OPTIMIZING THE SEARCH OF FINITE-STATE JOINT SOURCE-CHANNEL CODES BASED ON ARITHMETIC CODING

A. Diallo[1], C. Weidmann[2], and M. Kieffer[1]

[1] LSS – CNRS – SUPELEC – Univ. Paris-Sud
3 rue Joliot-Curie
91192 Gif-sur-Yvette, France

[2] INTHFT, Vienna University of Technology
Gusshausstrasse 25/389
1040 Vienna, Austria

ABSTRACT

Joint source-channel (JSC) coding is an important alternative to classic separate coding in wireless applications that require robustness without feedback or under stringent delay constraints. JSC schemes based on arithmetic coding can be implemented with finite-state encoders (FSE) generating finite-state codes (FSC). The performance of an FSC is primarily characterized by its free distance, which can be computed with efficient algorithms. This work shows how all FSEs corresponding to a set of initial parameters (source probabilities, arithmetic precision, design rate) can be ordered in a tree data structure. Since an exhaustive search of the code with the largest free distance is very difficult in most cases, a criterion for optimized exploration of the tree of FSEs is provided. Three methods for exploring the tree are proposed and compared with respect to the speed of finding a code with the largest free distance.

1. INTRODUCTION

Arithmetic coding (AC) [15] is an efficient data compression technique whose variants have been used in recent still image (JPEG 2000) and video (H.264/AVC) coders. Nevertheless, AC is particularly vulnerable to errors appearing, *e.g.*, when the compressed bitstream is sent over a noisy channel. This issue has motivated the development of Joint Source-Channel coding schemes based on AC (JSC-AC).

Robustness of AC against transmission errors is usually achieved by introducing some redundancy in the compressed bitstream by means of a forbidden symbol (FS), to which a non-zero probability is given during the partition of the coding interval [3]: the higher the FS probability, the higher the redundancy and the robustness against errors. This idea is extended in [14], which proposes the introduction of multiple forbidden symbols (MFS) (up to three in case of a binary source) and uses the stack sequential decoding algorithm [9] to estimate the encoded sequence from noisy measurements. In [12], both depth-first and breadth-first sequential decoders are used, in conjunction with error detection achieved by testing the presence of a FS in the decoded bitstream. In [5], trellis coded modulation is used jointly with AC, where a FS is exploited to discard erroneous paths during a Viterbi decoding process. Later, a MAP decoder for AC using the FS has been proposed in [6]. A finite state encoder (FSE) inspired from [11, 8] is introduced in [7] to represent quasi-arithmetic coding (QAC) [8], where transitions between states are modeled by a Markov process. Two types of three-dimensional trellises are proposed for decoding, using either a symbol clock or a bit clock. Redundancy is added by limiting the number of states and introducing synchronization markers. Another three-dimensional bit-clock trellis for soft decoding of AC is proposed in [2].

This work was supported by the European Commission in the framework of the FP7 Network of Excellence in Wireless COMmunications NEWCOM++ (contract n. 216715).

All these techniques improve the robustness of AC to transmission errors. Nevertheless, among the various ways to introduce redundancy, it is difficult to assess their efficiency other than experimentally. Optimizing the design of a given JSC-AC is thus quite laborious. To address this problem, an analytical tool for characterizing the effectiveness of the introduction of one or several FS in the case of QAC has been proposed in [1]. The free distance of a JSC-AC is evaluated with polynomial complexity in the number of states of the FSE representing the AC. Distance spectra may also be approximatively evaluated using extensions of one of the two techniques presented in [4]. First JSC-AC optimization attempts are also proposed in [1], assuming that the probabilities allotted to the MFS are constant. Nevertheless, the class of JSC-AC with constant probabilities for the MFS is significantly smaller than that with time-varying probabilities. For a fixed amount of redundancy, the JSC-AC obtained by [1] is thus suboptimal.

The aim of this paper is to *globally* optimize the introduction of time-varying MFS in binary input JSC-AC (JSC-BIAC) represented by a FSE. Here, only the free distance is optimized. The set of all JSC-AC when considering time-varying MFS may be huge, but this paper shows that it has a tree structure, where all possible JSC-AC correspond to leaves of the tree. An efficient branch-and-bound algorithm is introduced to explore this tree and discard nodes as soon as it can be shown that all JSC-AC stemming from a given node cannot have good performances in terms of free distance.

