Analytical tools for optimizing the error correction performance of arithmetic codes

Salma Ben–Jamaa, Michel Kieffer, Claudio Weidmann

Analytical Tools for Optimizing the Error Correction Performance of Arithmetic Codes

Salma Ben Jamaa*, Claudio Weidmann**, and Michel Kieffer*

* LSS – CNRS – Supélec – Université Paris-Sud, 11,

3 rue Joliot-Curie - 91192 Gif-sur-Yvette cedex, France

** ftw. Telecommunications Research Center Vienna

Donau-City-Strasse 1, A-1220 Vienna, Austria

Abstract

In joint source-channel arithmetic coding (JSCAC) schemes, additional redundancy may be introduced into an arithmetic source code in order to be more robust against transmission errors. The purpose of this work is to provide analytical tools to predict and evaluate the effectiveness of that redundancy. Integer binary Arithmetic Coding (AC) is modeled by a reduced-state automaton in order to obtain a bit-clock trellis describing the encoding process. Considering AC as a trellis code, distance spectra are then derived. In particular, an algorithm to compute the free distance of an arithmetic code is proposed. The obtained code properties allow to compute upper bounds on both bit error and symbol error probabilities and thus provide an objective criterion to analyze the behavior of JSCAC schemes when used on noisy channels. This criterion is then exploited to design efficient error-correcting arithmetic codes. Simulation results highlight the validity of the theoretical error bounds and show that for equivalent rate and complexity, a simple optimization yields JSCACs that outperform classical tandem schemes at low to medium SNR.

Keywords: Arithmetic codes, Source coding, Error correction coding, Communication system performance

I. Introduction

Arithmetic Coding (AC) [1] is currently being deployed in a growing number of compression standards, *e.g.*, H.264 and JPEG2000, as it yields higher compression performance

Parts of this paper have been presented at the IEEE MMSP Workshop, Victoria, Canada, 2006 and at the Joint NEWCOM / ACoRN Workshop, Vienna, Austria, 2006

when compared to other compression methods. However, its efficiency makes AC particularly vulnerable to transmission errors. This issue has motivated the recent development of joint source-channel techniques for AC-encoded data [2]–[7].

Improving the robustness of AC against transmission errors is usually achieved by introducing redundancy in the compressed bitstream. In [2], Boyd *et al.* introduced a *forbidden symbol* (FS) in the source alphabet and used it as an error detection device at the decoder side. The effectiveness of this technique was analyzed by Chou *et al.* in [4], where the FS was used for error detection and an ARQ protocol was implemented for error correction. In [3], Sayir considered the arithmetic encoder as a channel encoder and added redundancy in the transmitted bitstream by introducing gaps in the coding space; he also proposed to use the stack sequential decoding algorithm. In [5], Pettijohn *et al.* used both depth-first and breadth-first sequential decoding, where error detection was again achieved by testing the presence of a FS in the decoded bitstream. Grangetto *et al.* [7] proposed a MAP decoder for AC using the FS. In [8], Demiroglu *et al.* used Trellis Coded Modulation jointly with AC; the FS was exploited to discard erroneous paths during a Viterbi decoding process. In [6], Guionnet *et al.* used a finite state machine (FSM) inspired from [9]–[11] to represent a *quasi-arithmetic* encoder [11], and modeled the transitions between states by a Markov process. Two types of three-dimensional trellises were proposed, using either a symbol clock or a bit clock, and redundancy was added by limiting the number of states and introducing synchronization markers. Another three-dimensional bit-clock trellis for soft decoding of AC was proposed by Dongsheng *et al.* [12].

The comparison between the previously mentioned JSCAC approaches is usually experimental and is restricted to the simulation context. The main purpose of the present paper is to develop analytical tools that allow characterizing and objectively comparing these techniques. Our approach has been inspired, first, by classic results on the error correction properties of convolutional codes, and more generally of linear codes [13], and second, by the extension of these results to Variable-Length Codes (VLC) and Variable-Length Error-correcting Codes (VLEC) [14]. To the best of our knowledge, no similar approach has been carried out for arithmetic codes. In [13]–[15], the codes under study are represented by trellises from which asymptotic characterizations of the decoding error rates are deduced. Here, we consider a practical integer-based implementation of AC for a memoryless source. Then, we develop a specific FSM and trellis representation which is suited to efficient asymptotic error rate evaluation, unlike the trellis representations of [6] and [12], which serve other purposes. The

code distance properties involved in this asymptotic evaluation are then exploited to design efficient error-correcting arithmetic codes.

Section II introduces the basic principles of AC and its integer implementation. Section III explains how AC can be viewed as a FSM, and presents different versions of the derived trellis. Section IV recalls different ways of introducing redundancy proposed in the literature, and explains the trellis-based encoding and decoding process. In Section V, distance properties and error bounds are derived and a practical algorithm allowing to compute the free distance of non-adaptive AC is proposed. These tools are exploited in Section VI to design efficient JSCAC. Simulation results are presented in Section VII, before drawing some conclusions.

II. Integer arithmetic coding

This section recalls the basic principles of binary arithmetic coding. Although this work only deals with binary memoryless sources and binary AC, the derivations and results may be generalized to Markov sources as well as non-binary source alphabets. In the remainder of this paper, *symbols* will stand for the binary inputs of the encoder and *bits* for its outputs.

A. Binary arithmetic coding

Arithmetic coding is based on recursive partitioning of the interval $[0, 1)$ according to the source symbol probabilities. In the case of binary AC, the current *source interval* $[low, high)$ is partitioned into two subintervals I_0 and I_1, the widths of which are proportional to the probabilities P_0 and P_1 of the source symbols 0 and 1, respectively. One of these intervals is selected as the new source interval, according to the value of the current symbol. Once the last symbol is encoded, the encoder computes the *code interval* $[\alpha 2^{-\ell}, (\alpha+1)2^{-\ell}) \subset [low, high)$, such that α is an integer from $\{0, 1, \ldots, 2^{\ell} - 1\}$, and ℓ is the minimum number of bits needed to identify the final interval $[low, high)$. For sources with skewed probabilities and for long source sequences, subintervals may get too small to be accurately handled by a finite-precision computer. This problem is solved by integer binary AC.

B. Integer binary AC

Finite precision arithmetic coding was first introduced by Pasco [9] and Rissanen [10] in 1976. Howard *et al.* proved in [11] that a slight modification of the symbol probabilities such that interval bounds become rational numbers decreases the computational cost of AC without significantly degrading compression performance. Therefore, integer AC works like the ideal

AC presented above, but using the interval $[0, T)$ of integers, where $T = 2^p$, $p \geq 2$ being the bit-size of the initial interval, and rounding all interval boundaries to integers. Partition and selection are carried out every time a source symbol is encoded. Renormalization by doubling the size of the source interval is performed if one of the following conditions holds

1) If $high \leqslant T/2$, low and $high$ are doubled.
2) If $T/2 \leqslant low$, low and $high$ are doubled after subtracting $T/2$.
3) If $T/4 \leqslant low$ and $high \leqslant 3T/4$, low and $high$ are doubled after subtracting $T/2$.

If the current interval (before renormalization) overlaps the midpoint of $[0, T)$, no bit is output. The number of consecutive times this occurs is stored in a variable called $follow$. If the current interval (before renormalization) lies entirely in the upper or lower half of $[0, T)$, the encoder emits the leading bit of low (0 or 1) and $follow$ opposite bits (1 or 0). This is called the *follow-on* procedure [1].

At the decoder side, a sliding buffer of size p is initially formed by the first p bits of the code stream. The interval $[low, high)$ is initialized to $[0, T)$ and then partitioned into I_0 and I_1 according to the source probabilities. Let V be the integer whose binary representation is given by the p bits in the buffer. If $V \in I_0$, the symbol 0 is emitted by the decoder, and if $V \in I_1$, the symbol 1 is decoded. The decoder then performs the same selection and renormalization steps as the encoder. Whenever a renormalization of $[low, high)$ occurs, the next code bit is shifted into the buffer (becoming the new least significant bit) and V is updated accordingly.

III. Trellis-based arithmetic coding

Integer binary AC can be considered as an automaton represented by a finite number of states and transitions, except for the possibly unbounded $follow$ counter. This approach was proposed by Howard *et al.* in [11], where arithmetic operations were replaced by table lookups. In [6], the table-lookup representation was used in order to derive a stochastic automaton which was then concatenated with a convolutional code. Iterative decoding was used at the receiver side. Recently, integer arithmetic decoding was represented by a three-dimensional trellis taking into account the presence of a FS in the source alphabet [12].

