Soft and Joint Source-Channel Decoding of Quasi-Arithmetic Codes

Thomas Guionnet
Projet TEMICS, IRISA-INRIA, Campus Universitaire de Beaulieu, 35042 Rennes Cedex, France
Email: thomas.guionnet@irisa.fr

Christine Guillemot
Projet TEMICS, IRISA-INRIA, Campus Universitaire de Beaulieu, 35042 Rennes Cedex, France
Email: christine.guillemot@irisa.fr

Received 20 November 2002; Revised 7 August 2003; Recommended for Publication by Antonio Ortega

The issue of robust and joint source-channel decoding of quasi-arithmetic codes is addressed. Quasi-arithmetic coding is a reduced precision and complexity implementation of arithmetic coding. This amounts to approximating the distribution of the source. The approximation of the source distribution leads to the introduction of redundancy that can be exploited for robust decoding in presence of transmission errors. Hence, this approximation controls both the trade-off between compression efficiency and complexity and at the same time the redundancy (excess rate) introduced by this suboptimality. This paper provides first a state model of a quasi-arithmetic coder and decoder for binary and M-ary sources. The design of an error-resilient soft decoding algorithm follows quite naturally. The compression efficiency of quasi-arithmetic codes allows to add extra redundancy in the form of markers designed specifically to prevent desynchronization. The algorithm is directly amenable for iterative source-channel decoding in the spirit of serial turbo codes. The coding and decoding algorithms have been tested for a wide range of channel signal-to-noise ratios (SNRs). Experimental results reveal improved symbol error rate (SER) and SNR performances against Huffman and optimal arithmetic codes.

Keywords and phrases: robust arithmetic and quasi-arithmetic coding, joint source-channel coding, soft decoding, estimation, MAP.

1. INTRODUCTION

Entropy coding, producing variable length codewords (VLCs), is a core component of any data compression scheme. However, VLCs are very sensitive to channel noise: when some bits are altered by the channel, synchronization losses can occur at the receiver, the position of symbol boundaries are not properly estimated, leading to dramatic symbol error rates (SERs). This phenomenon has given momentum to extensive work on the design of procedures for soft decoding and joint source-channel decoding of VLCs. Soft VLC decoding ideas, exploiting residual source redundancy (the so-called excess rate), have also been shown to reduce the “desynchronization” effect as well as the residual bit error rate and SER [1, 2, 3]. Models incorporating both VLC-encoded sources and channel codes (CCs) have also been considered [4, 5, 6, 7].

The research effort has first focused on Huffman codes [3, 4, 5, 8] and on reversible VLCs [6, 9, 10]. However, arithmetic codes have gained increased popularity in practical systems, including JPEG2000, H.264, and MPEG-4 standards. Arithmetic coding allows to decouple the coding process from the source model, hence can be used in conjunction with any probabilistic model. A good statistical model of the source is a key element to obtain maximum compression performance. However, the counterpart to the high compression efficiency is an increased sensitivity to noise: a single bit error causes the internal decoder state to be in error. Methods considered to fight against noise sensitivity consist usually in reaugmenting the redundancy of the bit stream either by introducing an error-correcting code or by inserting dedicated patterns in the chain. Along those lines, the author in [11] reintroduces redundancy in the form of parity check bits embedded into the arithmetic coding procedure. A probability interval not assigned to a symbol of the source alphabet or markers inserted at known positions in the sequence of symbols to be encoded are exploited for error detection in [12, 13, 14]. This capability can then be coupled with an automatic request for retransmission (ARQ) procedure [13, 15] or used jointly with an error-correcting
The complexity of this approach is reduced in [18] by using trellis-coded modulation combined with a list Viterbi decoding algorithm. A soft decoding procedure is described in [19]. However, one difficulty comes from the fact that the code tree, hence the state-space dimension, or the number of states of the model grows exponentially with the number of symbols being encoded. A pruning technique is then used to limit the complexity within a tractable and realistic range. However, it brings inherent limitations when using the decoder in an iterative source-channel decoding structure. The pruning of the tree to limit the complexity is such that, in this particular case, the iterations do not bring a significant gain.

A fast arithmetic coding procedure called quasi-arithmetic coding has been introduced in [20]. It operates on an integer interval [0, T] and on integer subdivisions of this interval. This amounts to approximate the source distribution. This controlled approximation allows to reduce the number of possible coder states without significantly degrading the compression performance. The trade-off between the state-space dimension and the source distribution approximation is controlled by the parameter T. If T is sufficiently small, all state transitions and outputs can then be precomputed and table lookups be in turn substituted for arithmetic operations. A quasi-arithmetic coder can be regarded as an arithmetic coder governed by an approximation of the real source distribution.

In this paper, we first revisit finite-state automaton modeling of quasi-arithmetic coding and decoding processes for M-ary sources. Notice that the M-ary source could be coded directly with a quasi-arithmetic coder. An accurate approximation of the M-ary source distribution would however require to set the parameter T to a high value, resulting in a high state-space dimension, hence in high decoding complexity. To maintain the complexity within a tractable range, one would have to rely on a very coarse source distribution approximation. One can instead first map the M-ary source model into a binary model by means of a fixed-length binary code represented as a binary tree with symbols at the leaves. These trees are connected up to a depth function of the source model (e.g., for an order-1 Markov source, the depth is one). Leaves of the tree represent terminated symbols and are identified with the root of the next tree. The resulting finite binary tree can be regarded as a stochastic automaton that models the source symbol distribution. Once the M-ary source has been converted into a binary source, the latter can be encoded by a quasi-arithmetic coder. The transitions on the binary model govern the quasi-arithmetic coder. The design of an efficient estimation procedure based on the BCJR algorithm [21] follows quite naturally. The decoding complexity remains within a realistic range without the need for applying any pruning of the estimation trellis. The estimation algorithm has been validated under various channel conditions and for different levels of source correlation. Experimental results have shown very high error resilience while at the same time preserving a very good compression efficiency. For a comparable overall rate, in comparison with Huffman codes, better compression efficiency of quasi-arithmetic codes allows to dedicate extra redundancy (short "soft" synchronization patterns) specifically to decoder resynchronization, resulting in significantly higher error resilience. The usage of CCs is also considered in order to reduce the bit error rate seen by the source estimation algorithm. The latter can then be placed in an iterative decoding structure in the spirit of serially concatenated turbo codes, provided that the channel decoder and the quasi-arithmetic decoder are separated by an interleaver. Since, in contrast with optimal arithmetic coding, the estimation can be performed without pruning the trellis, the potential of the iterative decoding structure can be fully exploited, resulting in a very low SER (significantly lower than what can be obtained with Huffman codes). Overall, the great flexibility that quasi-arithmetic codes offer for adjusting compression efficiency, error resilience, and complexity allows an optimal adaptation to various transmission conditions and terminal capability requirements.