Section 2 briefly recalls the principles of AC. JSC-AC is then introduced in Section 3. The tree structure of the set of JSC-AC with time-varying MFS is then introduced in Section 4, before presenting the branch-and-bound algorithm for exploring this tree in Section 5. First optimization results are provided in Section 6, before drawing some conclusions and perspectives.

2. ARITHMETIC CODING

2.1 Basic principle

The basic idea of binary AC is to assign to every sequence of source symbols a unique subinterval of the unit interval $[0,1)$; then a subinterval of width w is represented by a binary fraction of length at least $\lceil \log_2 w \rceil$ bits. The source entropy can be approached by recursively partitioning the interval $[0,1)$ according to the source symbol probabilities. Let K

be the size of the source alphabet $\{a_1, a_2, \dots a_K\}$. The current *source interval* $[l,h)$ is partitioned into K non-overlapping subintervals $\{I_1, I_2, \dots, I_K\}$, and the width of subinterval I_i is proportional to the probability of the symbol a_i. The subinterval corresponding to the current symbol is then selected as the new source interval. Once the last symbol has been processed, the encoder chooses a value in the current source interval, and its binary representation is associated to the sequence of encoded symbols. For sources with skewed probabilities or for long source sequences, subintervals may however get too small to be accurately handled by a finite-precision computer. This problem is solved by integer AC.

2.2 Binary Integer Arithmetic Coding

Binary integer AC (BIAC) was introduced by Pasco and Rissanen in 1976 [11]. It works like the scheme presented above, but the initial interval is replaced by the integer interval $[0,T)$, where $T = 2^P$ and P is the binary precision (register size) of the encoding device. All interval boundaries are rounded to integers. During the encoding process, the bounds of the current interval $[l,h)$ are renormalized as follows:

- If $h \leq T/2$, l and h are doubled.
- If $T/2 \leq l$, l and h are doubled after subtracting $T/2$.
- If $T/4 \leq l$ and $h \leq 3T/4$, l and h are doubled after subtracting $T/4$.

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 f (for *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 l (0 or 1) and f opposite bits (1 or 0). This is called *follow-on* procedure [15].

2.3 Finite-state integer arithmetic coding

Since the BIAC process can be characterized by $[l,h)$ and f (and the source probabilities), the encoder state may be defined as (l,h,f). If the value of f is bounded, it is possible to precompute all the reachable states and the transitions between them, thus yielding a finite-state encoder (FSE). In general, f will grow without bounds, but it can be easily limited to $f \leq f_{\max}$, as in [2]. The present work takes the approach of [1]: whenever $f = f_{\max}$ and the current source interval is such that f could be further incremented, the symbol probabilities are modified in order to force the follow-on procedure after encoding the current symbol.

A FSE may be represented as a directed graph consisting of vertices (states) and directed edges (transitions). Each transition is labeled with a vector of input symbols and a vector of output bits, one of which may be empty under certain conditions, depending on the chosen representation. In [1], three types of FSE describing the AC operation were considered: a symbol-clock FSE (S-FSE) suited for encoding, where each transition is labeled with exactly one input symbol; a reduced FSE (R-FSE), with variable-length non-empty input and output labels, leading to a compact trellis better suited for decoding; and finally a bit-clock FSE (B-FSE) suited for the evaluation of distance spectra, where each transition is labeled with exactly one output bit. Details on how these FSEs are obtained for a given AC scheme can be found in [1].

3. JOINT SOURCE-CHANNEL ARITHMETIC CODING

In a joint source-channel coding scheme, redundancy is introduced in order to allow error detection and/or correction at the decoder side. In JSC-AC, this redundancy is usually introduced by means of a FS [3], which is never emitted by the source, although a positive probability P_ε is assigned to it during the coding process. On the receiver side, decoding a FS indicates that an error has occurred during transmission. Introducing a FS with probability P_ε adds a redundancy of $-\log_2(1-P_\varepsilon)$ bits/symbol to the coded bitstream. More generally, for a K-ary source the probability P_ε may be split among up to $K+1$ FSs; in the following, only binary ($K=2$) source alphabets will be considered. This FS technique may be applied to BIAC jointly with a bound on f, resulting in a FSE implementing JSC-BIAC.

