

Joint source-channel decoding of quasi-arithmetic codes

Thomas Guionnet ¹, Christine Guillemot

Abstract

This paper addresses the issue of robust and joint source-channel decoding of quasi-arithmetic codes. Quasi-arithmetic coding is a reduced precision and complexity implementation of arithmetic coding. This paper provides first a state model of a quasi-arithmetic 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 de-synchronization. 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. Experimental results reveal improved SER and SNR performances against Huffman and optimal arithmetic codes.

1 Introduction

Entropy coding, producing variable length codewords (VLC), 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. 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 “de-synchronization” effect as well as the residual bit and symbol error rates [9]. Models incorporating both VLC-encoded sources and channel codes (CC) have also been considered [8, 6].

The research effort has been first focused on Huffman codes [9, 8, 2]. However, arithmetic codes have gained increased popularity in practical systems, including JPEG2000, H.264 and MPEG-4 standards. The counterpart to their 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 re-augmenting the redundancy of the bitstream, either by introducing an error correcting code or by inserting dedicated patterns in the chain. Along those lines, the author in [4] 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 [3, 11]. Sequential decoding of arithmetic codes is investigated in [10] for supporting error correction capabilities. A soft decoding procedure is described in [5]. However, one difficulty comes from the fact that the codetree, hence the state space dimension, or 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

¹All authors are with IRISA-INRIA, Campus de Beaulieu, 35042 Rennes Cedex, FRANCE. E-mail:firstname.lastname@irisa.fr, Phone:+33 (0) 2 99 84 72 62, Fax:+33 (0) 2 99 84 71 71

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 (QA) coding* has been introduced in [7]. It operates on an integer interval $[0, T[$ and on integer sub-divisions 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 pre-computed and table lookups be in turn substituted for arithmetic operations. A QA 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 QA coding and decoding processes for M -ary sources. In order to maintain the coder and decoder complexity within a tractable range, with a good source distribution approximation, the M -ary source is first mapped into a binary source. The corresponding conversion 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 QA coder. The design of an efficient estimation procedure based on the BCJR algorithm [1] 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 QA codes allows to dedicate extra redundancy (short "soft" synchronization patterns) specifically to decoder re-synchronization, resulting in significantly higher error resilience. The usage of channel codes 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 the channel decoder and the QA 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 symbol error rate (significantly lower than what can be obtained with Huffman codes). Overall, the great flexibility QA codes offer for adjusting compression efficiency, error resilience and complexity, allows an optimal adaptation to various transmission conditions and terminal capability requirements.

2 Notations and problem statement

Let $A = A_1 \dots A_L$ be a sequence of quantized source symbols taking their values in a finite alphabet \mathcal{A} composed of $M = 2^q$ symbols, $\mathcal{A} = \{a_1, a_2, \dots, a_i, \dots, a_M\}$. The sequence $A = A_1 \dots A_L$ is assumed to form an order-one Markov chain. This sequence of M -ary symbols is converted into a sequence of binary symbols $S = S_1 \dots S_K$, where $K = q \times L$. This binary source is in turn coded into a sequence of information bits $U = U_1 \dots U_N$, by means of a QA coder. The length N of the information bit stream is a random variable, function of S , hence in turn of A . The bitstream 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 $X_u^v = \{X_u, X_{u+1}, \dots, X_v\}$.

3 Arithmetic and quasi-arithmetic coding

The arithmetic coding principle consists in the recursive subdivision of the unit interval proportionally to the input symbols probabilities. The final unit interval subdivision obtained when all the input symbols are processed corresponds to the probability of the sequence. Any number in this interval can be used to represent the sequence. Since the interval representing the sequence can be very small, practical implementations include techniques that allows to avoid numerical precision overflow. The reader is referred to [7] for a more detailed treatment of arithmetic coding.