The rest of the paper is organized as follows. Section 2 describes the notations we use and states the problem addressed. Sections 3 and 4 review the principles of arithmetic and quasi-arithmetic coding. Sections 5 and 6 address modeling issues of, respectively, coding/decoding processes and of the source. This material is exploited in the sequel (Sections 7 and 8) for explaining the estimation algorithm and the soft synchronization procedure. Section 9 outlines the construction of the iterative joint source-channel decoding procedure based on quasi-arithmetic codes. Finally, experimental results are described in Section 10. We first compare the performance of the algorithm in terms of SER and signal-to-noise ratio (SNR) with respect to soft Huffman and arithmetic decoding with theoretical Gauss-Markov sources. Simulations results of the joint source-channel turbo decoding algorithm in comparison with soft decoding of quasi-arithmetic codes are also provided.

### 2. NOTATIONS AND PROBLEM STATEMENT

Let \( A = A_1 \cdots A_L \) be a sequence of quantized source symbols taking their values in a finite alphabet \( \mathcal{A} \) composed of \( M = 2^L \) symbols, \( \mathcal{A} = \{ a_1, a_2, \ldots, a_M \} \). The sequence \( A = A_1 \cdots A_L \) is assumed to form an order-1 Markov chain. This sequence of M-ary symbols is converted into a sequence of binary symbols \( S = S_1 \cdots S_K \), where \( K = q \times L \). This binary source is in turn coded into a sequence of information bits \( U = U_1 \cdots U_N \) by means of a quasi-arithmetic coder as depicted in Figure 1. The length \( N \) of the information bit stream is a random-variable function of \( S \), hence of \( A \). The bit stream \( U \) is sent over a memoryless channel and received as measurements \( Y \); so the problem we address consists in estimating \( A \), given the observed values \( Y \). Notice that we reserve capital letters to random variables and small letters to values of these variables. For handling ranges of variables, we use the notation \( X^+_I = \{ X_{u+1}, \ldots, X_{v} \} \) or \( X_I \), where \( I \) is the index set \( \{ u, u + 1, \ldots, v \} \).
has been encoded, the current interval is \([0, 0.5832]\). Any number in this interval can be used to identify the sequence. We consider 0.576. The decoding of the sequence is performed by reproducing the coder behavior. First, the interval \([0, 1]\) is partitioned according to the cumulative probability of the source. Since the value 0.576 belongs to the interval \([0, 0.6]\), it is clear that the first symbol encoded has been \(a_1\). Therefore, the first symbol is decoded and the interval \([0, 0.6]\) is partitioned according to the cumulative probability of the source. The process is repeated until full decoding of the sequence. Practical implementations of arithmetic coding have been first introduced in \([22, 23]\) and developed further in \([24]\). One problem that may arise when implementing arithmetic coding is the high precision needed to represent very small real numbers. In order to overcome this difficulty, one can base the algorithm on the binary representation of real numbers in the interval \([0, 1]\) (see \([25]\)). Any number in the interval \([0, 0.5]\) will have its first bit equal to 0, while any number in the interval \([0.5, 1]\) will have its first bit equal to 1. Therefore, during the encoding process, as soon as the current interval is entirely under or over 1/2, the corresponding bit is emitted and the interval length is doubled. There is a specific treatment for the intervals straddling 1/2. When the current interval straddles 1/2 and is in \([0.25, 0.75]\], it cannot be identified by a unique bit. Its size is therefore doubled without emitting any bit, and the number of rescaling operations taking place before emitting any bit is memorized. When reaching an interval for which one bit \(U_i = u_i\) can be emitted, then this bit will be followed by a number of bits \(U_{i+1} = u_{i+1} \cdots U_{n+1} = u_{n+1}\), where \(n\) is the number of scaling operations that have been performed before the emission of \(U_n\) and where \(u_{i+1} = u_i + 1 \mod 2\). The use of this technique guarantees that the current interval always satisfies \(0 < 0.25 < 0.5 \leq \text{high} \) or \(0 < 0.5 < 0.75 \leq \text{high}\), where low and high are, respectively, the lower and upper bounds of the current interval. This avoids the problems of precision which may otherwise occur in the case of small intervals straddling the middle of the segment \([0, 1]\).

### 4. FAST-REDUCED PRECISION IMPLEMENTATION

Arithmetic coding is near optimality in terms of compression. Error-resilient decoding solutions can be designed \([19]\), however, their complexity can be an issue in some contexts. The coding process can indeed be modeled under the form of a stochastic automaton, where the states are defined by three variables: low, up (denoting the bounds of the subinterval resulting from successive subdivisions of the interval \([0, 1]\)), and \(n\) scl denoting the number of scalings performed since the last emitted bit, as explained in Section 3. Subdivisions of the interval \([\text{low}, \text{up}]\), hence next states, are functions of the cumulative probability distribution of the source. Without the source model, the possible number of subdivisions, hence of states, may be infinite. If the source distribution is known, the number of states still grows exponentially with the number of symbols being encoded.

#### 4.1. Quasi-arithmetic coding

It is observed in \([20]\) that controlled approximations can reduce the number of possible states without significantly degrading compression performance. All state transitions and outputs can then be precomputed and table lookups can be used instead of arithmetic operations. This fast, but reduced precision, implementation of arithmetic coding is called quasi-arithmetic coding \([20]\). Instead of using the real interval \([0, 1]\), quasi-arithmetic coding is performed on an integer interval \([0, T]\). The value of \(T\) controls the trade-off between complexity and compression efficiency: if \(T\) is sufficiently large, then the interval subdivisions will follow closely the distribution of the source. In contrast, if \(T\) is small, all the interval subdivisions can be precomputed.