A. AC interpreted as a FSM

When performing integer AC, the number of possible subintervals of $[0, T)$ is finite. Considering that after normalization, $range = high - low$ is always greater than $T/4$ and

that $[low, high)$ cannot be a proper subinterval of $[T/4, 3T/4)$, the number of possible intervals $[low, high)$ is $3T^2/16$. This can be shown by considering that the number of possible intervals after normalization is the number of pairs of positive integers (i, j), such that $[T/2 - i, T/2 + j) \subseteq [0, T)$ and $[T/2 - i, T/2 + j) \nsubseteq [T/4, 3T/4)$. The number of such pairs is $T^2/4 - T^2/16 = 3T^2/16$.

For a memoryless source and known symbol probabilities, the encoder is entirely characterized by the current interval $[low, low + range)$ and the value of $follow$. Hence, the encoder state can be represented by $(low, range, follow)$ as defined in [6]. For an order M Markov source, the encoder state has to be extended to include the last M encoded symbols, in order to properly take into account the memory of the source. For the sake of simplicity, only memoryless sources will be considered in the following.

The idea is thus to precompute all possible states of the arithmetic encoder such that any source stream may be encoded using table lookups rather than arithmetic operations. However, as the variable $follow$ might grow without bound, the number of states $(low, high, follow)$ could be infinite. In [6], $follow$ was considered as an output variable and only the update of this variable was taken into account in the lookup table. In this work, the value of $follow$ is a part of the encoder state.

In order to cope with the uncontrolled growth of $follow$, we choose to keep it bounded by a given threshold $F_{\max}$ as in [12]. To this end, renormalizations incrementing $follow$ are only performed as long as $follow < F_{\max}$. Whenever $follow = F_{\max}$ and the current source interval is such that $follow$ could be further incremented, the symbol probabilities are modified in order to force a *follow-on* procedure after encoding the current symbol. Let $[low_f, high_f)$ be the current interval, such that $T/4 \leqslant low_f < T/2$ and $T/2 < high_f \leqslant 3T/4$. This interval is subdivided into $[low_f, T/2)$ and $[T/2, high_f)$. The first subinterval is selected if the next symbol is 0, the second is selected if the next symbol is 1.

In the remainder of this section, three FSMs describing the AC operations with a bound on $follow$ are proposed, namely a symbol-clock FSM, which is well suited for encoding, a reduced FSM leading to a compact trellis better suited for decoding, and a bit-clock FSM suited for distance evaluation.

B. Symbol-clock FSM

The first proposed FSM is such that each transition corresponds to a single encoded symbol; hence it is called symbol-clock FSM. Starting at the initial state $(0, T, 0)$, the encoder is fed

by the two possible input symbols, and may either reach a new state or return to $(0, T, 0)$. This defines the two starting transitions of the FSM describing the AC. Every arrival state is then considered as a starting state of two new transitions leading either to new states or to already known ones. This exhaustive search stops when no new states are found. The set of states and the set of transitions between states of this *symbol-clock* FSM are denoted by $\mathcal{S}^{\text{s}}_{\text{FSM}}$ and $\mathcal{T}^{\text{s}}_{\text{FSM}}$, respectively.

Figure 1 shows the symbol-clock FSM representing an integer AC with $p = 4$, $P_0 = 0.2$ and $F_{\max} = 1$. The transitions are labeled with input symbol / output bits. When no bits are output by the encoder, the transition label is $1/-$ or $0/-$. These are called *mute* transitions.

C. Reduced FSM

When dealing with noisy transmissions, a trellis-based soft-input decoder may be implemented as in [6], [12]. In this work, we use a Viterbi decoder which keeps only the best path among all those converging in the same state at a given depth of the trellis. The saved path is called *survivor* (see Section IV). If the decoder uses a likelihood-based metric to compare converging paths, the associated bit sequences must be of equal length. Therefore, the symbol-clock trellis derived from the FSM described in Section III-B will not be appropriate for such a decoder, as merging paths would have equal length in symbols, but variable length in bits. A bit-clock trellis is needed instead.

One possible representation of a bit-clock trellis is the three-dimensional trellis proposed in [12], where the third dimension keeps track of mute transitions. Running a Viterbi Algorithm (VA) on such a trellis requires processing the states in a particular order. For example, if there is a mute transition from state x to state y, the survivor at state x has to be computed *before* the one at y, otherwise the decoder becomes suboptimal.

The trellis adopted in this work is derived from the FSM proposed in Section III-B, which is modified in order to have no mute transitions. This allows not only to have a two- instead of three-dimensional trellis, but also to reduce the number of states and thus to reduce the memory required by the VA, which will work without any constraint on the evaluation order of survivors. To obtain a *reduced* FSM, we allow transitions to have more than one input symbol. In fact, every mute transition is *extended* until at least one bit is output. The transition extension algorithm is explained below.

Let $\mathcal{T}_{xy}$ be the set of transitions from state x to state y of a FSM. The input *symbols* associated to $t_{xy} \in \mathcal{T}_{xy}$ are denoted by $in(t_{xy})$ and its output *bits* by $out(t_{xy})$. If t_{xy} is extended

with a transition t_{yz}, the resulting transition $t_{xz} = t_{xy} \circ t_{yz}$ has $in(t_{xz}) = in(t_{xy}) \circ in(t_{yz})$ and $out(t_{xz}) = out(t_{xy}) \circ out(t_{yz})$, respectively ($\circ$ denotes concatenation).

Let $\mathcal{T}_{mute}$ be the subset of $\mathcal{T}^{\text{s}}_{\text{FSM}}$ containing all the mute transitions. Algorithm 1 describes how the mute transitions may be removed from a FSM to obtain an equivalent reduced FSM, whose sets of states and transitions will be denoted by $\mathcal{S}_{\text{FSM}}$ and $\mathcal{T}_{\text{FSM}}$, respectively.

Algorithm 1 (Removing the mute transitions from the FSM):

0. Find all mute transitions in $\mathcal{T}^{\text{s}}_{\text{FSM}}$. Save them in $\mathcal{T}_{mute}$
$\mathcal{T}_{\text{FSM}} = \mathcal{T}^{\text{s}}_{\text{FSM}}$; $\mathcal{S}_{\text{FSM}} = \mathcal{S}^{\text{s}}_{\text{FSM}}$

1 **For** all $t_{xy} \in \mathcal{T}_{mute}$:

- **a** **For** all $t_{yz} \in \mathcal{T}_{\text{FSM}}$:
  - **a.1** Create a new transition $t_{xz} = t_{xy} \circ t_{yz}$.
  - **a.2** **If** t_{xz} is mute, add it to $\mathcal{T}_{mute}$, **else** add it to $\mathcal{T}_{\text{FSM}}$.
- **b** Delete t_{xy} from $\mathcal{T}_{mute}$ and from $\mathcal{T}_{\text{FSM}}$.

2 **If** $\mathcal{T}_{mute}$ is not empty, go to **1**.

3 Remove all states having no incoming transitions from $\mathcal{S}_{\text{FSM}}$.

If a state has a single incoming mute transition, after extension of that transition it will have no incoming connections and can therefore be removed from the set of states (see, *e.g.*, state 5 in Figure 1). Hence, this procedure allows to reduce the number of states in the FSM. Moreover, in some cases $\mathcal{S}_{\text{FSM}}$ may contain states with a single incoming transition or a single outgoing transition. Algorithm 1 may also be applied in order to remove such states from the FSM. The reduced FSM obtained from the symbol-clock FSM of Figure 1 is depicted in Figure 2.

The trellis resulting from the reduced FSM will have transitions with variable-length inputs and outputs, and it may be represented both as a bit-clock or as a symbol-clock trellis. The bit-clock trellis generated from the reduced FSM of Figure 2 is represented in Figure 3. Every path starting at *depth* 0 and reaching a given depth n is associated to a code sequence of length n bits and a variable-length source sequence. The encoder and the decoder in this work have been implemented with this type of trellis (see Section IV).