Error resilient decoding solutions can be designed [5], however their complexity can be an issue in some contexts. The coding process can indeed be modeled under the form of a stochastic automaton. However, the number of states grows exponentially with the number of symbols being encoded. It is observed in [7] that controlled approximations can reduce the number of possible states without significantly degrading compression performance. All state transitions and outputs can then be pre-computed and table lookups can be used instead of arithmetic operations. This fast, but reduced precision, implementation of arithmetic coding is called *quasi-arithmetic (QA) coding* [7]. Instead of using the real interval $[0, 1[$, QA 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 sub-divisions will follow closely the distribution of the source. In contrast, if T is small, all the interval sub-divisions can be pre-computed.

Given the M -symbol alphabet \mathcal{A} , the sequence of symbols A_1^L is translated into a sequence of bits U_1^N by an M -ary decision tree. This tree can be regarded as an automaton that models the bitstream 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 length. Successive branching on the tree (or transitions between states) follow the distribution of the source ($\mathbb{P}(A_l|A_{l-1})$ for an order one Markov source, or $\mathbb{P}(A_l)$ in the zero-th order case). Let X_l denote the state of the automaton at each symbol instant l . The state X_l of the QA coder is defined by three variables : $lowA_l$, upA_l and $nschl$. The terms $lowA_l$ and upA_l denote the bounds of the subinterval resulting from successive sub-divisions of the interval $[0, T[$ triggered by the encoding of the sequence A_1^l . The quantity $nschl$ is used to manage the numerical precision issue. Since there is a finite number of possible integer sub-divisions of the interval $[0, T[$, all the possible states of the QA coder can be pre-computed without knowledge of the source.

Similarly, the QA decoder can be expressed in the form of an automaton. Let X_n be its state at bit instant n . X_n stores the four variables $[lowU_n, upU_n[$ and $[lowA_{L_n}, upA_{L_n}[$ denoting respectively the intervals defined by the received bits and the decoded symbols [7]. There is a finite number of states for the QA decoder which can be pre-computed. Tab. 1 gives the states, transitions and outputs of the QA decoder for a binary source and $T = 4$, with MPS and LPS denoting respectively the most and least probable symbols. The decoder in this particular example has two states. Further sub-divisions that will lead to transitions to next states, are function of the source probability distribution (e.g. $\mathbb{P}(MPS)$ in Tab. 1). They are chosen in such a way that the corresponding distribution approximation will minimize the excess rate [7]. Let us assume, for example, that the automaton is in state $X_n = 0$ (defined

by the two intervals $[lowU_n, upU_n[= [0, 4[$ and $[lowA_{L_n}, upA_{L_n}[= [0, 4[$, 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 sub-divided into $[0, 3[$ (if MPS probability is higher than 0.63) or into $[0, 2[$ (if MPS probability is lower than 0.63), both sub-divisions resulting into the state 0.

state X_n	state variables	$\mathbb{P}(MPS)$ (corresponding sub-division of $[lowA_{L_n}, upA_{L_n}[$)	$U_n = 0$		$U_n = 1$	
			out	next	out	next
0 (start)	$[lowU_n, upU_n[: [0, 4[$ $[lowA_{L_n}, upA_{L_n}[: [0, 4[$	$0.63 \leq \mathbb{P}(MPS)$ $([0, 3[)$ $0.37 \leq \mathbb{P}(MPS) < 0.63$ $([0, 2[)$	MPS, MPS MPS	0 0	- LPS	1 0
1	$[lowU_n, upU_n[: [2, 4[$ $[lowA_{L_n}, upA_{L_n}[: [0, 4[$	$0.63 \leq \mathbb{P}(MPS)$ $([0, 3[)$	MPS, LPS	0	LPS	0

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

4 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 [7]. This motivates a conversion of the M -ary source into a binary source, to be then encoded by the QA coder. This conversion amounts to a fixed length binary coding of the source.

