L(\mathbf{S})\cdot\mathbb{E}(\delta(R_t)) + \mathbb{E}(L(\mathbf{U}))\right) \quad (3)$$

$$= \sum_{a_i \in \mathcal{A}} \mathbb{P}(a_i)\delta(r_{i,1}). \quad (4)$$

■

Note that the proof of Property 5 does not require that the source is memoryless.

Example 6: Let us assume that $\mathbf{S}$ *is a source of stationary probabilities* $\boldsymbol{\mu}_1 = \{0.7, 0.2, 0.1\}$. *The entropy of this source is* 1.157. *The expected length of the code* $\mathcal{C}_1$ *is equal to* 1.3. *For the code* $\mathcal{C}_2$, *we have* $\delta(r_{1,1}) = \delta(r_{1,2}) = 1$ *and* $\delta(r_{2,1}) = \delta(r_{3,1}) = 2$. *The expected length of this code is also equal to* 1.3.

Property 6: Let $\mathcal{C}$ be a VLRS and $\mathbf{S}$ an i.i.d. source. The process $(Z_{t'})_{t'} = (R_{L(\mathbf{S})}, \ldots, R_t, \ldots, R_1)$ obtained from the process $(R_t)_t$ by reversing the symbol instant t as $t' = L(\mathbf{S}) - t + 1$ forms a homogeneous Markov chain of transition probabilities

$$\lambda_{i,j,i',j'} \triangleq \mathbb{P}(Z_{t'} = r_{i,j} | Z_{t'-1} = r_{i',j'}) \quad (5)$$

$$= \begin{cases} \mathbb{P}(a_i) & \text{if } \overline{l}_{i,j} \sqsubset \overline{b}_{i',j'}, \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

Proof: Let $R_t : S_t\overline{L}_t \to \overline{B}_t$ denote the rule used to encode the symbol S_t. From condition 3 of definition 1 we deduce that the tuple $(\overline{B}_{t+1}, S_t)$ and *a fortiori* the tuple (R_{t+1}, S_t) suffice to identify the rule R_t to trigger. Therefore, knowing the realization of R_{t+1}, the probabilities of the variable R_t are governed by the statistics of the source $\mathbf{S}$. From the memoryless assumption on the source $\mathbf{S}$, we deduce that the probabilities of the event R_t are fully defined by the knowledge of the event R_{t+1}, which leads to conclude that Z_t is a Markov chain. With the property that the elements of $\mathbf{S}$ are identically distributed, we also deduce that this chain is homogeneous. Note that the event $R_t = r_{i,j}$ occurs if and only if $S_t = a_i \wedge \overline{l}_{i,j} \sqsubset \overline{b}_{i',j'}$, where the codeword $\overline{b}_{i',j'}$ is the realization of $\overline{B}_{t+1}$ and has been generated by the previous production rule (at instant $t+1$). As a consequence, the probability $\lambda_{i,j,i',j'}$ is deduced from the source PMF as

$$\lambda_{i,j,i',j'} = \mathbb{P}(R_t = r_{i,j} | R_{t+1} = r_{i',j'}) \quad (7)$$

$$= \mathbb{P}(S_t = a_i, \overline{L}_t = \overline{l}_{i,j} | \overline{B}_{t+1} = \overline{b}_{i',j'}) \quad (8)$$

$$= \begin{cases} \mathbb{P}(a_i) & \text{if } \overline{l}_{i,j} \sqsubset \overline{b}_{i',j'}, \\ 0 & \text{otherwise.} \end{cases} \quad (9)$$

■

Let us denote $\Lambda = (\lambda_{i,j,i',j'})_{(i,j),(i',j')}$ the matrix of transition probabilities of the process $(Z_{t'})$. This matrix being known, it is possible to check whether the corresponding process is irreducible and aperiodic or not. Let us denote $\mathbb{I}$ the identity matrix which has the same dimension as Λ. If the matrix $\mathbb{I} - \Lambda$ is invertible, the solution of the linear system $\pi = \Lambda\pi$ is unique (see, e.g., [17]). In the following, we will assume that the VLRS and the source are defined so that the encoding process represented by $(Z_{t'})_{t'}$ is ergodic. If it is not the case, some rules can not be applied anymore when the sequence length becomes sufficiently high. This may occur when 1) some production rules of the VLRS are useless, 2) the source is singular ($\exists i /\ \mathbb{P}(a_i) = 0$) or 3) the VLRS does not define a valid prefix code.