D. Bit-clock FSM

As will become clear in Section V-A, an efficient recursive evaluation of the distance properties of trellis-based AC is possible if transitions have exactly one output bit. Such a

trellis may be obtained from the reduced FSM of Section III-C by introducing additional *intermediate states*, such that every state transition outputs exactly one bit. When a given transition outputs two bits, it is split into two transitions, the first of which inherits the input symbols and the first output bit, while the second has no input symbol and outputs the second bit. Introducing these states does not increase the total number of paths through the trellis, since the intermediate states have a single incoming transition and a single outgoing transition. The set of states in the bit-clock FSM is denoted by $\mathcal{S}_{\text{FSM}}^{\text{b}}$, and the set of transitions by $\mathcal{T}_{\text{FSM}}^{\text{b}}$. Notice that a bit-clock FSM could also be obtained from the AC decoding automaton, although it would not necessarily be the same as the one derived here, due to the prior removal of states with a single incoming transition. In any case, the analysis in Section V applies unchanged.

IV. Joint source-channel arithmetic codes

In joint source-channel coding schemes, redundancy is introduced in order to allow error detection and/or correction at the decoder side. According to [11], when considering integer probability spaces in $[0, T)$, the additional redundancy due to the integer approximation is at most $0.497/T + O(1/T^2)$ bits/symbol if correct probability estimates are used by the encoder. Limiting the value of $follow$ may be another source of redundancy. Nevertheless, as the probability of having $follow = F_{\max}$ decreases exponentially with $F_{\max}$, this additional redundancy remains small, see Table I in Section VII.

Section I mentioned that a well-known JSC technique for AC is based on the introduction of a forbidden symbol (FS) in the source alphabet. The FS is never emitted by the source, although a positive probability P_ε is assigned to it. In that way, decoding the FS indicates the occurrence of a transmission error. It can be shown that introducing a FS of probability P_ε adds a redundancy of $-\log(1 - P_\varepsilon)$ bits/symbol to the coded bitstream [4]. All the cited techniques for adding redundancy may be applied to integer AC jointly with a bound on $follow$. The resulting operations will thus be described by a FSM. Consequently, JSCAC operations may entirely rely on the trellis representation of AC.

The notation for the transmission scheme under study is as follows. The sequence of binary source symbols is denoted by $\mathbf{u}_{1:K}$. After encoding, one gets a sequence of bits $\mathbf{b}_{1:N}$, which after modulation becomes $\mathbf{m}_{1:N}$. The sequences of channel outputs and decoded symbols are denoted by $\mathbf{y}_{1:N}$ and $\widehat{\mathbf{u}}_{1:\widehat{K}}$, respectively. Encoded bits are assumed to be mapped onto symmetric BPSK signals $m_i = m(b_i) = (-1)^{b_i}\sqrt{E_{\text{b}}}$ and sent over an AWGN channel.

Various estimation criteria may be used by the decoder. Here, we use a ML Viterbi algorithm (VA) to maximize the likelihood $P(\mathbf{y}_{1:N}|\mathbf{b}_{1:N})$ such that the best estimate of the symbol sequence is $\widehat{\mathbf{u}}_{1:\widehat{K}}$ associated to $\widehat{\mathbf{b}}_{1:N}$, expressed by

$$\begin{aligned}\widehat{\mathbf{b}}_{1:N} &= \arg\max_{\mathbf{b}_{1:N}} P(\mathbf{y}_{1:N}|\mathbf{b}_{1:N}) \\ &= \arg\min_{\mathbf{b}_{1:N}} \sum_{i=1}^{N} (m(b_i) - y_i)^2. \end{aligned} \quad (1)$$

where N is supposed to be known. As consequence of this latter assumption, the decoder has to search $\widehat{\mathbf{b}}_{1:N}$ among a set of equal-length code sequences associated to variable-length source sequences. The soft-input decoder is thus implemented using the bit-clock trellis derived from the reduced FSM described in Section III-C.

Encoding and decoding using this trellis are now described. The encoder always starts from $(0, T, 0)$. From any given state, the set of input sequences labeling the transitions to the next states is prefix-free, thus the encoding process is instantaneous. However, decoding is not instantaneous, since the set of output bit sequences of a state is not prefix-free in general. Nevertheless the VA can be applied on the reduced bit-clock trellis, as it will only compare equal-length paths. Since the cost function to minimize in (1) is additive, at every trellis depth n, the VA evaluates the best estimate $\widehat{\mathbf{b}}_{1:n}$ among all paths of length n reaching a given state. Unlike the trellis proposed in [12], which needs a third dimension to keep track of mute transitions, here the constructed two-dimensional trellis is sufficient to run the VA.

When encoding the last symbols of the source sequence, the encoder may have the choice between several transitions beginning with these symbols and thus possibly several final states. There are different strategies to cope with this termination problem in practical AC. For our analysis and simulations, we will simply assume that the source sequence corresponds to an integer number of FSM transitions, since for long sequences this will have a negligible impact on the decoder error rate. The terminating state is assumed unknown to the decoder.

In JSCAC schemes, the effects of the additional redundancy in terms of error detection and correction capability appear in the trellis representation of the associated FSM. The effectiveness of this redundancy may be characterized by the distance properties derived from the trellis, as explained in the next section.

V. Performance analysis of trellis-based AC

As shown in Section III, the FSM interpretation naturally leads to consider the integer arithmetic code as a trellis code. The performance of trellis codes on channels with moderate

to large signal-to-noise ratios (SNR) can be predicted from their distance properties [13].

A. Free distance

At the decoder side, when a trellis-based VA is used, any two distinct equal-length paths that start at a common state and end in a common state may be confused by the decoder due to channel noise. The closest paths in terms of decoding metric determine the worst case for error detection; the distance between them is called the free distance d_{free}. Thus d_{free} is defined as the minimum Hamming distance between all paths of the same length in bits diverging at some state and converging in the same or another state. If the encoded messages are long enough to avoid boundary effects, d_{free} plays a similar role as the minimum distance for a block code, in that all error patterns of weight t can be corrected if $d_{\text{free}} > 2t$.

For convolutional codes, which are linear codes, d_{free} is equal to the smallest Hamming *weight* over all paths diverging at and then converging in the all-zero path [13]. In the case of non-linear codes (such as VLEC and arithmetic codes), comparing paths to the all-zero path may not be sufficient to determine d_{free}, since such codes are generally not geometrically uniform [16]. In [14], Buttigieg deduced a lower bound on d_{free} from the distance between all possible pairs of unequal-length VLEC codewords. Extending this technique to AC is not possible, as the transition outputs do not satisfy the prefix condition in general (see Figure 3), reducing Buttigieg's lower bound to zero. Therefore, we propose an algorithm for computing d_{free} for trellis-based AC. This algorithm relies on an iterative computation of the smallest distances between all different paths of equal length starting from a common state. Its iterative structure allows an evaluation of d_{free} with polynomial complexity in the number of states of $\mathcal{S}^{\text{b}}_{\text{FSM}}$ and in the number of transitions between these states.

The iterations for computing d_{free} will be performed on the length in bits of the paths on the trellis, it is thus advantageous to use the bit-clock trellis of Section III-D. Before presenting the algorithm for computing $d_{\text{free}},$ some notations have to be introduced.

A path of n bits on a bit-clock trellis (see Figure 3), starting from state x and ending in y is denoted by $\mathbf{p}^n_{xy}$. The Hamming distance between the output bits of two paths $\mathbf{p}^n$ and $\mathbf{q}^n$ of the same output length n is denoted by $d^{\text{out}}_{\text{H}}(\mathbf{p}^n, \mathbf{q}^n)$. When a path $\mathbf{p}^n_{xy}$ is extended by a transition t_{yz}, one obtains a new path $\mathbf{p}^{n'}_{xz} = \mathbf{p}^n_{xy} \circ t_{yz}$ of length $n' = n + \ell(out(t_{yz}))$, ℓ being the length function. The set of all *pairs* of paths diverging at the same starting state x and *converging* for the first time n bits later in state y is denoted by $\mathcal{C}_n(x, y)$. The set of all *pairs* of paths of length n bits, *diverging* at the starting state x and never converging is

$\mathcal{D}_n(x)$. Finally, let $\mathcal{T}^{\mathrm{b}}_{xy} \subset \mathcal{T}^{\mathrm{b}}_{\mathrm{FSM}}$ be the set of transitions starting from x and ending in y in the bit-clock FSM.