Let us first assume that the source quantized on $M = 2^q$ symbols is an order-0 Markov source. It can then be represented by a binary tree of depth q , as shown in Fig. 1-a 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 a three states automaton. In Fig. 1-b, this automaton is shown in the form of a trellis, with the probability of each bit indicated on the corresponding transition and in the particular case of an order-0 Markov source with $M = 4$. 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 ν of the automaton after k binary symbols have been produced. The sequence C_0, \dots, C_K is a Markov chain, and the resulting sequence of binary symbols is a function of transitions of this chain, i.e. $S_k = \phi(C_{k-1}, C_k)$.

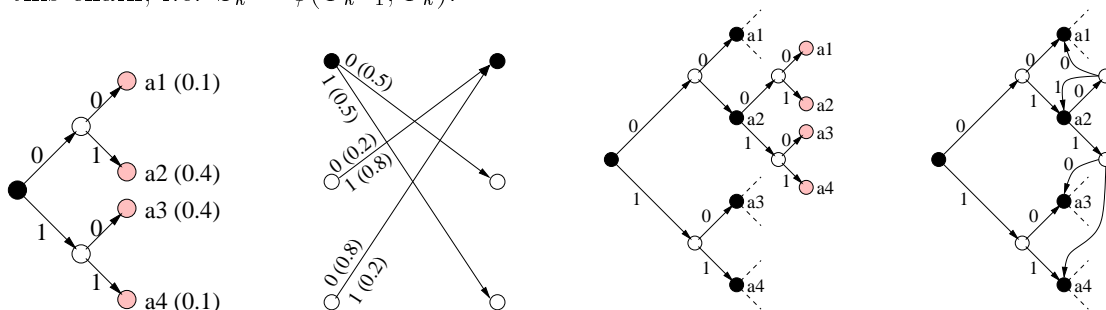


Figure 1: Graphical representation under the form of a a)-tree, b)-trellis of the *binary* model of the order-0 Markov source. c)- tree for the order-1 Markov source d)- automaton for the order-1 Markov source.

Let us 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 (ν, σ) where σ is the value of the last completed symbol

A_l , and ν is the current state of the stochastic automaton describing the construction of the next M -ary symbol, following $\mathbb{P}(A_{l+1}|A_l)$. Alternately, one can take into account the conditional distribution $\mathbb{P}(A_{l+1}|A_l)$, by connecting $M + 1$ trees of depth q as shown in Fig. 1-c, and define the state variable C_k as a node ν in the resulting tree. The complete model of the source is then obtained by additional transitions from leaves to intermediate nodes as shown in Fig. 1-d. 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 $C_1 \dots C_K$ is a Markov process, the transitions of which produce the sequence S of binary symbols.

5 Modeling bitstream 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 bitstream dependencies. A product model of source + decoder can thus be constructed. 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 $X_n = (lowU_n, upU_n, lowS_{K_n}, upS_{K_n}, C_{K_n})$. The state space dimension depends on the coder precision parametrized by T and of the source model. An example of product model is given in Fig. 2

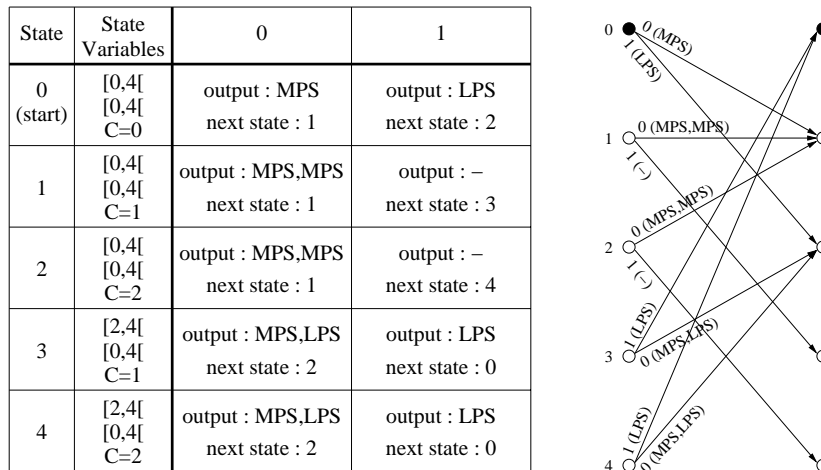


Figure 2: Source-decoder product model: a) states, outputs, transitions; b) trellis representation.