Given an initial state s_0, the operation of the FSE on all possible (semi-)infinite input sequences can be displayed with a trellis. The concatenation of the output labels on all paths trough the trellis forms a finite-state code (FSC), whose performance is primarily determined by its *free distance* d_{free} (a finer characterization is possible through the *distance spectrum*). If we assume that all states communicate with each other (*i.e.*, any state can be reached from any other in a finite number of transitions)[1], the free distance will be the same for every possible initial state.

More formally, let $\mathscr{S}$ the set of states of the FSE, $\mathscr{T}$ the set of transitions, $\sigma(t)$ the originating state of a transition $t \in \mathscr{T}$ and $\tau(t)$ its target state, $I(t)$ the input label of a transition and $O(t)$ its output label. A path $\underline{t} = (t_1, t_2, \cdots, t_k) \in \mathscr{T}^k$ on the trellis is a concatenation of transitions that satisfy $\sigma(t_{i+1}) = \tau(t_i)$ for $1 \leq i < k$ (this corresponds to a *walk* of length k on the encoder graph); we define $\mathscr{P}^k_{s_0}$ as the set of all paths with k transitions starting in s_0. By extension, we define $\sigma(\underline{t}) = \sigma(t_1)$ and $\tau(\underline{t}) = \tau(t_k)$, as well as $I(\underline{t})$ and $O(\underline{t})$, which are the concatenations of the input, respectively output, labels of $\underline{t}$.

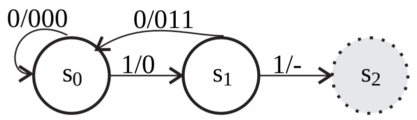
Definition 1 *The FSC $\mathscr{C}_{s_0}$ is the set of all infinite-length output sequences generated by the FSE starting in s_0, $\mathscr{C}_{s_0} = \{O(\underline{t}) : \underline{t} \in \mathscr{P}^\infty_{s_0}\}$*

Remark: The output length $|O(\underline{t})| = \infty$ for all $\underline{t} \in \mathscr{P}^\infty_{s_0}$, since any (JSC-)BIAC is uniquely decodable by construction and thus the FSE graph can have no closed loop with empty output labels.

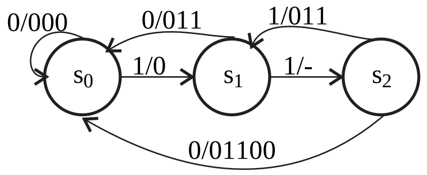
Definition 2 *The free distance of the FSC $\mathscr{C}_{s_0}$ is the minimum Hamming distance between any two distinct code sequences, $d_{\text{free}} = \min_{c_1 \neq c_2 \in \mathscr{C}_{s_0}} d_H(c_1, c_2)$.*

The key insight leading to more efficient code search algorithms is that the free distance on a FSE subgraph upper bounds the free distance of the entire FSE. We define a *complete automaton* (CA) as an S-FSE automaton in which all states have two outgoing transitions (with input labels 0 and 1), that is, an S-FSE which can encode any binary source sequence. An *incomplete automaton* (IA) is a S-FSE automaton with terminal states (without outgoing transitions),

[1]We observed that this holds for almost all JSC-BIACs we examined, the others having a transient component that can be eliminated. See also the remarks in [10, Sec. III.D]).



(a) Incomplete Automaton



(b) Complete Automaton

Figure 1: Example of an incomplete automaton (a) and a complete automaton (b) derived from the automaton (a)

in which encoding stops, see Figure 1. We call these *unexplored* states, since their successor states have not yet been determined.

Definition 3 *The free distance associated to an incomplete S-FSE automaton is the minimum Hamming distance between any two distinct output sequences, which are either infinite, or of equal length and associated to paths ending in the same state (all paths begin in s_0). If there is no pair of paths ending in the same state, the free distance is infinite.*

This definition takes care of unexplored states leading to finite-length prefixes of code sequences, which may only be compared under the conditions mentioned. An IA or CA A_1 is *derived* from an IA A_0 if it has been obtained by exploring (adding the successor states) of one or more terminal states of A_0 (hence the graph of A_0 is a subgraph of that of A_1).