To evaluate d_{free}, we build a three-dimensional array Δ_n defined as follows: for $y \neq z$, $\Delta_n(x,y,z)$ is the minimum distance between all pairs of paths of length n, starting from x, ending respectively in y and z, and never converging. For $z = y$, $\Delta_n(x,y,y)$ is defined as the minimum distance between pairs of paths of *at most* n bits, diverging at state x and converging for the first time in state y. Using these notations, one may write

$$\Delta_n(x,y,z) = \min_{(\mathbf{p}^n_{xy},\mathbf{q}^n_{xz})\in\mathcal{D}_n(x)} d^{\mathrm{out}}_{\mathrm{H}}(\mathbf{p}^n_{xy},\mathbf{q}^n_{xz}), \text{ if } y \neq z \tag{2}$$

$$\Delta_n(x,y,y) = \min_{n'\leqslant n} \min_{(\mathbf{p}^{n'}_{xy},\mathbf{q}^{n'}_{xy})\in\mathcal{C}_{n'}(x,y)} d^{\mathrm{out}}_{\mathrm{H}}(\mathbf{p}^{n'}_{xy},\mathbf{q}^{n'}_{xy}), \tag{3}$$

and

$$d_{\mathrm{free}} = \min_{n,x,y} \Delta_n(x,y,y). \tag{4}$$

The evaluation of $\Delta_n(x,y,z)$ seems to require the evaluation of distances between an exponentially growing number of paths pairs. The following proposition provides an iterative technique for the evaluation of $\Delta_n(x,y,z)$ with polynomial complexity.

Proposition 1: For any initial state $x \in \mathcal{S}_{\mathrm{FSM}}$, (2) and (3) can be evaluated recursively starting from

$$\Delta_1(x,y,z) = \min_{\substack{t_{xy}\in\mathcal{T}^{\mathrm{b}}_{xy}, t_{xz}\in\mathcal{T}^{\mathrm{b}}_{xz} \\ t_{xy}\neq t_{xz}}} d^{\mathrm{out}}_{\mathrm{H}}(t_{xy},t_{xz}), \tag{5}$$

with $y,z \in \mathcal{S}^{\mathrm{b}}_{\mathrm{FSM}}$, using

$$\Delta_n(x,y,z) = \min_{y'\neq z'} \min_{t_{y'y}\in\mathcal{T}^{\mathrm{b}}_{y'y}, t_{z'z}\in\mathcal{T}^{\mathrm{b}}_{z'z}} \left\{\Delta_{n-1}(x,y',z') + d^{\mathrm{out}}_{\mathrm{H}}(t_{y'y},t_{z'z})\right\}, \text{ if } y \neq z \tag{6}$$

$$\Delta_n(x,y,y) = \min\left\{\Delta_{n-1}(x,y,y), \min_{y'\neq z'} \min_{t_{y'y}\in\mathcal{T}^{\mathrm{b}}_{y'y}, t_{z'y}\in\mathcal{T}^{\mathrm{b}}_{z'y}} \left(\Delta_{n-1}(x,y',z') + d^{\mathrm{out}}_{\mathrm{H}}(t_{y'y},t_{z'y})\right)\right\} \tag{7}$$

where the minimization is intended over all distinct $y', z' \in \mathcal{S}^{\mathrm{b}}_{\mathrm{FSM}}$. $\diamondsuit$

By convention, if one of the sets $\mathcal{T}^{\mathrm{b}}_{xy}$ or $\mathcal{T}^{\mathrm{b}}_{xz}$ is empty, then $d^{\mathrm{out}}_{\mathrm{H}}(t_{xy},t_{xz}) = +\infty$. Note that x can only be a state of the original reduced FSM ($x \in \mathcal{S}_{\mathrm{FSM}} \subset \mathcal{S}^{\mathrm{b}}_{\mathrm{FSM}}$), since intermediate states in $\mathcal{S}^{\mathrm{b}}_{\mathrm{FSM}}$ have a single outgoing transition and thus no pair of paths can diverge at such a state.

The proof of Proposition 1 may be found in Appendix A.

Corollary 1: The complexity for evaluating any $\Delta_n(x,y,z)$ is $O\left(M^2_{\mathrm{T}} \cdot \left|\mathcal{S}^{\mathrm{b}}_{\mathrm{FSM}}\right|^2\right)$, with $M_{\mathrm{T}} = \max_{y,z\in\mathcal{S}^{\mathrm{b}}_{\mathrm{FSM}}} \left|\mathcal{T}^{\mathrm{b}}_{yz}\right|$. $\diamondsuit$

Proof: $\Delta_n(x,y,z)$ is evaluated from $\Delta_n(x,y',z')$, which is a matrix of $\left|\mathcal{S}_{\text{FSM}}^{\text{b}}\right|^2$ entries. From (6) and the definition of M_{T}, the number of operations to evaluate $\Delta_n(x,y,z)$ is then $O\left(M_{\text{T}}^2 \cdot \left|\mathcal{S}_{\text{FSM}}^{\text{b}}\right|^2\right)$. $\diamondsuit$

For a given $x \in \mathcal{S}_{\text{FSM}}$, assume that there exists some N_x such that

$$\min_{y',z' \in \mathcal{S}_{\text{FSM}}^{\text{b}}} \Delta_{N_x}(x,y',z') \geqslant \min_{y \in \mathcal{S}_{\text{FSM}}} \Delta_{N_x}(x,y,y). \tag{8}$$

The recursion (7) can then be stopped, since it will be no longer possible to reduce $\Delta_n(x,y,y)$ by merging pairs of paths ending in states y' and z', respectively and (8) will be satisfied for all $n \geqslant N_x$. Moreover, let $\overline{N} = \max_{x \in \mathcal{S}_{\text{FSM}}} N_x$. Then d_{free} may be evaluated as

$$d_{\text{free}} = \min_{n \leqslant \overline{N}, x, y} \Delta_n(x,y,y).$$

If the AC is non-catastrophic[1], $\overline{N}$ will be finite and Corollary 1 implies that also the evaluation of d_{free} is of polynomial complexity.

B. Distance spectra and error bounds

While the free distance dominates the error correction capability of a code, a finer error analysis is possible by using the *distance spectrum* to evaluate an upper bound on error probability. The distance spectrum of a code is the sequence $\{A_d\}$ that counts the average number of paths at distance $d \geqslant d_{\text{free}}$ from the correct path. For a convolutional code, this is identical to its weight spectrum (*i.e.*, A_d is the number of paths of weight d for $d \geqslant d_{\text{free}}$), due to linearity.

Let P_{er}^{b} be the bit error probability at any position in the code. Then the following union upper bound holds [13, Chap. 11]

$$P_{\text{er}}^{\text{b}} \leqslant \sum_{d=d_{\text{free}}}^{\infty} dA_d P_d, \tag{9}$$

where P_d is the probability that the decoder selects an erroneous path at Hamming distance d instead of the correct path. For BPSK signaling used over an AWGN channel,

$$P_d = \frac{1}{2} \text{ erfc } \sqrt{d\frac{E_b}{N_0}}, \tag{10}$$

where $N_0/2$ is the variance of the zero-mean Gaussian channel noise [17].

[1]An encoder is catastrophic if there exist two semi-infinite input sequences differing in *infinitely* many positions that are encoded into two output sequences differing only in a *finite* number of positions.

The bound (9) characterizes the decoder bit error probability in the code domain. In practice, an equally important figure of merit is the symbol error probability in the information domain. For convolutional encoders, this probability can be bounded using another spectrum, $\{B_d\}$ that counts the average number of nonzero information bits on paths of weight d. Then P_{er}^{s}, the symbol error probability at any source symbol position, is bounded by

$$P_{\text{er}}^{\text{s}} \leqslant \sum_{d=d_{\text{free}}}^{\infty} B_d P_d. \tag{11}$$

The two distance spectra $\{A_d\}$ and $\{B_d\}$ and the resulting bounds (9) and (11) may be extended to a (non-linear) time-invariant trellis code driven by a memoryless source. Buttigieg [14], [15] carried this out for non-linear VLEC trellises. The most important difference to convolutional codes is that it is no longer sufficient to consider path weights alone, hence A_d has to be defined as the average number of converging path pairs at Hamming distance d, which can be computed assuming a stationary probability distribution on the trellis. Another difficulty arises from the fact that although the decoder compares converging code paths of equal length, these may be associated to source sequences of different lengths. Therefore the spectrum $\{B_d\}$ must be defined in terms of a distance measure that allows comparing unequal-length sequences; the most common choice is the Levenshtein distance d_{L} [18], which is defined as the minimum number of deletions, insertions, or substitutions required to transform one sequence into the other. Hence B_d is defined as the average Levenshtein distance between the input sequences of all converging pairs of paths whose output sequences are at Hamming distance d.