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 $(X, K) = (X_1, K_1) \dots (X_N, K_N)$. Using this model, a sequence of bits U_1^N is translated into a sequence of symbols $S_1^{K_N}$, 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 $\bar{S}_{n+1} = S_{K_n+1}^{K_{n+1}}$ 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
$$\prod_{k=K_n+1}^{K_{n+1}} \mathbb{P}(S_k|C_{k-1}).$$

6 Estimation algorithm

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

$$\hat{X} = \arg \max_x \mathbb{P}(X = x|Y). \quad (1)$$

The optimization is performed over all possible *sequences* x . This applies directly to the estimation of the hidden states of the process (X, K) given the sequence of measurements.

Estimating the set of hidden states $(X, K) = (X_1, K_1) \dots (X_N, K_N)$ is equivalent to estimating the associated sequence of decoded symbols $S = S_1 \dots S_{K_n} \dots S_{K_N}$, 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

$$\mathbb{P}(X, K|Y) = \prod_{n=1}^N \mathbb{P}(X_n, K_n|Y). \quad (2)$$

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

$$\mathbb{P}(X_n, K_n|Y) \propto \mathbb{P}(X_n, K_n|Y_1^n) \cdot \mathbb{P}(Y_{n+1}^N|X_n, K_n), \quad (3)$$

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 [1]. The forward sweep concerns the first term

$$\begin{aligned} \mathbb{P}(X_n = x_n, K_n = k_n|Y_1^n) &= \sum_{(x_{n-1}, k_{n-1})} \mathbb{P}(X_{n-1} = x_{n-1}, K_{n-1} = k_{n-1}|Y_1^{n-1}) \cdot \\ &\quad \mathbb{P}(X_n = x_n, K_n = k_n|X_{n-1} = x_{n-1}, K_{n-1} = k_{n-1}) \cdot \\ &\quad \mathbb{P}(U_n = u_{(x_{n-1}, k_{n-1})(x_n, k_n)}|Y_n). \end{aligned} \quad (4)$$

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 5), 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 $\mathbb{P}(X_n, K_n|Y_1^n)$ for all possible states (x_n, k_n) , and for each bit clock instant $n = 1, \dots, N$.

The backward sweep provides the second term in Eqn. 3

$$\begin{aligned} \mathbb{P}(Y_{n+1}^N|X_n = x_n, K_n = k_n) &= \sum_{(x_{n+1}, k_{n+1})} \mathbb{P}(Y_{n+2}^N|X_{n+1} = x_{n+1}, K_{n+1} = k_{n+1}) \cdot \\ &\quad \mathbb{P}(X_{n+1} = x_{n+1}, K_{n+1} = k_{n+1}|X_n = x_n, K_n = k_n) \cdot \\ &\quad \mathbb{P}(U_{n+1} = u_{(x_n, k_n)(x_{n+1}, k_{n+1})}|Y_{n+1}). \end{aligned} \quad (5)$$

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

7 Soft synchronization

Termination constraints mentioned in section 6 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 \hat{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 $I_s = \{i_1, \dots, i_s\}$ in the symbol stream. This extra information takes the form of dummy binary symbols (in the spirit of the techniques described in [3, 10, 4, 11]) 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 bitstream 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.

8 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, e.g. of a systematic convolutional channel code (CC). Both codes can be concatenated in the spirit of serial turbo codes. Adopting this principle, one can therefore work on each model (QA coder and channel coder) separately and design an iterative estimator, provided an interleaver is introduced between the models.

Such a scheme requires extrinsic information on the bits U_n to be transmitted by the CC to the soft QA 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_1^{n-1}, Y_{n+1}^N on the conditional law of U_n given Y_n . The extrinsic information can be expressed as

$$Ext_{U_n}(Y|Y_n) \propto \frac{\mathbb{P}(U_n|Y)}{\mathbb{P}(U_n|Y_n)}. \quad (6)$$

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 QA decoder model described above.