Lemma 1 *Let A_0 an IA with $d_{\text{free}}^{(0)}$ and A_1 an IA or CA derived from A_0 with $d_{\text{free}}^{(1)}$. Then $d_{\text{free}}^{(0)} \geq d_{\text{free}}^{(1)}$.*

Proof: The S-FSEs of A_0 and A_1 satisfy $\mathscr{S}_{A_0} \subset \mathscr{S}_{A_1}$ and $\mathscr{T}_{A_0} \subset \mathscr{T}_{A_1}$. Hence it is obvious that $\mathscr{C}_{A_0} \subset \mathscr{C}_{A_1}$, where the code may also contain finite-length (prefix) sequences. Since the code sequences leading to $d_{\text{free}}^{(0)}$ are contained in $\mathscr{C}_{A_1}$, by definition we have $d_{\text{free}}^{(0)} \geq d_{\text{free}}^{(1)}$. □

4. A TREE OF JSC-BIAC AUTOMATA

In this section, we will first show how all possible automata for given *initial parameters* $(T, f_{\max}, P_0, P_\varepsilon)$ can be generated in the JSC-BIAC case, then introduce how they can be ordered in a *tree of automata*.

Assume that $\mathscr{A} = \{a_0, a_1\}$ is the source alphabet. P_0 and P_1 are the probabilities of occurrence of a_0 and a_1, respectively. (l, h, f) is the current state of the encoder, $w = h - l$ the width of the current interval. $[l_0, h_0)$ is the interval assigned to symbol a_0 and $w_0 = h_0 - l_0$ its width, while $[l_1, h_1)$ is the interval assigned to symbol a_1. and $w_1 = h_1 - l_1$ its width. When JSC-BIAC with a FS is performed, a positive probability P_ε is assigned to the FS although it is never emitted by the source. Let w_ε be the width of the interval assigned to FS. Given the initial parameters P_ε and P_0, the interval widths are

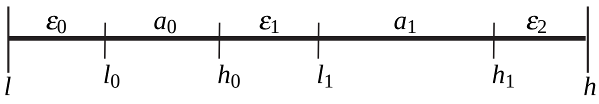


Figure 2: Partitioning the coding interval in the more general JSC-BIAC case

computed as follows:

$$w_\varepsilon = \text{round}(P_\varepsilon \times w), \tag{1}$$
$$w_0 = \text{round}(P_0 \times (h - l - w_\varepsilon)), \tag{2}$$
$$w_1 = h - l - w_0 - w_\varepsilon, \tag{3}$$

where round$(\cdot)$ rounds toward the nearest integer. In the more general case, a set of three FSs $\{\varepsilon_0, \varepsilon_1, \varepsilon_2\}$ may be defined, with corresponding probabilities $\{P_{\varepsilon_0}, P_{\varepsilon_1}, P_{\varepsilon_2}\}$.

Figure 2 shows how the current interval is subdivided during the coding process in the more general JSC-BIAC case. To our best knowledge, no analytical method is known that would allow to find the optimal proportions of the probabilities assigned to $\{\varepsilon_0, \varepsilon_1, \varepsilon_2\}$. In [1], an exhaustive search algorithm over a grid of values $(P_{\varepsilon_0}, P_{\varepsilon_1})$ was proposed, with $P_{\varepsilon_2} = P_\varepsilon - P_{\varepsilon_0} - P_{\varepsilon_1}$ for a given constant P_ε that determines the design code rate. The main characteristic of this approach is that it is *static* with respect to the encoder evolution, *i.e.*, the proportions and the order of the subintervals assigned to source symbols and FSs are the same for every state.

In the present work, we allow the *configuration* of subintervals to be *dynamic*, that is, only the sizes w_0 and w_1 of the subintervals for a_0 and a_1 will be computed as in (2) and (3), but the placement of these subintervals in the interval $[l, h)$ may change from state to state. Thus the probabilities $\{P_{\varepsilon_0}, P_{\varepsilon_1}, P_{\varepsilon_2}\}$ may change with the state, only their sum P_ε remains constant. Like in the static case, no analytic tools are available to determine the configuration yielding the largest d_{free}. Potentially, all encoders for a given set of initial parameters need to be tested in order to maximize d_{free}.