Given the \(M\)-symbol alphabet \(A\), the sequence of symbols \(A_{i1}^{T}\) is translated into a sequence of bits \(U_{i1}^{T}\) by an \(M\)-ary decision tree. This tree can be regarded as an automaton that models the bit stream distribution. The encoding of a symbol determines the choice of a vertex or branch in the tree. Each node of the tree identifies a state \(X\) of the arithmetic coder and to each transition can be associated the emission of a sequence of bits of variable lengths. Successive branching
on the tree (or transitions between states) follow the distribution of the source ($P(A_t|A_{t-1})$ for an order-1 Markov source or $P(A_t)$ in the zeroth-order case). Let $X_t$ denote the state of the automaton at each symbol instant $t$. As in the case of optimal arithmetic coding, the state $X_t$ of the quasi-arithmetic coder is defined by three variables: low $A_t$, up $A_t$, and $n_{scl}$. The terms low $A_t$ and up $A_t$ denote the bounds of the subinterval resulting from successive subdivisions of the interval $[0, T]$ triggered by the encoding of the sequence $A^t$. The quantity $n_{scl}$ is (re)set to zero when a bit is emitted and incremented each time a rescaling takes place. Hence, this quantity denotes the number of scalings performed since the last emitted bit. When a bit is emitted, it is followed by $n_{scl}$ bits of opposite value (see Section 3).

Since there is a finite number of possible integer subdivisions of the interval $[0, T]$, all the possible states of the quasi-arithmetic coder can be precomputed without knowledge of the source. This is however without accounting for the variable $n_{scl}$. Indeed, the variable $n_{scl}$ is not bounded. The solution is then to consider $n_{scl}$ as a variable resulting from state transitions (output variable) and not to consider this variable in the precomputation of the coder states. Table 1 gives the states, outputs, and all possible transitions from state transitions (output variable) and not to consider this variable in the precomputation of the coder states. The number of states can be further reduced by identifying the symbols as more probable (MPS) and less probable (LPS) rather than as 1 and 0. This amounts to reducing the number of possible combinations of the binary source probabilities, the MPS being either the symbol 0 or the symbol 1. This allows to combine transitions and eliminate states as shown in Table 1. The sequence $X_0 \cdot \cdots \cdot X_T$ is a Markov chain and the output of the coder is the function of transitions of this chain. This state representation can help designing a robust maximum a posteriori (MAP) decoder. For a deeper understanding of quasi-arithmetic coding, the reader may refer to [20].

## 4.2. Quasi-arithmetic decoding

We first consider the operation of an optimum arithmetic decoder. A sequence of arithmetically coded bits $U^n_T$ is translated into a sequence of symbols $A^t_T$ by a binary decision tree. Each bit determines the choice of a vertex in the tree. Each node $v$ of the tree identifies a state of the arithmetic decoder and corresponds to a tuple $U^n_v$ from which two transitions are possible: $U^n_v = 0$ or $U^n_v = 1$. The state of the decoder is specified by two intervals: [low $U^n_v$, up $U^n_v$] and [low $A_{L,n}$, up $A_{L,n}$]. The interval [low $U^n_v$, up $U^n_v$] defines the segment of the interval $[0, 1]$ selected by a given input bit sequence $U^n_v$. The interval [low $A_{L,n}$, up $A_{L,n}$] relates to the subinterval obtained when the symbol $A_{L,n}$ can be decoded without ambiguity. Both intervals must be scaled appropriately in order to avoid numerical precision problems. However, in contrast to the coding process, there is no need to keep track of the scalings that have been performed.

We now consider the interval $[0, T]$ and finite interval subdivisions. The quasi-arithmetic decoder can also be expressed in the form of an automaton. Let $X_n$ be its state at bit instant $n$; $X_n$ stores the four variables [low $U_n$, up $U_n$] and [low $A_{L,n}$, up $A_{L,n}$]. Since there is a finite number of possible subdivisions of the interval $[0, T]$, there is a finite number of states for the quasi-arithmetic decoder which can be pre-computed. Table 2 gives the states, transitions, and outputs of the quasi-arithmetic decoder for a binary source.

### Table 1: Quasi-arithmetic coder states, transitions, and outputs for $T = 4$.

<table>
<thead>
<tr>
<th>State $X_t$</th>
<th>[low $A_t$, up $A_t$]</th>
<th>$P(0)$ (corresponding interval subdivision)</th>
<th>Normal state model</th>
<th>Simplified state model</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (start)</td>
<td>[0, 4]</td>
<td>$0.63 \leq P(0) &lt; 0.63$</td>
<td>0</td>
<td>MPS Out 1 11 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$0.37 \leq P(0) &lt; 0.63$</td>
<td></td>
<td>LPS Out 1 11 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$P(0) &lt; 0.37$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$0.00 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>[0, 3]</td>
<td>$0.5 \leq P(0) &lt; 0.5$</td>
<td>0</td>
<td>MPS Out 0 10 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$0.00 0$</td>
<td></td>
<td>LPS Out 0 10 0</td>
</tr>
<tr>
<td>2</td>
<td>[1, 4]</td>
<td>$0.5 \leq P(0) &lt; 1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$0.01 0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: $P(0)$ denotes the probability of emitting a 0 in the state $X_t$.
and $T = 4$, with the MPS/LPS simplification. The decoder in this particular example has two states. Further subdivisions that will lead to transitions to next states are functions of the source probability distribution (e.g., $P_{\text{MPS}}$ in Table 2). We assume, for example, that the automaton is in state $X_n = 0$ (defined by the two intervals $[\text{low } U_n, \text{up } U_n] = [0, 4]$ and $[\text{low } A_{L_n}, \text{up } A_{L_n}] = [0, 4]$), and that the input is $U_n = 0$. Depending on the probability of the source to be coded (hence here of the MPS and LPS symbols), the interval $[0, 4]$ will be further subdivided into $[0, 3]$ (if MPS probability is higher than 0.63) or into $[0, 2]$ (if MPS probability is lower than 0.63), both subdivisions resulting into the state 0.

### 4.3. Source distribution approximation

Arithmetic coding realizes a conversion of source distributions: in the general case, it realizes a conversion of a sequence of symbols of an $M$-ary source of a given distribution into a sequence of symbols of a binary source with an independent and uniform distribution. A quasi-arithmetic coder does not produce a sequence of independently and uniformly distributed bits due to the approximation of the source distribution that it realizes. It has been shown in [26] that this approximation does not induce a significant increase in code length. This however depends on the source statistics. This statement is true if the symbol probabilities are comprised between $1/t$ and $(t - 1)/t$, where $t$ is the width of the current interval to be subdivided. Hence, the choice of the value of $T$ may depend on the source distribution. The excess rate resulting from the approximation can be computed as follows. Let $a$ be a symbol taking its value in $\mathcal{A}$, $P(a)$ the probability of the event $a$, and $Q(a)$ the approximation of $P(a)$ made by the quasi-arithmetic coder. The entropy of the source is given by

$$H = - \sum_{a \in \mathcal{A}} P(a) \log_2 P(a). \quad (1)$$

The performance of the quasi-arithmetic coder can then be measured by

$$R = - \sum_{a \in \mathcal{A}} P(a) \log_2 Q(a). \quad (2)$$

The excess rate induced by the quasi-arithmetic coder can hence be expressed as

$$E = R - H = \sum_{a \in \mathcal{A}} P(a) \log_2 \frac{P(a)}{Q(a)} = D(P || Q),$$

where $D(P || Q)$ is the Kullback-Leibler distance or relative entropy between the approximate distribution $Q$ and the true distribution $P$.