The Markov chain $(Z_{t'})$ being irreducible, the marginal probability distribution $\pi \triangleq \mathbb{P}(Z_{t'} = r_{i,j})$ is obtained from the transition matrix Λ as the unique solution of the equation $\pi = \Lambda\pi$ such that $||\pi||_1 = 1$. Therefore π is the normalized eigenvector associated with the eigenvalue 1. As t' grows to infinity (which requires that $t \to \infty$), the quantity $\delta(Z_{t'})$ is the expectation of the number of bits generated by a production rule. Given that $\delta(\mathbf{U})$ is a constant, from the Cesaro theorem, one can deduce the asymptotic value of the expected length as the sequence length increases, i.e. the quantity $\mathrm{EDL}_\infty(\mathcal{C})$.

Example 7: For the code $\mathcal{C}_4$, *the transition matrix corresponding to the source PMF of Example 6 is*

$$\begin{bmatrix} 0 & 0.7 & 0.7 & 0.7 \\ 0.7 & 0 & 0 & 0 \\ 0.2 & 0.2 & 0.2 & 0.2 \\ 0.1 & 0.1 & 0.1 & 0.1 \end{bmatrix},$$

which leads to $\mathbb{P}(R_t = r_{i,j}) = \{0.412, 0.288, 0.2, 0.1\}$. *Finally, the expected length of this code is* $EDL(\mathcal{C}_4) = 0.412 \cdot 0 + 0.288 \cdot 1 + 0.2 \cdot 3 + 0.1 \cdot 3 = 1.188$.

The expected length obtained in Example 7 is much closer to the entropy than the expected length obtained with Huffman codes. The expected number of bits required to code the symbol a_1 is less than 0.5 bit. One can also process the exact expected length of a VLRS for sequences of finite length. Indeed, the expectation of the number of termination bit(s) as well as the PMF $\mathbb{P}(R_t = r_{i,j} | t = L(\mathbf{S}))$ of the last rule can be obtained from the termination bit choice and from the source

PMF The exact probability $\mathbb{P}(R_t = r_{i,j}|t = \tau)$ of having a given rule for a given instant τ can then be computed and subsequently one can deduce the expectation of the number of bits generated to encode the symbol S_τ.

A first-order approximation of this expected length may be obtained by assuming that $0s$ and $1s$ have the same probability, hence allowing to compute directly the probabilities $\mathbb{P}(R_t)$. For the code $\mathcal{C}_4$, this approximation leads to $\mathrm{EDL}(\mathcal{C}_4) = 1.25$.

V. Lexicographic code design

In order to illustrate the strong expressiveness of VLRSs, we describe in this section a VLRS construction method which allows to preserve the lexicographic order of the source alphabet in the binary coded representation. As a starting point, we assume that the Huffman code corresponding to the source PMF $\boldsymbol{\mu}$ is already known. The length of the Huffman codeword associated with the symbol a_i is denoted k_i. Let $k^+ = \max_i k_i$ denote the length of the longest codeword.

First, let us underline that the union $\bigcup_{i,j}\{\overline{b}_{i,j}\}$ of all the bit sequences $\overline{b}_{i,j}$ forms a Fixed Length Code (FLC) $\mathcal{F}$ of length k^+. $\mathcal{F}$ contains 2^{k^+} codewords. These codewords will be assigned to production rules in the lexicographic order. Starting with the first symbol a_1, the algorithm proceeds as follows

1) $2^{k^+-k_i}$ rules are defined for the symbol a_i.
2) The left part of these rules are defined so that the set $\{\overline{l}_{i,j}\}_{j\in\{1..|\mathcal{R}_i|\}}$ forms a FLC of length $k^+ - k_i$. If $k_i = k^+$, this FLC only contains the element ε.
3) The $2^{k^+-k_i}$ smallest remaining codewords of $\mathcal{F}$, i.e. those which have not been assigned to previous symbols of $\mathcal{F}$, are then assigned to these production rules so that $\forall j, \overline{l}_{i,j} \leq \overline{l}_{i,j'} \Rightarrow \overline{b}_{i,j}\overline{l} \leq \overline{b}_{i,j'}$.
4) If $i = |\mathcal{A}|$, the construction procedure is completed. Otherwise the algorithm restarts at Step 1 with the symbol a_{i+1}.