Extending the results for VLEC trellises to AC trellises is straightforward, the proofs for the bounds (9) and (11) follow along the same lines as in [14], [15], so that the only major difference is the computation of the distance spectra. Define $d_{\text{L}}^{\text{in}}(\mathbf{p},\mathbf{q}) = d_{\text{L}}(in(\mathbf{p}), in(\mathbf{q}))$, where $\mathbf{p},\mathbf{q}$ are paths formed by concatenating one or more FSM transitions. The spectral components A_d and B_d may then be expressed as

$$A_d = \sum_{n=d}^{\infty} \sum_{\substack{(\mathbf{p}^n,\mathbf{q}^n)\in\mathcal{C}_n:\\ d_{\text{H}}^{\text{out}}(\mathbf{p}^n,\mathbf{q}^n)=d}} P(\mathbf{p}^n), \tag{12}$$

$$B_d = \sum_{n=d}^{\infty} \sum_{\substack{(\mathbf{p}^n,\mathbf{q}^n)\in\mathcal{C}_n:\\ d_{\text{H}}^{\text{out}}(\mathbf{p}^n,\mathbf{q}^n)=d}} P(\mathbf{p}^n)\; d_{\text{L}}^{\text{in}}\left(\mathbf{p}^n,\mathbf{q}^n\right), \tag{13}$$

where $P(\mathbf{p}^n)$ denotes the *a priori* probability of the path $\mathbf{p}^n$ of output length n bits, and $\mathcal{C}_n$

is the set of all path pairs on the reduced FSM trellis diverging at the same starting state and converging for the first time n bits later (thus $\mathcal{C}_n = \bigcup_{x,y \in \mathcal{S}_{\text{FSM}}} \mathcal{C}_n(x,y)$).

For the evaluation of $\{A_d\}$ and $\{B_d\}$ for VLECs in [14], only paths beginning at a single initial state (corresponding to depth $n = 0$) had to be considered. In our case, two paths may diverge at any state of the reduced trellis. Assuming that the encoder has already encoded a large number of symbols from a memoryless source before the first error event occurs, the stationary probability $P(x)$ of being in a given state x can be evaluated using the *Markov transition matrix* Π of the reduced FSM. $\Pi(x,y)$ is the probability that the next state will be y, knowing that the current one is x:

$$\Pi(x,y) = \sum_{t_{xy} \in \mathcal{T}_{xy}} P(t_{xy}|x) = \sum_{t_{xy} \in \mathcal{T}_{xy}} P(in(t_{xy})). \tag{14}$$

Then $P(x)$ is the stationary distribution of the Markov chain defined by Π, if the FSM is ergodic (*i.e.*, a forward path with positive probability exists between any two states). Now the *a priori* probability of a given path $P(\mathbf{p}_{xy}^n)$ starting in x and ending in y, which is needed in (12) and (13), can be computed using $P(x)$ and the *a priori* source probabilities, yielding

$$P(\mathbf{p}_{xy}^n) = P(x) P\left(in\left(\mathbf{p}_{xy}^n\right)\right). \tag{15}$$

The evaluation of distance spectra is carried out by an exhaustive enumeration of the path pairs in $\mathcal{C}_n$. This may be done using one of the two algorithms proposed in [14]. Both algorithms perform an exhaustive search over all paths on the trellis. The first algorithm is fast, but requires a lot of memory, as all paths that have not converged are stored and compared. The search is stopped when some pre-determined depth is reached. In the second algorithm, the required memory is limited, but the computations are slow. In that algorithm, only two paths are stored at a given time, but the same paths are repeatedly constructed, stored and erased. In this work, as in Buttigieg's first algorithm, an exhaustive search is performed and all paths that have not converged are stored. To reduce memory requirements and increase speed, we limit the value of d for which A_d and B_d are computed, since the first few non-zero spectral components (with small d) dominate in the error bounds. In practice, also the depth n will be limited, so only lower bounds on A_d and B_d are obtained. We observed that this is more problematic with low-redundancy codes, which tend to have a huge number of short paths at $d_{\text{free}} = 1$. For higher redundancy, the first spectral components quickly approach their true value with growing n.

A polynomial-time evaluation of A_d is possible using an iterative matrix-based algorithm not presented here due to lack of space. Also that algorithm evaluates a lower bound $A_d^{n_{\max}} \leq A_d$ by considering only path pairs of less than $n_{\max}$ bits. An upper bound on the approximation error could likely be obtained in similar fashion as for convolutional codes [19].

C. Extension to context-adaptive AC

Many practical AC schemes are adaptive and context-based. The extension of the presented approach to an adaptive scheme is possible: this would require a FSM with more more states, and thus larger trellises. The situation is more complicated for context-based arithmetic codes. Taking into account contexts, which may depend on previously decoded data, would result in an unmanageable increase of the complexity of the encoder FSM. Nevertheless, if the context is provided as an external variable, context-based arithmetic codes may be represented with several FSM working in parallel, the selection of the active FSM being determined by the context. Under that hypothesis, an extension of the proposed analysis techniques appears possible, provided that the number of contexts remains small.

VI. Design of robust AC using distance properties

This section presents a design method for robust JSC arithmetic codes based on optimizing the distance properties of integer binary AC with a forbidden symbol (FSAC). As mentioned in Section IV, introducing a FS of probability P_ε adds $-\log(1-P_\varepsilon)$ bits/symbol of redundancy to the code bitstream. This holds for high-precision AC (with $p \gg 1$); in the more interesting case of low-precision AC with few states, despite a constant P_ε the actual redundancy will vary considerably with the placement of the FS, due to rounding effects. In order to account for this changing code rate, we chose to ignore spectral efficiency and compare codes at equal SNR per *source* symbol, $E_\mathrm{s}/N_0 = rE_\mathrm{b}/N_0$, where r is the code rate in bits/symbol. Inserting this into (11) through (10), one sees that for large SNR the bound is dominated by the first term, which can be approximated as $B_{d_\mathrm{free}} \exp(-\frac{d_\mathrm{free}}{r}\frac{E_\mathrm{s}}{N_0})$. Therefore we say that a code is *asymptotically better* if it has larger d_free/r; in case of equality, the code with smaller B_{d_free} will be better. This establishes a criterion for optimizing FSAC codes based on their distance spectrum.

At each iteration in binary FSAC, the current interval is split into three parts: I_0 and I_1, for the source symbols 0 and 1, respectively, and I_FS for the FS. Note that I_FS may be composed of disjoint subintervals (alternatively, these subintervals might be seen as corresponding to

distinct forbidden symbols). This is not the case for I_0 and I_1, which have to be connected intervals. In [3], it is shown that by adapting the configuration of I_0, I_1, and I_{FS}, one may in principle implement any block or convolutional code with an arithmetic code.

Here, the following simple FS configuration scheme is considered. Let s_{FS} be the current total size of the intervals allocated to the FS, and $s_0 = (1-P_\varepsilon)P_0 \times range$ the range allocated to the source symbol 0. Then I_{FS} may be written as

$$\begin{aligned} I_{\text{FS}} &= [low,\ low + \lfloor q_1 s_{\text{FS}} \rfloor) \\ &\cup [low + \lfloor q_1 s_{\text{FS}} \rfloor + s_0,\ low + \lfloor q_1 s_{\text{FS}} \rfloor + s_0 + \lfloor q_3 s_{\text{FS}} \rfloor) \\ &\cup [high - \lfloor q_2 s_{\text{FS}} \rfloor,\ high), \end{aligned} \tag{16}$$

where q_1, q_2 and q_3 are such that $q_1 + q_2 + q_3 = 1$, and $\lfloor \cdot \rfloor$ stands for rounding towards $-\infty$. Thus, for given P_ε only two parameters q_1 and q_2 have to be adjusted in order to find the best FS configuration according to the previously defined criterion. As neither d_{free}/r nor $B_{d_{\text{free}}}$ are smooth functions of (q_1, q_2), the optimization of q_1 and q_2 is performed on a grid with resolution δ, satisfying the constraints $q_1 \geqslant 0$, $q_2 \geqslant 0$, and $q_1 + q_2 \leqslant 1$. Since the maximum value of s_{FS} is $s_{\text{FS}}^{\max} = P_\varepsilon \times 2^p$, there is no need to consider values of δ less than $1/s_{\text{FS}}^{\max}$.

VII. Experimental results and discussion

The encoder used for testing the error correction performance is characterized by four parameters: p, the bit size of the initial interval $[0, T)$, P_0, the probability of source symbol 0, $F_{\max}$, the chosen upper limit of $follow$, and P_ε, the probability assigned to the FS.