The result of the QA soft decoding procedure is the a-posteriori probabilities on the states (X_n, K_n) of the estimation trellis, given the observations : $\mathbb{P}(X_n, K_n|Y)$. 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 whose 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

$$P(U_n = u_n|Y)|_{u_n=0,1} = \sum_{(x_{n-1}, k_{n-1})} \mathbb{P}(x_{n-1}, k_{n-1}|Y) \cdot \frac{\mathbb{P}(succ_{u_n}(x_{n-1}, k_{n-1})|Y)}{\sum_{u_n=0,1} \mathbb{P}(succ_{u_n}(x_{n-1}, k_{n-1})|Y)}, \quad (7)$$

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

9 Experimental results

To evaluate the performance of the soft QA 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 ρ . The source is quantized with an 8 levels 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 noise channel with a 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 [6], arithmetic codes [5] and QA codes with $T = 4$, for comparable overall rates. Soft synchronization patterns are inserted in the arithmetically and quasi-arithmetically encoded streams up to an overall rate comparable to Huffman coding. Fig. 3-a shows the residual symbol error rates (SER) obtained for different channel E_b/N_0 , for $\rho = 0.5$. For values of E_b/N_0 lower than 5 dB. (i.e. for high bit error rates), QA coding outperforms both arithmetic and Huffman coding. The same trend has been observed for different rates. The gain brought on soft QA decoding by synchronization markers is illustrated on Fig. 3-b for $\rho = 0.5$.

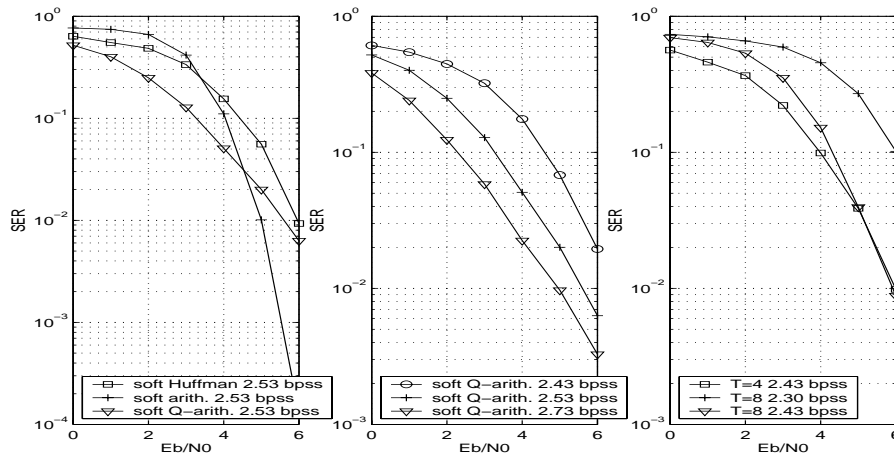


Figure 3: SER performances of (a) soft Huffman decoding, soft arithmetic decoding and soft quasi-arithmetic decoding (b) soft quasi-arithmetic decoding for different rates (c) soft quasi-arithmetic decoding for different precisions T ($\rho = 0.5$, 200 symbols, 3000 channel realizations).

In the second experiment, the impact of the precision of the QA coder parameterized by the variable T has been analyzed. Fig. 3-c provides the SER performances 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 re-synchronize the decoding process. Then the overall performances appear to be similar 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 estimation 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.

The last experiment aimed at evaluating the performance of the iterative channel/QA decoding algorithm. Fig. 4-a and 4-b depict the SER performances, 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, higher gain being obtained when the source correlation is high (see Fig. 4-b). Nevertheless, if the source correlation is low, the soft synchronization outperforms the iterative scheme. In this range of channel signal to noise ratios, the channel code cannot correct all the errors, and the de-synchronization phenomenon prevails. It is therefore preferable to dedicate redundancy within the source to fight against these de-synchronizations. In contrast, when the source correlation is high ($\rho = 0.9$), the SER is lower with the iterative solution due to a proper exploitation of the inter-symbol correlation for segmenting and estimating the bit stream. It can also be observed that the gain brought by the iterations depends on the degree of redundancy present on both sides of the interleaver.