By construction, the proposed algorithm leads to a VLRS with the lexicographic property and with the same compression efficiency as the code from which it is constructed. In some cases, the set of production rules generated in previous steps may be simplified according to property 4.

Example 8: Let us now assume that the source $\mathbf{S}$ *is memoryless of PMF* $\boldsymbol{\mu}_2 = \{0.2, 0.7, 0.1\}$. *Since* a_2 *has the highest probability, the Huffman code* $\mathcal{H}_2 = \{10, 0, 11\}$ *corresponding to this PMF is not lexicographic. The VLRS is constructed according to the proposed construction procedure.For* $\mathcal{H}_2$, *we have* $k_2 = 1$ *and* $k_1 = k_3 = k^+ = 2$. *Hence* $\mathcal{F} = \{00, 01, 10, 11\}$. *Since* $k_1 = 2$, *only* 1 *production rule* $r_{1,1}$ *is assigned to the symbol* a_1 *and* $b_{1,1} = \varepsilon$, *which implies* $r_{1,1} : a_1 \rightarrow 00$. *The symbol* a_1 *is then assigned two production rules* $r_{2,1}$ *and* $r_{2,2}$ *as follows:*

$$r_{2,1} : a_2 0 \ \rightarrow \ 01$$
$$r_{2,2} : a_2 1 \ \rightarrow \ 10.$$

The construction algorithm finishes with the assignment of rule $r_{3,1} : a_3 \rightarrow 11$ *to symbol* a_3. *Finally, we obtain the code* $\mathcal{C}_3$ *proposed in Section II, for which the EDL is equal to* 1.3 *and which has the lexicographic property. The Hu-Tucker code associated with this source is the code* $\mathcal{C}_1$ *proposed in Example 1 and its EDL is equal to* 1.8.

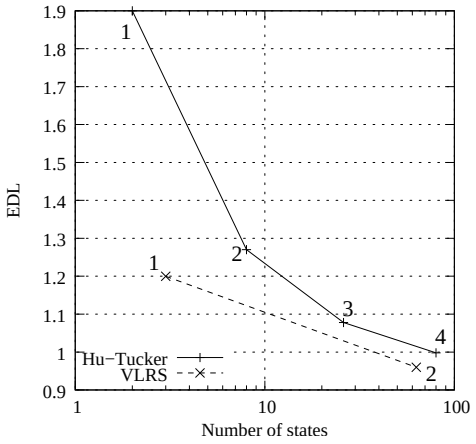

Fig. 3. Proposed lexicographic construction method against Hu-Tucker codes: trade-off between the EDL and the number of states. The numbers indicate the block lengths used for the source vectorisation.

In Example 8, the corresponding FSMs do not have the same number of states (2 for the Hu-Tucker code *versus* 3 for the proposed VLRS). The EDL is compared in Figure 3 with respect to the number of states in the decoding FSM for a 3-symbol source. The performance depicted are the ones of the generalized Hu-Tucker codes and the lexicographic VLRSs constructed from the generalized Huffman codes. Let us recall that, by generalized, we mean the Hu-Tucker and Huffman codes applied on the product alphabet $\mathcal{A}^n$. Such an alphabet contains $|A|^n$ elements. That example evidences that the Hu-Tucker codes are outperformed by the proposed VLRSs for this particular source for the trade-off between the compression efficiency and the number of states of the decoder. However, it is worth noticing that, even though the proposed construction allows to obtain lexicographic codes with the same compression efficiency as codes from which they are constructed, it does not construct, in general, the best lexicographic VLRSs from a compression efficiency point of view. Hence, finding some lexicographic VLRS with lower EDL with respect to the number of states is still an opened issue.

VI. Mirror Code Design

The code design described in this section allows to obtain codes with marginal bit probabilities that are asymptotically equal to 0.5 as the sequence length increases. Let us consider a VLC code $\mathcal{H} = \{\overline{b}_{1,1}, \ldots, \overline{b}_{|\mathcal{A}|,1}\}$ and the code $\tilde{\mathcal{H}} = \{\tilde{\overline{b}}_{1,1}, \ldots, \tilde{\overline{b}}_{|\mathcal{A}|,1}\}$ defined so that each bit transition of the codetree characterizing $\tilde{\mathcal{H}}$ is the opposite value from the corresponding bit transition in $\mathcal{H}$, as depicted in Fig. 4.