For a binary memoryless source with $P_0 = 1/8$, Table I shows the additional redundancy due to the limitation of $follow$, computed as $\left(\overline{\ell} - h\left(P_0\right)\right)/h\left(P_0\right)$ (in percent), where $\overline{\ell}$ is the average number of bits per symbol in the code bitstream, and $h\left(P_0\right) = -P_0 \log_2 P_0 - (1-P_0)\log_2(1-P_0)$ is the entropy of the source. Similar behavior occurred for other values of P_0. We also observed that the redundancy due to $F_{\max}$ tends to get larger with p (likely because higher-precision AC has more interval configurations leading to increment $follow$). Table I also compares the number of states $|\mathcal{S}_{\text{FSM}}^{\text{s}}|$ of the symbol-clock FSM described in Section III-B with the number of states $|\mathcal{S}_{\text{FSM}}|$ of the reduced FSM of Section III-C. Using the reduced FSM significantly reduces the number of states without loss of compression performance, except for a slightly longer termination.

Figure 4 compares the distance-spectra upper bounds[2] on BER and SER to simulations obtained for different values of P_ε, and a fixed configuration of the intervals allocated to the FS: $[low, low + range \cdot \frac{P_\varepsilon}{2}) \cup [high - range \cdot \frac{P_\varepsilon}{2}, high)$. Simulation results have been obtained using source sequences of 1024 symbols, and a ML Viterbi decoder fed with the noisy modulated bits. The SER is measured with the average Levenshtein distance between the source sequence and the decoder output. As expected, the bounds become tighter at medium to high SNR. In particular, the asymptotic behavior of the bounds allows to predict and compare the code performance at high SNR without needing any simulation.

In the next set of experiments, we use the spectrum to optimize FSAC as explained in Section VI. Table II shows the properties of encoders designed for different FS probabilities. For each value of P_ε, we compute the pairs $(q_1, q_2)_{\mathrm{b}}$ and $(q_1, q_2)_{\mathrm{w}}$ leading to the best and to the worst bounds, respectively (worst performance is obtained by minimizing d_{free}/r and then maximizing $B_{d_{\mathrm{free}}}$). The binary source has $P_0 = 0.2$, the encoder parameters are $p = 5$, $F_{\max} = 2$ and the search step size for (q_1, q_2) is 0.05. At low redundancy, all codes have $d_{\mathrm{free}} = 1$ and thus the one with the lowest rate will be chosen as asymptotically optimal. If performance at medium SNR is also important, one could use a mixed (Lagrangian) criterion to jointly minimize the rate and $B_{d_{\mathrm{free}}}$. Table II also shows that the simple approach of splitting the FS in fixed proportions is not sufficient to obtain strong codes. Increasing the design freedom, *e.g.*, by adapting characteristics of the forbidden intervals with the state of the AC may improve its error correction performance.

Figure 5 compares the SER obtained with the best and the worst configurations of the FS subintervals. For $P_\varepsilon = 0.1$ (small redundancy), only a few tenths of dB of coding gain can be obtained; the asymptotic decay is almost the same for both codes. The picture changes for $P_\varepsilon = 0.5$ (large redundancy), where the gain is already 2 dB at SER 10^{-4} and rapidly increasing with SNR, due to the difference in asymptotic decay. The simulations for $P_\varepsilon = 0.5$ were carried out with source blocks of 5000 symbols, in order to reduce the incidence of errors towards the end of the block. This evidences the necessity of a proper termination strategy for FSAC to maintain its error correction capability over short blocks (by design, the distance-spectra bounds hold only for semi-infinite sequences, not for short blocks).

The last series of experiments compare a FSAC joint source-channel scheme to a classical tandem scheme relying on the concatenation of a source and a channel encoder. FSAC was

[2]Actually, the plotted bounds are approximations computed with up to 10^6 pairs of paths of length $n \leq 30$ bits.

most often proposed as a low-redundancy error detection device for use in conjunction with an ARQ protocol [4]. Here we show that such low amounts of redundancy may even yield better error correction performance than a tandem system at low to medium SNR.

The reference system consists of a binary source with $P_0 = 0.1$ encoded with a high-precision arithmetic encoder with $p = 16$ and $F_{\max} = 2^{31}$, serially concatenated with a punctured convolutional code (CC) of rate $7/8$, which is truncated (not terminated). Three different CCs will be considered, with constraint lengths $c_l = 3$, 4 and 5 (thus 4, 8 and 16 states), generators $(5, 7)_o$, $(15, 17)_o$ and $(23, 35)_o$, puncturing matrices $[1011111; 1100000]$, $[1000010; 1111101]$ and $[1010011; 1101100]$, and free distances $d_{\text{free}} = 2$, 2, and 3, respectively. These CCs have the best free distance for the given rate and constraint lengths [20]. For blocks of 2048 source symbols, the total rate is 0.545 bits/symbol, including overhead from AC termination and puncturing. The CC is decoded with a soft-input, hard-output ML Viterbi decoder and fed into a standard arithmetic decoder. SER is measured in Levenshtein distance.

The joint system is a FSAC with $p = 5$, $F_{\max} = 2$, $P_\varepsilon = 0.16$, $q_1 = 0.25$, $q_2 = 0.45$, $d_{\text{free}} = 1$; its FSM has 11 states and 103 transitions. The rate is also 0.545 bits/symbol (so the *effective* FS probability is $P_{\varepsilon,\text{eff}} = 1 - 2^{h(0.1)-0.545} = 0.051$), thus both systems have the same spectral efficiency. Decoding uses a ML or MAP Viterbi algorithm on the reduced FSM trellis.

Figure 6 shows that with ML decoding, the FSAC outperforms comparable tandem systems up to $E_s/N_0 = 3.5$ dB (SER 10^{-3}). MAP decoding, which costs only one additional multiplication per transition, yields another 0.2–0.3 dB gain compared to ML. The storage complexity of FSAC and tandem systems is comparable, since it is the product of the channel block size and the number of states. Precisely assessing the computational complexity of the FSAC is more difficult, since branch metric calculations could be optimized (many transition outputs have common suffixes), but it is certainly higher than for the CCs. However, this is without considering the complexity of the AC decoder in the tandem system.

We observed that the number of states and transitions, and hence decoding complexity, of FSAC grows mainly with the compression ratio, which is inversely related to source entropy. This disadvantage is partly compensated by the better MAP decoding performance for skewed, low-entropy sources.

VIII. Conclusions

In this paper, we have proposed analytical tools for assessing the effectiveness of the redundancy introduced into finite-state arithmetic coding schemes. When the redundancy is

small, the free distance is not a sufficient characterization; therefore, distance spectra have also been considered. Indeed, the majority of AC-based JSC techniques in the literature can be evaluated using the proposed tools, since the introduced redundancy directly affects the distance properties of the derived trellis. We have also shown that these tools may be used to design efficient AC-based JSC encoders, which at low to medium SNR outperform classical tandem schemes with equivalent rate and storage complexity.

Ongoing work includes further optimization of the FS placement, efficient recursive evaluation of the distance spectra, as well as more advanced methods of introducing redundancy in arithmetic coding.

ACKNOWLEDGMENTS

This work has been partly supported by the European Network of Excellence NEWCOM. The authors would like to thank Pierre Duhamel for valuable suggestions and comments.