### 5. Source Model

For a binary source, the variable $T$ can be limited to a small value (down to 4) at a small cost in terms of compression [26]. This motivates a conversion of the $M$-ary source into a binary source to be then encoded by the quasi-arithmetic coder. This conversion amounts to a fixed-length binary coding of the source.

We first assume that the source quantized on $M = 2^t$ symbols is an order-0 Markov source. It can then be represented by a binary tree of depth $g$, as shown in Figure 2a for $M = 4$. The transition probabilities on the binary tree can be computed easily from the distribution of the source. The resulting binary tree can be seen as an automaton that models the $M$-ary source stationary distribution. A complete model of the source can be obtained by connecting the successive local models. One possible solution consists in identifying the leafnodes of the binary tree with the rootnode of the next tree. This leads to the three states automaton of Figure 2b in the particular case of an order-0 Markov source with $M = 4$. In Figure 2c, the same automaton is shown in the form of a trellis, with the probability of each bit indicated on the corresponding transition. Relying on this stochastic automaton model, the sequence of the resulting binary symbols can be modeled as a function of a hidden Markov model. Let $C_k$ denote the state $v$ of the automaton after $k$ binary symbols have been produced. The sequence $C_0, \ldots, C_k$ is a Markov chain, and the resulting sequence of binary symbols is a function of transitions of this chain, that is, $S_k = \phi(C_{k-1}, C_k)$.

We now assume that the source is an order-1 Markov source. To take into account the correlation present in the $M$-ary source, in the model construction, one must in addition keep track of the last $M$-ary symbol coded. In order to do so, one could define the state variable $C_k$ as a pair $(\nu, \sigma)$, where $\nu$ is the value of the last completed symbol $A_i$ and $\nu$ is the current state of the stochastic automaton describing the construction of the next $M$-ary symbol, following
Figure 2: Graphical representation under the form of (a) a tree, (b) a stochastic automaton, and (c) a trellis of the binary model of the order-0 Markov source. Black dots correspond to leafnodes identified to rootnodes of next trees. White dots correspond to intermediate nodes of the binary representation of the $M$-ary symbols ($M = 4$).

Figure 3: Graphical representation under the form of (a) a tree and (b) a stochastic automaton of the binary model of the order-1 Markov source ($M = 4$).

Alternately, one can take into account the conditional distribution $P(A_{l+1} | A_l)$, by connecting $M + 1$ trees of depth $q$ as shown in Figure 3a, and define the state variable $C_k$ as a node $\nu$ in the resulting tree. The complete model of the source is then obtained by additional transitions from leaves to intermediate nodes as shown in Figure 3b. In this particular case, the model leads to a state space of dimension 15. The state-space dimension for an order-$n$ Markov source quantized on $M$ symbols is given by $M^{n+1} - 1$. For both state models ($C_k = (\nu, \sigma)$ and $C_k = (\nu)$), the sequence $C_1 \cdots C_K$ is a Markov process, the transitions of which produce the sequence $S$ of binary symbols.

Remark 1. The construction of the binary model, that is, the assignment of the binary codewords to the different symbols, has a strong impact on the reliability of the estimation. The transition probabilities on the binary tree are indeed defined by this symbol-to-leave assignment. Increased estimation reliability can be obtained when the different paths (or branches) on the model have a highly nonuniform likelihood, that is, when the uncertainty of some branches is much lower than for others. The assignment of binary codewords to symbols must hence be such that the expression $\sum_X P(X)H(X)$ is minimized, where $P(X)$ is the probability of the state $X$ of the model and where $H(X)$ is the entropy of the transitions leaving $X$. For small source alphabets, the minimization can be achieved by a simple exhaustive search approach. However, for larger size alphabets, the exhaustive search among all possible index assignments rapidly becomes untractable. This complexity can be reduced by limiting the
search space to a subset of index assignments. In the experiments reported here, this subset has been obtained by a simple circular shift of an initial index assignment. This initialization has an impact on the resulting SER and SNR performances. A lexicographic index assignment to symbols ranked by decreasing probability values turned out to provide good performance.

6. Modeling Bit Stream Dependencies

In order to design efficient algorithms for estimating the sequence of symbols that has been emitted, one has now to build a model of bit stream dependencies. For this, we consider the product (in the sense of product on automata or of tensor product of stochastic models) of the source and coder or source and decoder models. Estimation algorithms can be defined for both models. However, in Section 7, the estimation is performed only on the product of the source and decoder models, since with the source and coder models, handling the \( n \) scalar values of the source model has 2 states and the binary source model has 3 states, the dimension of the resulting state space should in general be reduced. Hence reducing the state-space dimension from 6 to 4.

Table 3: Source-coder product model: states, outputs, and transitions.

<table>
<thead>
<tr>
<th>State</th>
<th>State Variables</th>
<th>MPS</th>
<th>LPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (start)</td>
<td>([0, 4])</td>
<td>output: 0</td>
<td>output: 1</td>
</tr>
<tr>
<td>(C = 0)</td>
<td>next state: 1</td>
<td>next state: 2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>([0, 4])</td>
<td>output: —</td>
<td>output: 11</td>
</tr>
<tr>
<td>(C = 1)</td>
<td>next state: 3</td>
<td>next state: 0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>([0, 4])</td>
<td>output: —</td>
<td>output: 11</td>
</tr>
<tr>
<td>(C = 2)</td>
<td>next state: 3</td>
<td>next state: 0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>([0, 5])</td>
<td>output: 0</td>
<td>output: 10</td>
</tr>
<tr>
<td>(C = 0)</td>
<td>next state: 1</td>
<td>next state: 2</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: Source-coder product model: trellis representation.

| \(X_k = 0\) | MPS (0) | LPS (10) |
| \(X_k = 1\) | MPS (0) | LPS (11) |

Figure 5: Quasi-arithmetic coder model: (a) \(0.63 \leq \mathbb{P}(MPS)\) and (b) \(0.5 \leq \mathbb{P}(MPS) < 0.63\).