The VLRS is obtained by putting together these two codes. The codes $\mathcal{H}$ and $\tilde{\mathcal{H}}$ are respectively used to define the two

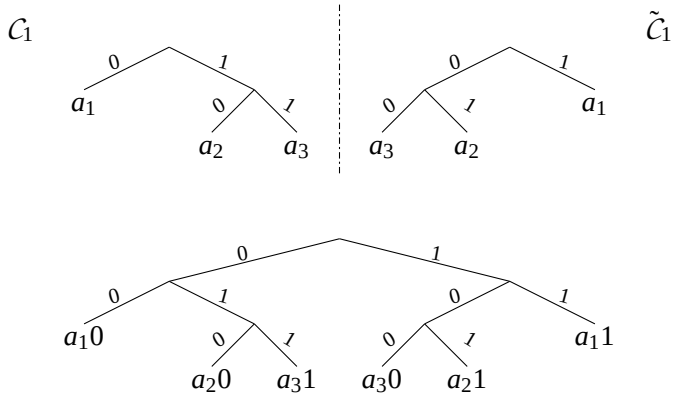

Fig. 4. Primitive code $\mathcal{C}_1$, its opposite $\tilde{\mathcal{C}}_1$ and the resulting mirror VLRS.

sets of $|\mathcal{A}|$ production rules forming the new VLRS as

$$\mathcal{M} = \begin{cases} \{a_i b_{i,1}^{L(\overline{b}_{i,1})} \rightarrow 0\,\overline{b}_{i,1}\}_{i\in\{1..|\mathcal{A}|\}} \\ \{a_i \tilde{b}_{i,1}^{L(\tilde{\overline{b}}_{i,1})} \rightarrow 1\,\tilde{\overline{b}}_{i,1}\}_{i\in\{1..|\mathcal{A}|\}}. \end{cases} \tag{10}$$

Note that the production rules associated with codes $\mathcal{H}$ and $\tilde{\mathcal{H}}$ respectively define the subtrees corresponding to bit transitions 0 and 1.

Property 7: The VLRS $\mathcal{M}$ is a suffix-constrained code.

Proof: By construction. ∎

Example 9: The construction associated with the code $\mathcal{C}_1$ leads to the following VLRS:

$$\underbrace{\begin{cases} r_{1,1}: & a_1 0 & \rightarrow & 00 \\ r_{2,1}: & a_2 0 & \rightarrow & 010 \\ r_{3,1}: & a_3 1 & \rightarrow & 011 \end{cases}}_{\text{obtained from } \mathcal{H}=\mathcal{C}_1}$$

$$\underbrace{\begin{cases} r_{1,2}: & a_1 1 & \rightarrow & 11 \\ r_{2,2}: & a_2 1 & \rightarrow & 101 \\ r_{3,2}: & a_3 0 & \rightarrow & 100. \end{cases}}_{\text{obtained from } \tilde{\mathcal{H}}=\tilde{\mathcal{C}}_1} \tag{11}$$

Property 8: $\forall n,\ \lim_{L(\mathbf{S})\rightarrow\infty} \mathbb{P}(E_n = 0) = 0.5$.

Proof: Let us consider a VLRS $\mathcal{M}$ constructed according to the previous guidelines. The notation $b_{i,j}$ refers to this VLRS (not to the VLC from which it is constructed). Let $f_t = \mathbb{P}(B_t^1 = 0)$ denote the marginal bit probability associated with the first bit generated by a given production rule. Since the VLRS is constructed from a VLC, we have $\forall i, j,\ \delta(r_{i,j}) \geq 1$, which means that every rule produces at least one bit. The value f_t can be written as

$$f_t = \sum_{i\in\{1..|\mathcal{A}|\}, j\in\{1,2\}} \mathbb{P}(R_t = r_{i,j}, B_t^1 = 0) \tag{12}$$

$$= \sum_{i\in\{1..|\mathcal{A}|\}, j=1} \mathbb{P}(S_t = a_i, B_{t+1}^1 = l_{i,1}^{L(l_{i,1})}) \tag{13}$$

$$= \sum_{i\in\{1..|\mathcal{A}|\}, j=1} \mathbb{P}(S_t = a_i, L_t^{L(L_t)} = 0)\, f_{t+1} + \sum_{i\in\{1..|\mathcal{A}|\}, j=1} \mathbb{P}(S_t = a_i, L_t^{L(L_t)} = 1)\,(1 - f_{t+1}). \tag{14}$$