APPENDIX

A. Proof of Proposition 1

The initialization is a trivial rewriting of (2) and (3) for $n = 1$. Then, to obtain (6), one may first use the fact that for any pair of paths $(\mathbf{p}_{xy}^n, \mathbf{q}_{xz}^n) \in \mathcal{D}_n(x)$, for $n \geq 2$ there exist $y', z' \in \mathcal{S}_{\text{FSM}}^{\text{b}}$ with $y' \neq z'$ (since $\mathbf{p}_{xy}^n$ and $\mathbf{q}_{xz}^n$ have not converged) such that $\mathcal{T}_{y'y}^{\text{b}}$ and $\mathcal{T}_{z'z}^{\text{b}}$ are not empty and such that $\mathbf{p}_{xy}^n = \mathbf{p}_{xy'}^{n-1} \circ t_{y'y}$ and $\mathbf{q}_{xz}^n = \mathbf{q}_{xz'}^{n-1} \circ t_{z'z}$, with $t_{y'y} \in \mathcal{T}_{y'y}^{\text{b}}$ and $t_{z'z} \in \mathcal{T}_{z'z}^{\text{b}}$. Thus $\left(\mathbf{p}_{xy'}^{n-1}, \mathbf{q}_{xz'}^{n-1}\right) \in \mathcal{D}_{n-1}(x)$ and (2) may be rewritten as

$$\begin{aligned}
\Delta_n(x,y,z) &= \min_{y' \neq z'} \min_{t_{y'y} \in \mathcal{T}_{y'y}^{\text{b}}, t_{z'z} \in \mathcal{T}_{z'z}^{\text{b}}} \min_{\substack{(\mathbf{p}_{xy'}^n, \mathbf{q}_{xz'}^n) \\ \in \mathcal{D}_{n-1}(x)}} \left(d_{\text{H}}^{\text{out}}(\mathbf{p}_{xy'}^{n-1}, \mathbf{q}_{xz'}^{n-1}) + d_{\text{H}}^{\text{out}}(t_{y'y}, t_{z'z}) \right) \\
&= \min_{y' \neq z'} \min_{t_{y'y} \in \mathcal{T}_{y'y}^{\text{b}}, t_{z'z} \in \mathcal{T}_{z'z}^{\text{b}}} \left\{ \left(\min_{\substack{(\mathbf{p}_{xy'}^n, \mathbf{q}_{xz'}^n) \\ \in \mathcal{D}_{n-1}(x)}} d_{\text{H}}^{\text{out}}(\mathbf{p}_{xy'}^{n-1}, \mathbf{q}_{xz'}^{n-1}) \right) + d_{\text{H}}^{\text{out}}(t_{y'y}, t_{z'z}) \right\} \\
&= \min_{y' \neq z'} \min_{t_{y'y} \in \mathcal{T}_{y'y}^{\text{b}}, t_{z'z} \in \mathcal{T}_{z'z}^{\text{b}}} \left\{ \Delta_{n-1}(x, y', z') + d_{\text{H}}^{\text{out}}(t_{y'y}, t_{z'z}) \right\}.
\end{aligned}$$

To obtain (7), one first rewrites (3) as

$$\Delta_n(x,y,y) = \min \left\{ \Delta_{n-1}(x,y,y), \min_{(\mathbf{p}_{xy}^n, \mathbf{q}_{xy}^n) \in \mathcal{C}_n(x,y)} d_{\text{H}}^{\text{out}}(\mathbf{p}_{xy}^n, \mathbf{q}_{xy}^n) \right\}.$$

Then, observing that the paths in the last term may be decomposed in the same fashion as above, since $\mathbf{p}_{xy}^n$ and $\mathbf{q}_{xy}^n$ converge for the first time in y, one easily gets (7).

REFERENCES

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

[2] C. Boyd, J. Cleary, I. Irvine, I. Rinsma-Melchert, and I. Witten, "Integrating error detection into arithmetic coding," *IEEE Trans. on Comm.*, vol. 45(1), pp. 1–3, 1997.

[3] J. Sayir, "Arithmetic coding for noisy channels," *Proc. IEEE Information Theory Workshop*, pp. 69–71, 1999.

[4] J. Chou and K. Ramchandran, "Arithmetic coding-based continuous error detection for efficient ARQ-based image transmission," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 6, pp. 861–867, 2000.

[5] B. D. Pettijohn, W. Hoffman, and K. Sayood, "Joint source/channel coding using arithmetic codes," *IEEE Trans. on Comm.*, vol. 49(5), pp. 826–836, 2001.

[6] T. Guionnet and C. Guillemot, "Soft decoding and synchronization of arithmetic codes: Application to image transmission over noisy channels," *IEEE Trans. on Image Processing*, vol. 12(12), pp. 1599–1609, 2003.

[7] M. Grangetto, P. Cosman, and G. Olmo, "Joint source/channel coding and MAP decoding of arithmetic codes," *IEEE Trans. on Comm.*, vol. 53(6), pp. 1007–1016, 2005.

[8] C. Demiroglu, W. Hoffman, and K. Sayood, "Joint source channel coding using arithmetic codes and trellis coded modulation," *Proc. of DCC, Snowbird, Utah, USA.*, pp. 302–311, 2001.

[9] R. C. Pasco, *Source Coding Algorithms for Fast Data Compression*. Stanford University, CA: Ph.D. Thesis Dept. of EE, 1976.

[10] J. Rissanen, "Generalized Kraft inequality and arithmetic coding," IBM *Journal of Research and Development*, vol. 20, no. 3, p. 198, 1976.

[11] P. G. Howard and J. S. Vitter, "Practical implementations of arithmetic coding," *Image and Text Compression*, vol. 13(7), pp. 85–112, 1992.

[12] B. Dongsheng, W. Hoffman, and K. Sayood, "State machine interpretation of arithmetic codes for joint source and channel coding," *Proc. of DCC, Snowbird, Utah, USA.*, pp. 143–152, 2006.

[13] S. Lin and D. Costello, "Error control coding: Fundamentals and applications," *Englewood Cliffs, Prentice-Hall*, 1983.

[14] V. Buttigieg, "Variable-length error correcting codes," PhD dissertation, University of Manchester, Univ. Manchester, Manchester, U.K., 1995. [Online]. Available: http://www.eng.um.edu.mt/ vjbutt/research/thesis.zip

[15] V. Buttigieg and P. Farrell, "Variable-length error-correcting codes," *IEE Proceedings on Communications*, vol. 147, no. 4, pp. 211–215, Aug. 2000.

[16] G. D. Forney, "Geometrically uniform codes," *IEEE Trans. on Information Theory*, vol. 37(5), pp. 1241–1260, 1991.

[17] A. J. Viterbi and J. Omura, "Principles of digital communication and coding," *McGraw-Hill, New-York, USA.*, 1979.

[18] V. Levenshtein, "Binary codes with correction of deletions, insertions and substitution of symbols," *Dokl. Akad. Nank. SSSR*, vol. 163, no. 4, pp. 845–848, 1965.

[19] J. Lassing, T. Ottosson, and E. Strom, "On the union bound applied to convolutional codes," in *Proc. 54th IEEE Vehicular Technology Conference (VTC 2001 Fall)*, vol. 4, 7-11 Oct. 2001, pp. 2429–2433.

[20] D. Haccoun and G. Bégin, "High rate punctured convolutional codes for Viterbi and sequential decoding," *IEEE Trans. on Comm.*, vol. 37, no. 11, pp. 1113–1125, Nov. 1989.

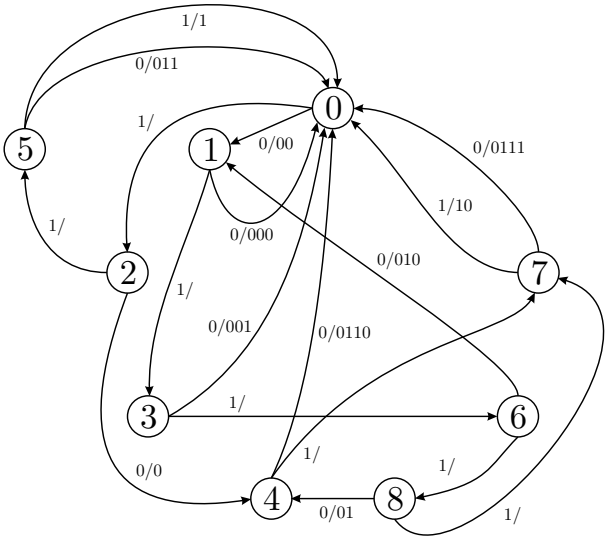

Fig. 1. Finite state machine obtained for $p = 4,\ P_0 = 0.2,\ F_{\max} = 1$.

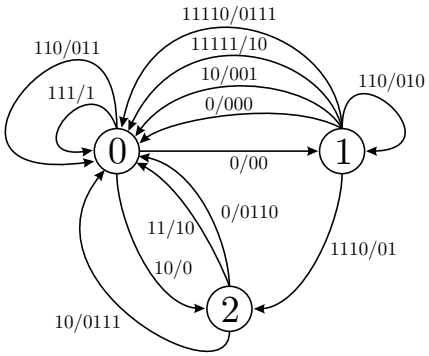

Fig. 2. Reduced state machine obtained for $p = 4,\ P_0 = 0.2,\ F_{\max} = 1$.