In general, the state-space dimension cannot be known a priori; it must be computed, given \( T \) and the binary source model. If \( N_C \) is the number of states of the binary source model, the maximum number of states is \(3T^2N_C/16\).

The number of bits produced by each transition on the above model being random, the structure of dependencies between the sequence of measurements and the model states is random. In order to capture this randomness, we actually consider the augmented Markov chain \((X, N) = (X_1, N_1) \cdots (X_K, N_K)\). This yields the structure of dependencies graphically depicted in Figure 6. Using this model, a sequence of binary symbols \(s_k \) is translated into a sequence of bits \(U_k^N\), where \(N_k\) is the number of bits emitted when \(K\) symbols have been encoded. Given a state \(X_k\) and an input symbol \(S_{k+1}\), the automaton specifies the bits
\[ \bar{U}_{k+1} = U_{N_k+1}^{N_{k+1}} \] that have to be emitted and the next state \( X_{k+1} \). Notice that no bits may be emitted by a transition. The probabilities of successive branchings (e.g., of transitions between \( (X_k, N_k) = (\text{low } S_k, \text{up } S_k, C_k, N_k) \) and \( (X_{k+1}, N_{k+1}) = (\text{low } S_{k+1}, \text{up } S_{k+1}, C_{k+1}, N_{k+1}) \) in the trellis are given by the binary source model, that is, \( P(C_{k+1} | C_k) = P(S_{k+1} | C_k) \). Measurements \( \bar{Y}_k \) on the bits \( \bar{U}_k \) are gathered at the output of the transmission channel.

### 6.2. Product model: source and decoder

A product model of source and decoder can be constructed similarly. The state of the product system must gather state information of the source and decoder models. Hence, the state \( X_n \) of the product system is defined as \( X_n = (\text{low } U_n, \text{up } U_n, \text{low } S_n, \text{up } S_n, C_n) \). The system resulting from the product of the decoder of Table 2 and the simple source model of Figure 2c is illustrated in Table 4 and Figure 7, respectively. Again, the state space dimension depends on the coder precision parametrized by \( T \) and of the source model.

The number of symbols being produced by each transition on the above model is random. Therefore, the structure of dependencies between the sequence of measurements and the sequence of decoded symbols is random. This is handled by considering the augmented Markov chain \( (X, k) = (X_1, K_1) \cdots (X_N, K_N) \) with the structure of dependencies graphically depicted in Figure 8. Using this model, a sequence of bits \( U_1^N \) is translated into a sequence of symbols \( S_1^K \), where \( K_n \) is the number of symbols decoded when \( N \) bits have been received. Given a state \( X_n \) and an input bit \( U_{n+1} \), the automaton produces the sequence of symbols \( S_{n+1} = S_{K_n+1}^{K_{n+1}} \) and the next state \( X_{n+1} \). The probabilities of successive branchings (i.e., of transitions between \( (X_n, K_n) \) and \( (X_{n+1}, K_{n+1}) \)) in the trellis depend on the binary source model and are given by

\[
\prod_{k=K_n+1}^{K_{n+1}} P(S_k | C_{k-1}). \tag{4}
\]

**Table 4: Source-decoder product model: states, outputs, and transitions.**

<table>
<thead>
<tr>
<th>State</th>
<th>State variables</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (start)</td>
<td>[0, 4]</td>
<td>output: MPS</td>
<td>output: LPS</td>
</tr>
<tr>
<td>C = 0</td>
<td>[0, 4]</td>
<td>next state: 1</td>
<td>next state: 2</td>
</tr>
<tr>
<td>1</td>
<td>[0, 4]</td>
<td>[0, 4]</td>
<td>output: MPS, MPS</td>
</tr>
<tr>
<td>C = 1</td>
<td>[0, 1]</td>
<td>next state: 1</td>
<td>next state: 2</td>
</tr>
<tr>
<td>2</td>
<td>[0, 4]</td>
<td>[0, 4]</td>
<td>output: MPS, MPS</td>
</tr>
<tr>
<td>C = 2</td>
<td>[0, 1]</td>
<td>next state: 1</td>
<td>next state: 3</td>
</tr>
<tr>
<td>3</td>
<td>[2, 4]</td>
<td>[0, 4]</td>
<td>output: MPS, LPS</td>
</tr>
<tr>
<td>C = 1</td>
<td>[0, 1]</td>
<td>next state: 2</td>
<td>next state: 0</td>
</tr>
<tr>
<td>4</td>
<td>[2, 4]</td>
<td>[0, 4]</td>
<td>output: MPS, LPS</td>
</tr>
<tr>
<td>C = 2</td>
<td>[0, 1]</td>
<td>next state: 2</td>
<td>next state: 0</td>
</tr>
</tbody>
</table>

**Figure 6: Source-coder product model \( (N_k \geq N_{k-1}) \).** White and black dots represent, respectively, the hidden and observed variables.

**Figure 7: Source-decoder product model: trellis representation.**
6.3. Source distribution approximation

The entropy of the M-ary source computed on the binary model in Section 6 (e.g., Figure 2c) is given by

\[ H(S|C) = - \sum_{c=0}^{N_C-1} P(c) \sum_{x=0,1} P(s|x) \log_2 P(s|x), \quad (5) \]

where \( C \) denotes the state variable of the source model, \( c \) the indices of the possible values it can take, and \( N_C \) the dimension of the state space. The performance of the quasi-arithmetic coder when applied on this binary source for a given value of the parameter \( T \) can be measured by

\[ R(S|X) = - \sum_{x=0}^{N_X-1} P(x) \sum_{s=0,1} P(s|x) \log_2 Q(s|x), \quad (6) \]

where \( X \) denotes the state variable of the product model (source + coder), \( x \) represents the possible indices taken by the variable \( X \), and \( N_X \) is the dimension of the state-space function of \( T \). The quasi-arithmetic coder realizes an approximation of the distribution of the binary source, hence of the \( M \)-ary source. This approximation is, however, for a given value of the parameter \( T \), more accurate than when applying the quasi-arithmetic coder directly on the \( M \)-ary source. Here, the excess rate for a given value of \( T \) is given by

\[ E(S|X) = R(S|X) - H(S|X) \]

\[ = \sum_{x=0}^{N_X-1} P(x) \sum_{s=0,1} P(s|x) \log_2 \frac{P(s|x)}{Q(s|x)} \]

\[ = D(P(s|x)||Q(s|x)), \quad (7) \]

where \( D(P(s|x)||Q(s|x)) \) is the conditional relative entropy between the approximate conditional distribution \( Q \) and the true conditional distribution \( P \). One may notice that several states of the quasi-arithmetic coder can correspond to a given state \( c \) of the binary source model. Hence, several different approximations \( Q(s|x) \) of \( P(s|x) \) can exist (see, e.g., the states 0 and 3 of Table 3). The excess rate can be considered as a measure of the bit stream redundancy.