Let

$$\alpha \triangleq \sum_{a_i\in\mathcal{A}} \mathbb{P}(a_i, l_{i,1}^{L(\overline{l}_{i,1})} = 0). \tag{15}$$

This entity corresponds to the sum of the probabilities of the symbols to which a codeword ending with 0 has been assigned. Note that $\forall i,\ \mathbb{P}(a_i) > 0 \Rightarrow 0 < \alpha < 1$. Inserting this entity in Eqn. 14, we obtain

$$f_t = \alpha\, f_{t+1} + (1-\alpha)(1 - f_{t+1}). \tag{16}$$

We can now study the asymptotic behavior of this sequence as $t' = L(\mathbf{S}) - t + 1$ tends to $+\infty$ (note that $f_{L(\mathbf{S})}$ is a constant). The absolute value of the derivative of the function $g(x) = \alpha\, x + (1-\alpha)(1-x)$ is strictly lower than 1 when $0 < \alpha < 1$. Consequently, the fixed-point theorem applies and the sequence $f_{L(\mathbf{S})}, f_{L(\mathbf{S})-1}, \ldots, f_{t'}$ converges to the solution of $x = g(x)$, which is 0.5. Subsequently, $\forall i$, opposite codewords $\overline{b}_{i,1}$ and $\overline{b}_{i,2}$ are equiprobable,

Since $\mathcal{M}$ is a suffix-constrained code, a bit produced by a production rule R_t is not modified by any subsequent production rule. Hence, the bits $b_{i,j}$ produced by the rules of the code $\mathcal{M}$ actually correspond to the bits forming the bitstream $\mathbf{S}$. This concludes the proof. ∎

VII. Soft decoding and simulation results

The amenability of the mirror codes to improve the performance of soft decoding has been assessed by simulations. For this purpose, we have considered the RVLC $\mathcal{C}_{12}$ of [19], defined as

$$\mathcal{C}_{12} = \{00, 11, 010, 101, 0110\}.$$

The corresponding mirror code is then defined by the set of rules that follows.

$$\begin{cases} a_1 0 & \rightarrow & 000 \\ a_1 1 & \rightarrow & 111 \\ a_2 0 & \rightarrow & 100 \\ a_2 1 & \rightarrow & 011 \\ a_3 0 & \rightarrow & 0010 \\ a_3 1 & \rightarrow & 1101 \\ a_4 0 & \rightarrow & 1010 \\ a_4 1 & \rightarrow & 0101 \\ a_5 0 & \rightarrow & 00110 \\ a_5 1 & \rightarrow & 11001. \end{cases}$$

Since this code is reversible, this set of rules can also be considered by reversing both the codewords and by switching

the symbol and the bit in the left part of the rule. It amounts to considering rules of the form $\overline{l}a \rightarrow \overline{b}$ instead:

$$\begin{cases} 0a_1 & \rightarrow & \mathit{0}00 \\ 0a_2 & \rightarrow & \mathit{0}01 \\ 0a_3 & \rightarrow & \mathit{0}100 \\ 0a_4 & \rightarrow & \mathit{0}101 \\ 0a_5 & \rightarrow & \mathit{0}1100 \\ 1a_1 & \rightarrow & \mathit{1}11 \\ 1a_2 & \rightarrow & \mathit{1}10 \\ 1a_3 & \rightarrow & \mathit{1}011 \\ 1a_4 & \rightarrow & \mathit{1}010 \\ 1a_5 & \rightarrow & \mathit{1}0011. \end{cases}$$

Since the original code was suffix free, the VLC associated with the mirrored code is also suffix free. Consequently, the code defined by the inverted rules is prefix free. Moreover, by construction the bit conditioning the choice of the rule is not modified by a subsequent rule. The set of codewords associated with a given conditioning bit also forms a prefix free code. As a consequence, a sequence encoded with a mirrored code can be instantaneously decoded in the backward direction as well. The states of the corresponding decoder then correspond to the internal nodes of the codetree formed by the right part of the production rules. In our example, these internal nodes correspond to the code prefixes

$$\{\varepsilon, 0, 1, 00, 01, 10, 11, 010, 0110, 101, 1001\}.$$

It is of interest because it is then possible to apply either the BCJR algorithm or the Viterbi algorithm on the state model composed of these states [4]. Note that this model is similar to the one proposed by Balakirsky for VLCs in [12].