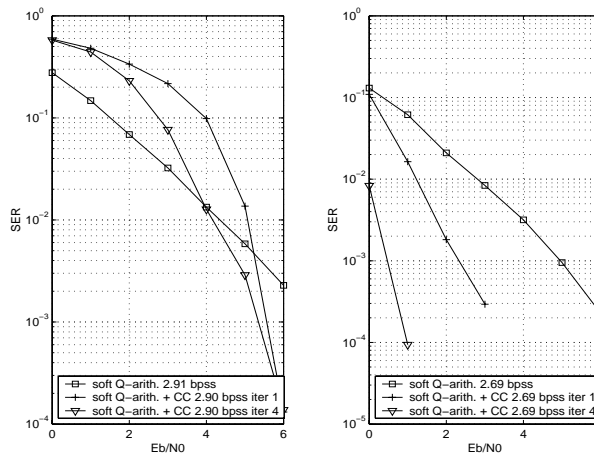


Figure 4: SER performances of soft quasi-arithmetic decoding with soft synchronization and turbo quasi-arithmetic/channel decoding for (a) $\rho = 0.5$ (b) $\rho = 0.9$ (200 symbols, 3000 channel realizations).

10 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 run on the coding tree can help to fight against errors and possible decoder de-synchronization but at the expense of rather high complexity. The coding tree grows exponentially with the number of symbols in the se-

quence 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 sake of clarity, in the examples we have considered simple source models. 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 de-correlated sequences of symbols.

References

- [1] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE trans. on information theory*, 20:284–287, March 1974.
- [2] R. Bauer and J. Hagenauer. Iterative source/channel decoding based on a trellis representation for variable length codes. In *Proc. IEEE Intl. Symposium on Information Theory, ISIT*, page 117, June 2000.
- [3] C. Boyd, J. G. Cleary, S. A. Irvine, I. Rinsma-Melchert, and I. H. Witten. Integrating error detection into arithmetic coding. *IEEE trans. on Communications*, 45(1):1–3, Jan. 1997.
- [4] G. F. Elmasry. Embedding channel coding in arithmetic coding. *IEE proc.-Communications*, 146(2):73–78, April 1999.
- [5] T. Guionnet and C. Guillemot. Soft decoding and synchronization of arithmetic codes : application to image transmission over error-prone channels. *to appear in IEEE trans. on Image Processing*, December 2003.
- [6] A. Guyader, E. Fabre, C. Guillemot, and M. Robert. Joint source-channel turbo decoding of entropy coded sources. *IEEE Journal on Selected Areas in Communication, special issue on the turbo principle : from theory to practice*, 19(9):1680–1696, September 2001.
- [7] P. G. Howard and J. S. Vitter. Design and analysis of fast text compression based on quasi-arithmetic coding. In *Data Compression Conference*, pages 98–107, Snowbird, Utah, March-April 1993.
- [8] A.H. Murad and T.E. Fuja. Joint source-channel decoding of variable length encoded sources. In *Proc. Information Theory Workshop, ITW*, pages 94–95, June 1998.
- [9] M. Park and D.J. Miller. Decoding entropy-coded symbols over noisy channels by MAP sequence estimation for asynchronous HMMs. In *Proc. Conf. on Info. Sciences and Systems*, May 1998.
- [10] B. D. Pettijohn, M. W. Hoffman, and K. Sayood. Joint source/channel coding using arithmetic codes. *IEEE trans. on Communications*, 49(5):826–836, May 2001.
- [11] I. Sodagar, B. B. Chai, and J. Wus. A new error resilience technique for image compression using arithmetic coding. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, Istanbul, Turkey, June 2000.