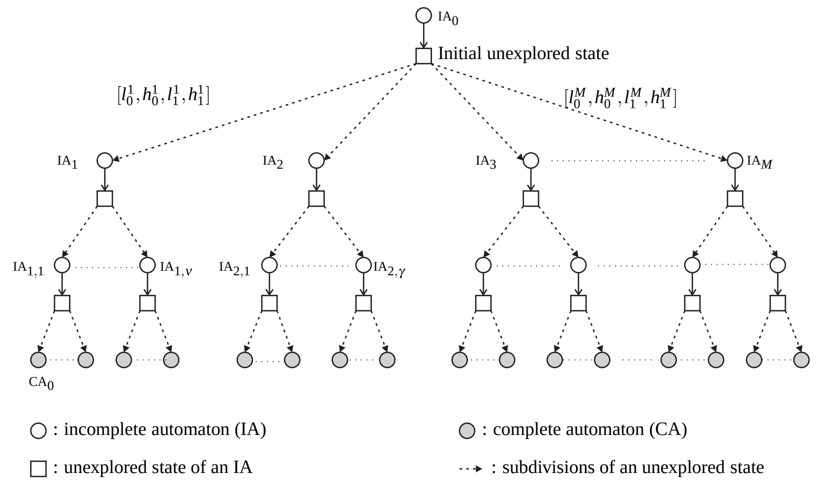
The set of all encoders can be obtained by recursively exploring the successors of all states, starting from the initial state $(l = 0, h = T, f = 0)$, for every admissible configuration of the subintervals of a state. To this end, we let both l_0 and l_1 vary from l to $h - 1$ in steps of one, then check if one of the following two admissibility conditions is satisfied:

$$l \leq l_0 < h_0 = l_0 + w_0 \leq l_1 < h_1 = l_1 + w_1 \leq h, \tag{4}$$
$$l \leq l_1 < h_1 = l_1 + w_1 \leq l_0 < h_0 = l_0 + w_0 \leq h. \tag{5}$$

If that is the case, two new states (obtained from $[l_0, h_o)$ and $[l_1, h_1)$ after applicable renormalizations) are created and explored in turn.

Clearly, the set of automata thus constructed becomes quickly unmanageably large and it is therefore necessary to structure it in a way that allows to exclude (using Lemma 1) large parts from the search space for the largest d_{free}. The set of partially explored states forms an IA, in which unexplored states are terminal states. This suggests using a tree structure, where internal nodes correspond to IA and leaves to CA, and each edge corresponds to the exploration of a terminal state. The children of a given node correspond to all admissible configurations of the state that is being explored.

Figure 3 shows how all the possible automata for given initial parameters form a tree. At the root is the initial IA



IA$_0$
Initial unexplored state
$[l_0^1, h_0^1, l_1^1, h_1^1]$
$[l_0^M, h_0^M, l_1^M, h_1^M]$
IA$_1$
IA$_2$
IA$_3$
IA$_M$
IA$_{1,1}$
IA$_{1,\nu}$
IA$_{2,1}$
IA$_{2,\gamma}$
CA$_0$
: incomplete automaton (IA)
: complete automaton (CA)
: unexplored state of an IA
: subdivisions of an unexplored state

Figure 3: All automata for given initial parameters on a tree

consisting of the sole initial state $(0,T,0)$, which is also a terminal (as yet unexplored) state. The first layer of internal nodes corresponds to M distinct subinterval configurations for the initial state.

Figure 4 shows an example of a tree of automata for the initial parameters $T = 8$, $f_{\max} = 1$, $P_0 = \frac{1}{4}$ and $P_\varepsilon = \frac{1}{2}$. The small circles labeled $s_o, s_1, \ldots$ represent the states of the IA or CA; they are shaded for unexplored (terminal) states. The initial incomplete automaton IA$_0$ consists of the initial state $s_0 = (0,8,0)$ which is a terminal state. On exploration of the initial state, the first configuration assigns the interval $[0,1)$ to symbol a_0 and the interval $[1,4)$ to symbol a_1. This leads to the second incomplete automaton IA$_1$. The last configuration assigns the intervals $[7,8)$ and $[4,7)$ to symbols a_0 and a_1, respectively, yielding the incomplete automaton IA$_M$. The first complete automaton CA$_0$ is shown shaded on the bottom left side of the tree.