Remarks:

- The number of states of the state model is equal to the number of internal nodes of the code formed by the right part of the rules (11 in the example). Hence, it is about twice the number of states of the state model associated with the original RVLC.
- The reversibility property of the VLC has been explicitely used for the construction of the state model. Although mirror codes constructed from Huffman codes are likely to admit a bayesian estimation as well, the state model associated with this estimation will increase with the sequence length.

This soft decoding on this state model has then been applied for two sources distribution. The first distribution is quite balanced and given by

$$\mu_1 = \{0.4, 0.2, 0.2, 0.1, 0.1\}.$$

Its marginal bit probability is equal to 0.6. The second source is defined as

$$\mu_1 = \{0.9, 0.025, 0.025, 0.025, 0.025\}$$

with $\mathbb{P}(0) = 0.917$.

[4]It can also be seen as a couple $(bit, \mathcal{N})$ comprising the last bit (in the backward direction) and an internal node of a new codetree with $|\mathcal{A}|$ leaves deduced from the set of rules, here defined as $\{00, 01, 100, 101, 1100\}$.

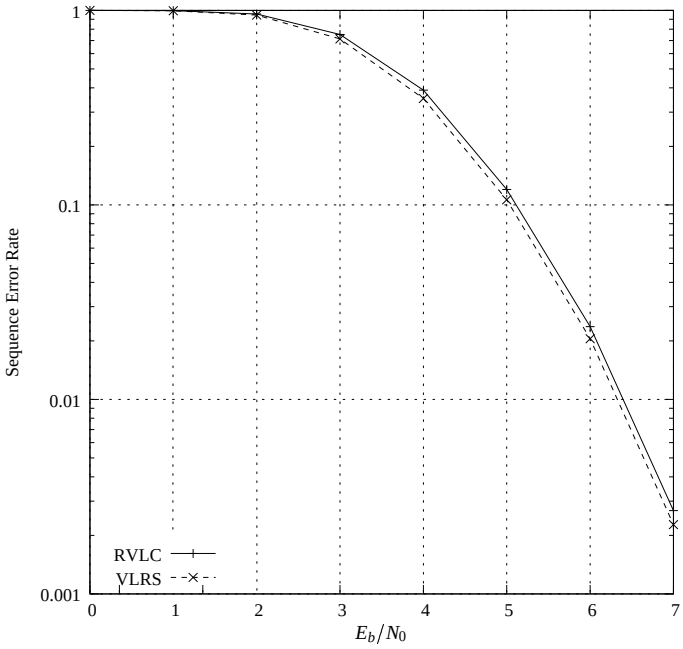

Fig. 5. Soft Decoding of Mirrored VLRS: the balanced case.

The proposed code has been compared against the original code and depicted in Fig. 5 and 6. The simulations have been performed assuming an additive white Gaussian noise channel together with a binary phase shift keying modulation. The estimation has been performed using the Viterbi algorithm in order to minimize the sequence error rate. As expected, the improvement is important for the unbalanced source only. Note however that this improvement is free since the expected length of the code is identical to the one of the original code.

VIII. Concluding remarks

This paper has described new families of source codes based on VLRSs. These codes can be modeled as FSM. They offer a good trade-off between the number of states, hence coding/decoding complexity, and compression efficiency. In this paper, VLRSs have been defined by rules of the form $a\overline{l} \rightarrow \overline{b}$. The same kind of analysis applies if rules of the form $\overline{l}a \rightarrow \overline{b}$ are considered instead. The VLRS formalism allows the design of codes with various interesting properties, such as preserving the lexicographical order of the symbol binary representation, or producing compressed bitstreams with a uniform marginal bit probability. The interest of this property is illustrated by an increased MAP estimation performance of these codes in presence of transmission errors.