7. ESTIMATION ALGORITHM

The above models of dependencies can be exploited to help the estimation of the bit stream (hence of the symbol sequence). The MAP estimation criterion corresponds to the optimal Bayesian estimation of a process \( X \), given available measurements \( Y \):

\[ \hat{X} = \arg \max_x P(X=x|Y). \quad (8) \]

However, if the mean square error (MSE) is the performance measure, the MAP criterion is suboptimal. The conditional mean or minimum MSE (MMSE) is in this case the optimal decoder. The decoder then seeks a sequence of symbol reproductions that will minimize the expected distortion, given the sequence of observations denoted by \( E[D(A,A)|Y] \). These expected distortions can be computed in a very straightforward way, given the MAP estimates, provided the probability measures on the sequence of binary symbols \( S \) are converted into probability measures on the sequence of \( M \)-ary symbols \( A \). In Section 10, only the MAP estimates have been considered.

The optimization is performed over all possible sequences \( x \). This applies directly to the estimation of the hidden states of the processes \( (X,N) \) (symbol-clock model of source + coder) and \( (X,K) \) (bit-clock model of source + decoder), given the sequence of measurements. The estimation is run on the source-decoder product model in order to avoid handling the \( n \) scl variable.

Estimating the set of hidden states \( (X,K) = (X_1,K_1) \cdots (X_N,K_N) \) is equivalent to estimating the associated sequence of decoded symbols \( S = S_1 \cdots S_K \cdots S_K \), given the measurements \( Y_{1:N} \) at the output of the channel. The best sequence \( (X,K) \) can be obtained from the local probabilities on the pairs \( (X_n,K_n) \) by the equation

\[ P(X,K|Y) = \prod_{n=1}^{N} P(X_n,K_n|Y). \quad (9) \]

The computation of the entity \( P(X_n,K_n|Y) \) can be organized around the factorization

\[ P(X_n,K_n|Y) \propto P(X_n,K_n|Y^n) \cdot P(Y_{n+1}|X_n,K_n), \quad (10) \]
where $\propto$ denotes a renormalization factor. The Markov property allows a recursive computation of both terms of the right-hand side, using the BCJR algorithm [21]. The forward sweep concerns the first term

$$P(X_n = x_n, K_n = k_n | Y^n) = \sum_{(x_{n-1}, k_{n-1})} P(X_{n-1} = x_{n-1}, K_{n-1} = k_{n-1} | Y_{i-1}^{n-1})$$

$$\cdot P(X_n = x_n, K_n = k_n | X_{n-1} = x_{n-1}, K_{n-1} = k_{n-1})$$

$$\cdot P(U_n = u(x_{n-1}, k_{n-1}) | (x_n, k_n), Y_n).$$

The terms on the right-hand side of the equation are, respectively, the recursive term, the transition probability given by the product model (see Section 6), and the probability to have emitted the bit $U_n$ triggering the transition between $X_{n-1} = x_{n-1}$ and $X_n = x_n$, given the measure $Y_n$ (channel model). The process is initialized at the starting state $(0, 0)$ and allows to compute $P(X_n, K_n | Y^n)$ for all possible states $(x_n, k_n)$ and for each bit-clock instant $n = 1, \ldots, N$.

The backward sweep provides the second term in (10)

$$P(Y_{n+1}^N | X_n = x_n, K_n = k_n)$$

$$= \sum_{(x_{n+1}, k_{n+1})} P(Y_{n+2}^N | X_{n+1} = x_{n+1}, K_{n+1} = k_{n+1})$$

$$\cdot P(X_{n+1} = x_{n+1}, K_{n+1} = k_{n+1} | X_n = x_n, K_n = k_n)$$

$$\cdot P(U_{n+1} = u(x_{n+1}, k_{n+1}) | (x_n, k_n), Y_{n+1}).$$

The process is initialized for all possible “last” states $(x_N, k_N)$ and allows to compute $P(Y_{n+1}^N | X_n, K_n)$ for all possible states $(x_n, k_n)$ and for each bit-clock instant consecutively from $N$ to 1.

As in [2, 5], a termination constraint can be introduced; one can ensure that the decoder produces the right number of symbols ($K_N = K$) (if known). All the paths in the trellis which do not lead to a valid sequence length are suppressed. The trellis on which the estimation is performed can be precomputed, with all transitions and outputs stored. Figure 9 shows the trellis computed with the example product model of Figure 7 for a sequence of $K = 7$ symbols producing $N = 6$ bits.

8. SOFT SYNCHRONIZATION

Termination constraints mentioned in Section 7 can be regarded as means to force synchronization at the end of the sequence; they indeed constrain the decoder to have the right number of symbols ($K_N = K$) (if known) after decoding the estimated bit stream $U$. These constraints ensure synchronization at the end of the bit stream, but do not ensure synchronization in the middle of the sequence. One can introduce extra information specifically to help the resynchronization “in the middle” of the sequence. For this, we consider here the introduction of extra bits at some known positions $I_n = \{i_1, \ldots, i_j\}$ in the symbol stream. This extra information takes the form of dummy binary symbols (in the spirit of the techniques described in [11, 12, 14, 17, 18]) inserted in the binary symbol stream at some known symbol-clock positions after the conversion of the M-ary source into the binary source. Since these dummy symbols are inserted at some known symbol-clock instants, the position of the corresponding extra bits in the coded bit stream depends on the sequence of symbols encoded, hence is random.

Models and algorithms above have to account for this extra information. Inserting an extra dummy symbol at known positions in the symbol stream amounts to add a section with deterministic transitions in the binary tree model of the source. The presence of this extra information can be exploited by the estimation. During the estimation process, the variable $K_n$ indicates when a marker should be expected. The corresponding transition probabilities in the estimation trellis are updated accordingly. A null probability is given to all transitions that do not emit the expected sequence of binary symbols, while a probability of one is set to the others. Therefore, some paths in the estimation trellis become forbidden and can be suppressed, leading to a reduction of the number of states.