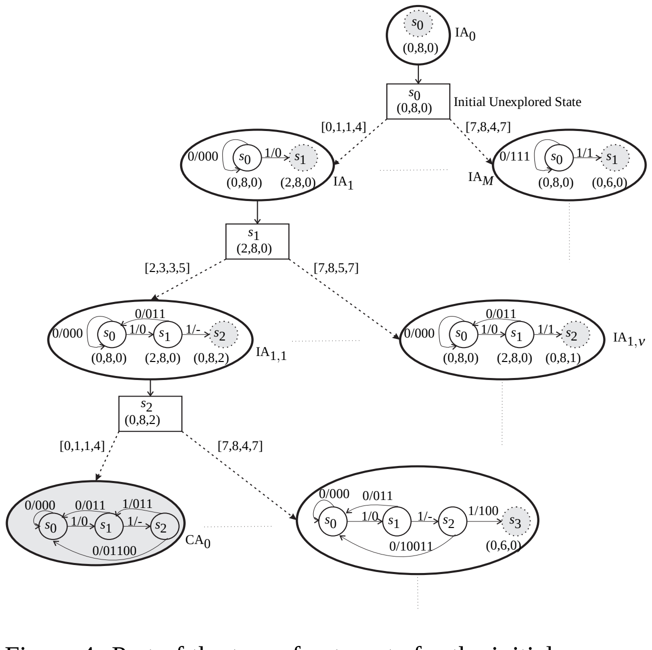


Figure 4: Part of the tree of automata for the initial parameters $T = 8$, $f_{\max} = 1$, $P_0 = \frac{1}{4}$, $P_\varepsilon = \frac{1}{2}$.

5. FINITE-STATE CODE SEARCH ALGORITHM

This section outlines how Lemma 1 can be used in a branch and bound algorithm in order to substantially reduce the time needed to find the best FSC (with largest d_{free}) for given initial parameters. Let IA$_c$ be the current IA to be completed. IA$_c$ is initialized to IA$_0$, which consists of the initial state $(0,T,0)$ as an unexplored state. Notice that $d_{\text{free}}(\text{IA}_0)$ is infinite by definition. The idea of the branch and bound algorithm is to set an initial integer value d_{t} as a threshold value for d_{free}. If $d_{\text{free}}(\text{IA}_c) > d_{\text{t}}$, then an unexplored state of IA$_c$ will be explored. According to the initial parameters, a certain number μ of new IAs or CAs will be created as children of IA$_c$. These automata are indexed IA$_{c,i}$ for $i = \{1,\ldots,\mu\}$. For instance, in Figure 4, for $c = 0$, $\mu = M$, and for $c = 1$, $\mu = \nu$. For $i = \{1,\cdots,\mu\}$, if IA$_{c,i}$ is an IA and $d_{\text{free}}(\text{IA}_{c,i}) > d_t$, then IA$_{c,i}$ is stored in a buffer to be inspected later. Else, if IA$_{c,i}$ is a CA and $d_{\text{free}}(\text{IA}_{c,i}) > d_{\text{t}}$, then the value of d_{t} is updated to $d_{\text{t}} = d_{\text{free}}(\text{IA}_{c,i})$ and IA$_{c,i}$ becomes the new best FSE. Otherwise, IA$_{c,i}$ is discarded, as well as all IAs and CAs derived from it, if there are any. According to Lemma 1, all the IAs and CAs derived from IA$_{c,i}$ will have a d_{free} smaller or equal to $d_{\text{free}}(\text{IA}_{c,i})$, hence they cannot have larger d_{free} than the best FSE found so far.

Then IA$_c$ is discarded and the next IA in the buffer is chosen to be the current IA to be explored. And so on until the buffer becomes empty.

Three ways of exploring the tree are proposed, which depend on the way the newly generated IAs are stored in the buffer. The first method puts the generated IAs at the front of the buffer (Depth-First Exploration). The second stores the generated IAs at the back of the buffer (Breadth-First Exploration). The last method sorts the buffer according to the d_{free} of IAs and chooses the IA with the largest d_{free} to be the current IA to be explored (Sort Method). The comparison between these three methods will be made in the next section.

6. FIRST RESULTS

This section first compares the computational efficiency of the branch and bound algorithm to an exhaustive search for the best FSE. Then we compare the three methods for exploring the tree presented in Section 5.

The first simulation is made with initial parameters $T = 8$, $f_{\max} = 1$, $P_0 = 0.1$, $P_\varepsilon = 0.1$, for which the time needed to exhaustively generate all possible automata and compute their free distances is 1 h 45 min. Using the branch and bound algorithm with Breadth-First Exploration, the time needed to find the largest d_{free} is 25 sec., a time saving of 99.6%.