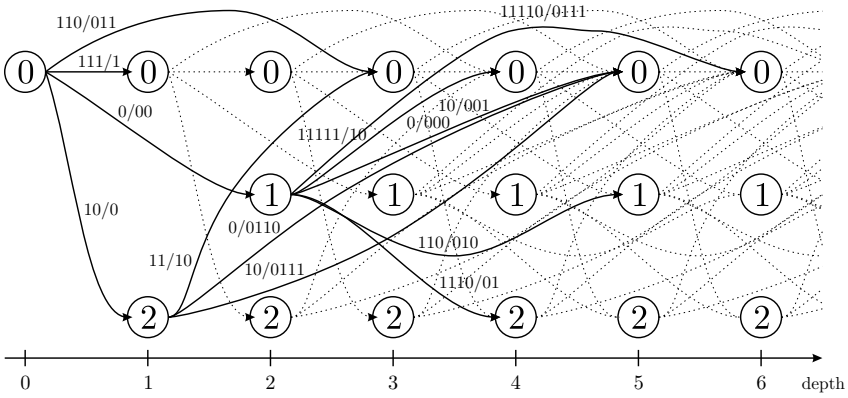

Fig. 3. Trellis representation derived from the reduced FSM ($p = 4, P_0 = 0.2, F_{\max} = 1$)

$F_{\max}$	1	3	6	9
Additional redundancy (bits/symbol)	0.0505 (9.3%)	0.0109 (2.0%)	0.0008 (0.14%)	0.0004 (0.06%)
$\lvert\mathcal{S}^{\mathrm{s}}_{\mathrm{FSM}}\rvert$	84	154	207	240
$\lvert\mathcal{T}^{\mathrm{s}}_{\mathrm{FSM}}\rvert$	168	308	414	480
$\lvert\mathcal{S}_{\mathrm{FSM}}\rvert$ for the reduced trellis	19	32	32	32
$\lvert\mathcal{T}_{\mathrm{FSM}}\rvert$ for the reduced trellis	130	248	364	410

TABLE I

EFFECT OF THE CHOICE OF $F_{\max}$ ON THE TRELLIS PROPERTIES, $p = 6, P_0 = \frac{1}{8}, P_\varepsilon = 0, h(P_0) = 0.5435$ BITS/SYMBOL

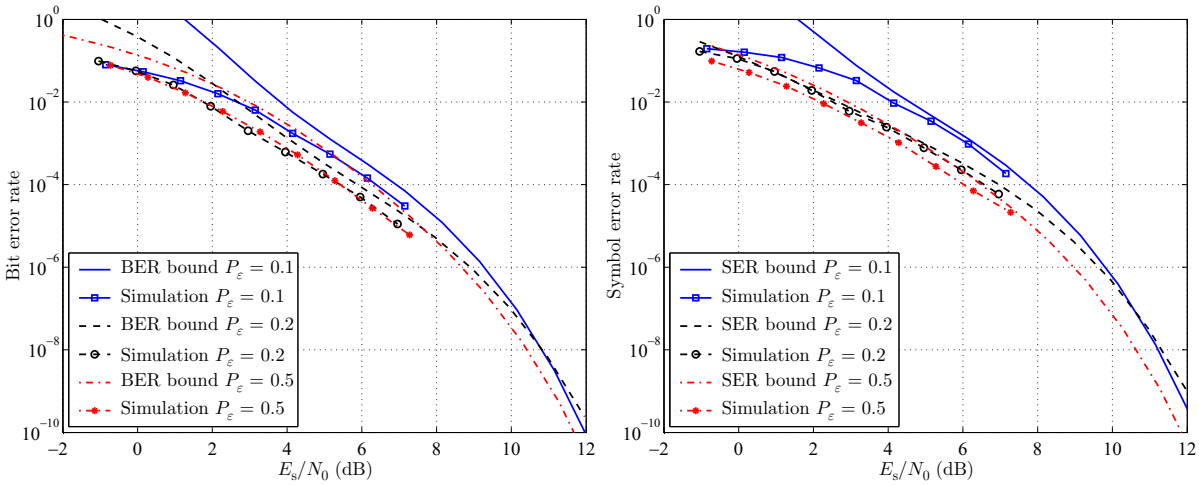

Fig. 4. Bounds on BER and SER compared to simulations for different amounts of additional redundancy

P_ε	0.1		0.2		0.3		0.4		0.5	
	b	w	b	w	b	w	b	w	b	w
q_1	0.4	0	0.2	0	0.7	0	0	0.3	0.45	0
q_2	0.3	1	0.55	1	0.05	1	0.6	0.7	0.5	1
Rate	0.75	0.90	0.88	1.08	1.12	1.29	1.27	1.57	1.60	1.76
d_{free}	1	1	1	1	1	1	2	1	3	1
$A_{d_{\mathrm{free}}}$	0.90	0.31	0.22	0.20	0.004	0.03	0.59	0.02	0.13	0.009
$B_{d_{\mathrm{free}}}$	4.75	1.56	0.95	0.87	0.016	0.13	2.66	0.07	0.43	0.009
$\lvert\mathcal{S}_{\mathrm{FSM}}\rvert$	17	22	15	9	8	12	2	11	11	4
$\lvert\mathcal{T}_{\mathrm{FSM}}\rvert$	101	122	86	55	37	59	9	52	33	21

TABLE II

EFFECT OF P_ε ON THE TRELLIS FEATURES, $p = 5, F_{\max} = 2, P_0 = 0.2$. COLUMNS B AND W DENOTE THE BEST AND THE WORST CASE, RESPECTIVELY.

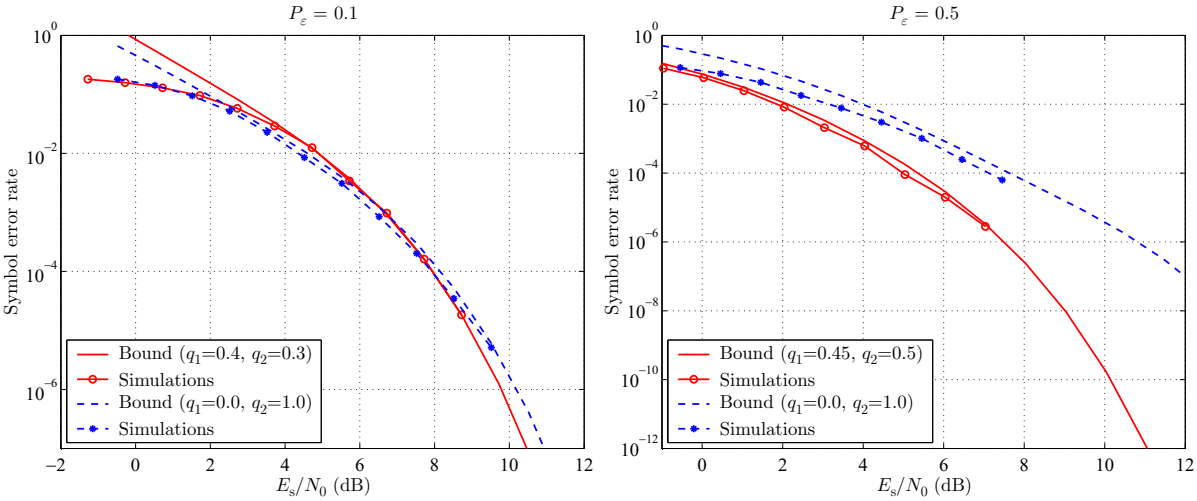

Fig. 5. Comparison between best and worst configurations of I_{FS}, for $P_\varepsilon = 0.1$ and $P_\varepsilon = 0.5$ (codes from Table II)

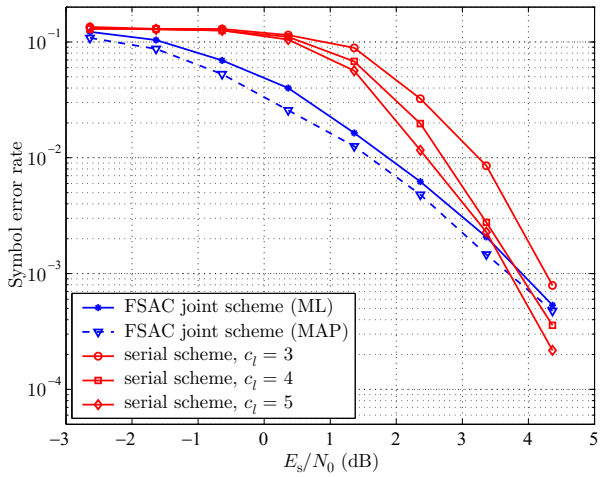

Fig. 6. Comparing the error correcting performance of the proposed joint scheme to a classical tandem scheme (AC + CC), using rate $7/8$ punctured CC with $c_l = 3$, $c_l = 4$, and $c_l = 5$