9. ITERATIVE CC-AC DECODING ALGORITHM

The soft synchronization mechanism described above increases significantly the reliability of the segmentation and estimation of the sequence of symbols. One can however consider, in addition, the usage of an error correction code, for example, of a systematic convolutional CC. Both codes can be concatenated in the spirit of serial turbo codes. Adopting this principle, one can therefore work on each model (quasi-arithmetic coder and channel coder) separately and design an iterative estimator, provided an interleaver is introduced between the models. The structure of dependencies between the variables involved is outlined in Figure 10.

Such a scheme requires extrinsic information on the bits $U_n$ to be transmitted by the CC to the soft arithmetic decoder and reciprocally. The extrinsic information on a bit $U_n$ represents the modification induced by the introduction of the rest of the observations $Y_{i-1}^n$, $Y_{n+1}^N$ on the conditional law of $U_n$ given $Y_n$. The extrinsic information can be expressed as

$$\text{Ext}_{U_n}(Y | Y_n) \propto \frac{P(U_n | Y)}{P(U_n | Y_n)}.$$  

The iterative estimation proceeds by first running a BCJR algorithm on the channel coder model. The extrinsic information on the useful bits $U_n$ is a direct subproduct of the BCJR algorithm. These measurements can in turn be used as input for the estimation run on the quasi-arithmetic decoder model described above.

The result of the quasi-arithmetic soft decoding procedure is the a posteriori probabilities on the states $(X_n, K_n)$ of the estimation trellis, given the observations $P(X_n, K_n | Y)$. These a posteriori probabilities must then be converted into extrinsic information on bits $U_n$. The probability of having a
bit \( U_n = u_n \) given \( Y \) is the sum of the transition probabilities between all states \((x_{n-1}, k_{n-1})\) and \((x_n, k_n)\) for which this transition exists and is triggered by \( u_n \). Thus, the a posteriori probability of a bit \( U_n \) given the observations is obtained by the equation

\[
P(U_n = u_n | Y) \mid u_n = 0,1 = \sum_{(x_{n-1}, k_{n-1})} \mathbb{P}(x_{n-1}, k_{n-1} | Y) \cdot \frac{\mathbb{P}(\text{succ}_{u_n}(x_{n-1}, k_{n-1}) | Y)}{\sum_{u_n=0,1} \mathbb{P}(\text{succ}_{u_n}(x_{n-1}, k_{n-1}) | Y)},
\]

where \( \text{succ}_{u_n}(x_{n-1}, k_{n-1}) \) is the state \((x_n, k_n)\) reached by the transition from \((x_{n-1}, k_{n-1})\) triggered by the bit \( U_n = u_n \).

**10. EXPERIMENTAL RESULTS**

To evaluate the performance of the soft quasi-arithmetic decoding procedure, a set of experiments has been performed on a first-order Gauss-Markov source, with zero-mean, unit-variance, and different correlation factors \( \rho \). The source is quantized with an eight-level uniform quantizer (3 bits) on the interval \([-3, 3]\). We consider sequences of \( K = 200 \) symbols with different source correlation factors. All the simulations have been performed assuming an additive white Gaussian channel with a binary phase shift keying (BPSK) modulation. The results are averaged over 3000 realizations.

The first experiment aimed at comparing the performances in terms of soft decoding of Huffman codes [7],
arithmetic codes [19], and quasi-arithmetic codes with $T = 4$
for comparable overall rates. When the source correlation in-
creases, the compression efficiency of the arithmetic coder
increases; soft synchronization patterns are inserted in the
arithmetically encoded stream up to a comparable overall
rate. Figures 11 and 12 show the residual SERs and SNR ob-
tained for different channel $E_b/N_0$ for, respectively, $\rho = 0.5$
and $\rho = 0.9$. For sources with low correlation ($\rho = 0.5$
and under) and for values of $E_b/N_0$ lower than 5 dB (i.e., for
high bit error rates), quasi-arithmetic coding outperforms
both arithmetic and Huffman coding. However, for sources
with high correlation, the quasi-arithmetic coder and de-
coder turn out to be slightly less efficient than the optimal
arithmetic coder for this range of $E_b/N_0$. The reason is that
excess rate induced by quasi-arithmetic coding increases with
the source correlation (see Section 4.3). The optimal arith-
metic coding fully exploiting the source correlation, one can
then insert a higher amount of soft synchronization pat-
terns, for the same overall rate, resulting in an improved
error resilience. The same trend has been observed for dif-
erent rates. The gain brought on soft quasi-arithmetic de-
coding by synchronization markers is illustrated in Figures
13 and 14, respectively, for $\rho = 0.5$ and $\rho = 0.9$. Notice
that, even for high correlation sources, the performances of
the quasi-arithmetic decoder would obviously increase if one
would allow a higher complexity, that is, a higher value for
the parameter $T$. In order to compare fairly the three me-
thods, one must also consider their complexity. It can be mea-
sured by the size of the trellis used for the estimation. Con-
sidering a sequence of 200 symbols, soft decoding of Huff-
mans codes needs two trellises with about 900 states, respec-
tively, per bit-clock and symbol-clock instants. The complex-
ity of soft decoding of arithmetic codes is limited to 100 states
per symbol-clock instant, using pruning. Finally, the soft de-
coding of quasi-arithmetic codes leads to a trellis containing
about 2400 states per bit-clock instant, hence being the most
complex of the three. Pruning may be considered to reduce
this complexity.

The second experiment aimed at evaluating the perfor-
maance of the iterative channel/quasi-arithmetic decoding al-
gorithm. Figures 15 and 16 depict the SER and SNR per-
formances in comparison with the decoding approach with
soft synchronization, respectively, for $\rho = 0.5$ and $\rho = 0.9$.
The first observation is that the iterations bring significant
improvements and higher gain is being obtained when the
source correlation is high (see Figure 16). Nevertheless, if the
source correlation is low, the soft synchronization outper-
forms the iterative scheme. In this range of channel SNRs,
the CC cannot correct all the errors, and the desynchro-
ization phenomenon prevails. It is therefore preferable to
dedicate redundancy within the source to fight against these
desynchronizations. In contrast, when the source correlation
is high ($\rho = 0.9$), the SER is lower with the iterative solu-
tion due to a proper exploitation of the intersymbol corre-
lation for segmenting and estimating the bit stream. It can
also be observed that the gain brought by the iterations de-
pends on the degree of redundancy present on both sides of
the interleaver. Figures 17 and 18 show the performances ob-
tained when combining synchronization markers and chan-
nel coding, respectively, for $\rho = 0.5$ and $\rho = 0.9$. Three ap-
proaches are compared, with equal or sufficiently close over-
all rates. In the first one, only channel coding ($k/n = 2/3$
and 4 iterations) is used to add redundancy. In the second
one, channel coding ($k/n = 5/6$ and 4 iterations) is com-
bined with synchronization markers. Finally, in the third ap-
proach, only synchronization markers are used. For sources
with high correlation, channel coding leads to higher per-
formance since the correlation present in the source is al-
ready exploited efficiently to fight against desynchroniza-
tions. For sources with low correlation and for values of
$E_b/N_0$ lower than 3 dB, on the contrary, the combination
of channel coding and synchronization markers has been
found to be the best strategy. It has also been observed that,
in this case, iterations bring a higher gain than in other
cases.