References

[1] J. Kieffer and E.-H. Yang, "Grammar-based codes: a new class of universal lossless source codes," *IEEE Trans. Inform. Theory*, vol. 46, pp. 737–754, 2000.

[2] J. Ziv and A. Lempel, "A universal algorithm for data compression," *IEEE Trans. Inform. Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[3] ——, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inform. Theory*, vol. 24, no. 5, pp. 530–343, 1978.

[4] C. E. Shannon, "A mathematical theory of communication," *Bell System Tech. J*, vol. 27, pp. 379–423 623–656, 1948.

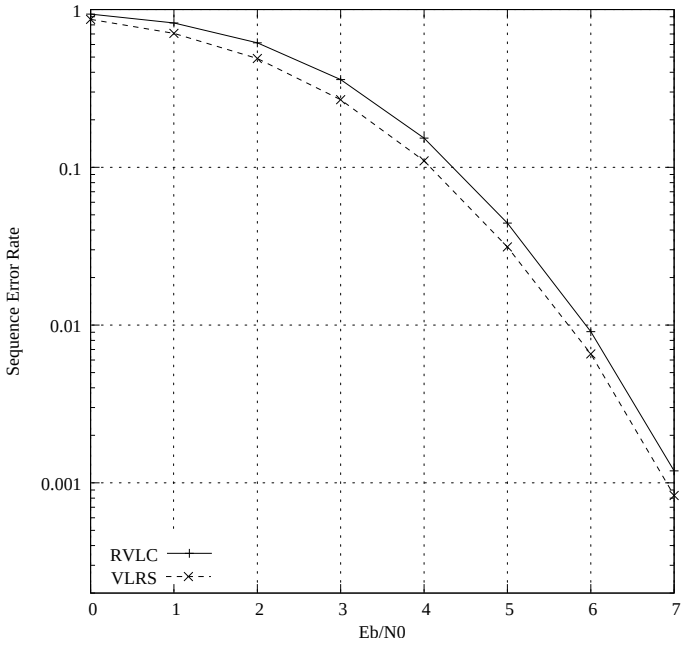

Fig. 6. Soft Decoding of Mirrored VLRS: the unbalanced case.

[5] M. J. Gormish, "Source coding with channel, distortion and complexity constraints," Ph.D. dissertation, Stanford, Mar. 1994.

[6] T. Eriksson, "Trellis source coding methods for low rates and short blocks," Ph.D. dissertation, Lund University, May 2005.

[7] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Trans. Inform. Theory*, vol. 44, pp. 2325–2384, Oct. 1998.

[8] P. G. Howard and J. S. Vitter, "Practical implementations of arithmetic coding," *Image and Text compression*, pp. 85–112, 1992.

[9] E.-H. Yang and D.-K. He, "Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform – part two: with context models," *IEEE Trans. Inform. Theory*, vol. 49, pp. 2874–2894, 11 2003.

[10] D. Huffman, "A method for the construction of minimum redundancy codes," in *Proc. of the Inst. Electr. Radio Eng.*, 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, no. 4, pp. 514–532, 1971.

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

[13] K. P. Subbalakshmi and J. Vaisey, "On the joint source-channel decoding of variable-length encoded sources: The BSC case," *IEEE Trans. Commun.*, vol. 49, no. 12, pp. 2052–2055, Apr. 2001.

[14] S. Lonardi and W. Szpankowski, "Joint source-channel LZ'77 coding," in *Proc. Data Comp. Conf.*, 2003, pp. 273–282.

[15] G.-C. Zhu and F. Alajaji, "Turbo codes for nonuniform memoryless sources over noisy channels," *IEEE Commun. Lett.*, vol. 6, pp. 1253–1262, Feb. 2002.

[16] G.-C. Zhu, F. Alajaji, J. Bajcsy, and P. Mitran, "Transmission of nonuniform memoryless sources via nonsystematic turbo codes," *IEEE Trans. Commun.*, vol. 8, no. 52, pp. 1344–1354, Aug. 2004.

[17] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. John Wiley, 1991.

[18] H. Jégou and C. Guillemot, "Suffix-constrained codes for progressive and robust data compression: self-multiplexed codes," in *Proc. Europ. Sign. Proc. Conf.*, Sept. 2004, vienna.

[19] 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.