The second set of simulations compares the three exploration methods for the initial parameters $T = 16$, $f_{\max} = 1$, $P_0 = 0.1$, $P_\varepsilon = 0.26$, for which an exhaustive exploration is unreasonably time-consuming. Table 1 shows the times needed for Depth-First Exploration, Breadth-First Exploration and the Sort Method to find the best automaton (since d_{free} is the only optimization criterion, different automata may be found). $\|\mathscr{S}^{R-FSE}\|$ and $\|\mathscr{T}^{R-FSE}\|$ denote the number of states and the number of transitions of the corresponding R-FSE, respectively. The coding rate is expressed in bits/symbol. It can be seen that the Sort Method is the best method to find the largest d_{free}. This is mainly due to the fact that this method explores first the IA with the highest potential to have a large d_{free}, so that d_{t} may be rapidly increased. Having a large value of d_{t} at the beginning of the algorithm facilitates pruning large parts of the tree being explored.

It would be useful to find a relation between the initial parameters and the computation time for finding the best automaton. However, this is very difficult, since the number of automata generated depends on these parameters in intricate ways. One may compute an upper bound on the number of states per automaton and thus on the number of distinct automata, but this upper bound will likely be too loose to be useful. An additional difficulty resides in estimating the time consumption of the algorithm for computing d_{free} of an automaton. The best such algorithm has linear complexity in the number bits of the output labels of the FSE. Again, this is extremely difficult to estimate from the parameters without building the actual FSE.

Methods	Depth-First	Breadth-First	Sort Method
$\|\mathscr{S}^{R-FSE}\|$	8	2	3
$\|\mathscr{T}^{R-FSE}\|$	28	9	12
d_{free}	3	3	3
Rate	0.93	0.92	0.92
Time	125h 21mn	8h 42mn	3h 27mn

Table 1: Comparison between the three methods to explore the tree for $T = 16$, $\text{P}_0 = 0.1$, $P_\varepsilon = 0.26$

7. CONCLUSION AND PERSPECTIVES

The presented results show that the branch and bound algorithm (using the Sort Method) is a fast way to find the JSC-AC with largest d_{free} for given initial parameters. But compared to an equivalent tandem scheme (AC followed by a Convolutional Code (CC)), d_{free} of the obtained JSC-AC remains suboptimal. For instance, for the example of Table 1, the equivalent tandem scheme uses an AC with $T = 16$, $P_0 = 0.1$, $P_\varepsilon = 0$, followed by a rate 1/2 CC. The free distance of the tandem scheme depends on the constraint length of the CC. For constraint length 2 (3), the best d_{free} of a rate 1/2 CC is 4 (5) [13, Chapter 8]. The weakness of the JSC-AC is mainly due to its small effective memory (which is related to the set of states), that is more geared towards good compression than towards large d_{free}. Therefore, we are studying the extension of the FSE states in JSC-BIAC with a m-bit memory, which can be used to improve d_{free} by separating paths that would lead to small distances. The memory holds an integer $0 \le \lambda \le 2^m - 1$, so that the FSE state can be represented as (l, h, f, λ). The set of FSEs of JSC-AC with memory m contains the set of tandem schemes with CC with constraint length $m+1$. Therefore one may expect to find at least FSEs with performance (compression, d_{free}) equivalent to the tandem schemes, but hopefully less complex (regarding the number of states and transitions).

REFERENCES

[1] S. Ben-Jamaa, C. Weidmann, and M. Kieffer. Analytical tools for optimizing the error correction performance of arithmetic codes. *IEEE Trans. Commun.*, 56(9):1458–1468, Sept. 2008.

[2] D. Bi, W. Hoffman, and K. Sayood. State machine interpretation of arithmetic codes for joint source and channel coding. *Proc. of DCC, Snowbird, Utah, USA.*, pages 143–152, 2006.

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

[4] V. Buttigieg. *Variable-Length Error Correcting Codes*. Phd dissertation, University of Manchester, Univ. Manchester, Manchester, U.K., 1995.

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

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

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

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

[9] F. Jelinek. Fast sequential decoding algorithm using a stack. *IBM J. Res. Develop.*, 13:675–685, 1969.

[10] D. L. Neuhoff and R. K. Gilbert. Causal source codes. *IEEE Trans. Inform. Theory*, IT-28(5):701–713, Sept. 1982.

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

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

[13] J. G. Proakis. *Digital Communications*. Maidenhead, Berkshire, UK, 2001.

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

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