In another set of experiments, the influence of the length
of the sequence on the system performance is considered.
Figures 19 and 20 depict the SER and SNR performances
with three different lengths, respectively, for $\rho = 0.5$ and
$\rho = 0.9$. As expected, the performance decreases when the
length of the sequence increases. This can be explained by
the exploitation of the termination constraint which con-
tributes to the decoder resynchronization. The length of the
sequence has also an influence on the complexity of the de-
coding. Indeed, this complexity depends on the size of the es-
timated trellis, which depends mainly on the excursion, that
is, the possible values, of the variable $K_n$. The excursion is
higher in the middle than at the extremities of the sequence.
Hence, it reaches higher values with longer sequences. We
have measured experimentally the average number of states
of the trellis per bit-clock instant $n$. The values obtained are
557, 1177, and 2376, respectively, for sequences of 50, 100,
and 200 symbols. Pruning techniques may be required for
longer sequences.

In the last experiment, the impact of the precision of the
quasi-arithmetic coder parameterized by the variable $T$ has
been analyzed. Figure 21 provides the SER and SNR per-
formances for $T = 4$ and $T = 8$ without extra redundancy, and
for $T = 8$ with soft synchronization patterns added to obtain
a bit rate comparable to the case where $T = 4$ ($\rho = 0.5$). The
lower precision coder ($T = 4$) due to the presence of residual
redundancy is more resilient to errors. However, the coder
with a higher precision ($T = 8$) allows, by better exploiting
the source statistics, to obtain higher compression efficiency
and in turn dedicate redundancy to resynchronize the decod-
ing process. Then the overall performances appear to be sim-
ilar for $E_b/N_0$ larger than 4 dB. Once again, the complexity is
an issue. Indeed, increasing the value of $T$ leads necessarily to
a higher number of states. In this experiment, we have found
the following average number of states of the estimated trellis
per bit-clock instant: 1193 when $T = 4$, 4736 when $T = 8$,
and 4297 when $T = 8$ with the insertion of synchronization
markers. Therefore, it is highly desirable to choose $T$ as small
as possible.
Figure 11: SER and SNR performances of (i) soft Huffman decoding, (ii) soft arithmetic decoding, and (iii) soft quasi-arithmetic decoding ($\rho = 0.5$, 200 symbols, 3000 channel realizations).

Figure 12: SER and SNR performances of (i) soft Huffman decoding, (ii) soft arithmetic decoding, and (iii) soft quasi-arithmetic decoding ($\rho = 0.9$, 200 symbols, 3000 channel realizations).
Figure 13: SER and SNR performances of soft quasi-arithmetic decoding for different rates ($\rho = 0.5$, 200 symbols, 3000 channel realizations).

Figure 14: SER and SNR performances of soft quasi-arithmetic decoding for different rates ($\rho = 0.9$, 200 symbols, 3000 channel realizations).
Figure 15: SER and SNR performances of (i) soft quasi-arithmetic decoding with soft synchronization and (ii) turbo quasi-arithmetic/channel decoding ($\rho = 0.5$, 200 symbols, 3000 channel realizations).

Figure 16: SER and SNR performances of (i) soft quasi-arithmetic decoding with soft synchronization and (ii) turbo quasi-arithmetic/channel decoding ($\rho = 0.9$, 200 symbols, 3000 channel realizations).
Figure 17: SER and SNR performances of (i) turbo quasi-arithmetic/channel decoding, (ii) turbo quasi-arithmetic/channel decoding with soft synchronization, and (iii) soft quasi-arithmetic decoding with soft synchronization ($\rho = 0.5$, 200 symbols, 1500 channel realizations).

Figure 18: SER and SNR performances of (i) turbo quasi-arithmetic/channel decoding, (ii) turbo quasi-arithmetic/channel decoding with soft synchronization, and (iii) soft quasi-arithmetic decoding with soft synchronization ($\rho = 0.9$, 200 symbols, 1500 channel realizations).
Figure 19: SER and SNR performances of soft quasi-arithmetic decoding for, respectively, 50, 100, and 200 symbols ($\rho = 0.5$, 3000 channel realizations).

Figure 20: SER and SNR performances of soft quasi-arithmetic decoding for, respectively, 50, 100, and 200 symbols ($\rho = 0.9$, 3000 channel realizations).
11. CONCLUSION

Arithmetic codes are becoming more and more popular in practical compression systems and emerging standards. Their well-known drawback is however their very high sensitivity to noise. MAP estimators running on the coding tree can help to fight against errors and possible decoder desynchronization but at the expense of rather high complexity. The coding tree grows exponentially with the number of symbols in the sequence to be coded. Here, we have considered an alternate solution based on reduced-precision arithmetic codes, called quasi-arithmetic codes. A quasi-arithmetic coder can be viewed as a finite-state stochastic automaton. One can then run MAP estimators on the resulting model. For the sake of clarity, we have considered simple source models in the examples. The results reported have been obtained considering an order-1 Markov source. However, the approach extends very easily to higher-order source models. The state model of the coding and decoding process is of finite size. Its size depends on the acceptable approximation of the source distribution. The decoding complexity remains within a realistic range without the need for applying any pruning. Placed in an iterative decoding structure in the spirit of serially concatenated turbo codes, the estimation process can then benefit from the iterations. Overall, the flexibility they offer for adjusting compression efficiency, complexity, and error resilience allows an optimal adaptation to various transmission conditions and terminal capabilities. Notice that, for low complexity, a very good trade-off compression-noise resilience can be achieved with quasi-arithmetic codes for low correlation sources. This emphasizes the interest of the above solution for practical systems, where the coder is applied on quantized decorrelated sequences of symbols.

REFERENCES


