Efficient computation and optimization of the free distance of variable-length finite-state joint source-channel codes

Amadou Diallo, Claudio Weidmann, Michel Kieffer

Efficient computation and optimization of the free distance of variable-length finite-state joint source-channel codes

A. Diallo, C. Weidmann, *Member, IEEE*, and M. Kieffer, *Senior Member, IEEE*

Abstract

This paper considers the optimization of a class of joint source-channel codes described by finite-state encoders (FSEs) generating variable-length codes. It focuses on FSEs associated to joint-source channel integer arithmetic codes, which are uniquely decodable codes by design. An efficient method for computing the free distance of such codes using Dijkstra's algorithm is proposed. To facilitate the search for codes with good distance properties, FSEs are organized within a tree structure which allows the use of efficient branch-and-prune techniques avoiding a search of the whole tree.

Index Terms

Variable length codes, finite state machines, source coding, channel coding, arithmetic codes.

I. Introduction

Today, most communication systems are based on Shannon's separation principle [1], which states that source and channel coding may be optimized separately, without loss of optimality compared to a joint design. However, this result has been obtained under the hypothesis of a stationary channel, which is seldom the case in wireless communication systems. As a consequence, channel codes are usually difficult to adapt to time-varying channel conditions. Moreover, source codes are suboptimal due to complexity constraints.

A. Diallo is with the L2S – CNRS - SUPELEC - Univ Paris-Sud, France. C. Weidmann is with INTHFT, Vienna University of Technology, Vienna, Austria. M. Kieffer is with Telecom ParisTech, Paris, France, on leave from L2S. This work was supported by the European Commission in the framework of the FP7 Network of Excellence in Wireless COMmunications NEWCOM++ (contract n. 216715). Parts of this paper have been presented at EUSIPCO 2009.

These issues have prompted the development of joint source-channel (JSC) coding techniques, which aim at designing low-complexity codes simultaneously providing data compression and error correction capabilities. The hope is to get joint codes outperforming separate codes when the length of the codes is constrained, see [2]. Compression efficiency is measured by the ratio of the average code length to the source entropy [3], while the error-correction performance may be predicted with an union bound using the *distance properties* of the code, *i.e.*, its *free distance* and *distance spectrum*, see [4]. JSC coding using error-correcting variable-length codes (JSC-VLC) was introduced in [5], while design techniques aiming at optimizing distance properties of such codes were reported in [6] and later for a small subclass of JSC integer arithmetic codes (JSC-IAC) in [7].

This paper focuses on the optimization of codes generated by *finite-state encoders* (FSEs), which can be used to describe many JSC codes, including JSC-VLC and JSC-IAC. More precisely, our aim is to efficiently explore the (already very large) subclass of *finite-state codes* (FSCs) corresponding to JSC-IAC. One prerequisite for such optimization is the availability of efficient tools to evaluate distance properties of FSCs, which constitutes the first contribution of this paper, see Section III.

The first tools for evaluating distance properties considered linear FSCs, such as convolutional codes (CCs) [8]. In [9], Viterbi computed transfer functions on the state diagram of CCs to obtain their distance spectra and deduced their free distance. In [10], a variant of Dijkstra's shortest path algorithm is applied on the CC state diagram to compute the free distance without generating the spectrum. Later, [11] proposed a fast tree search algorithm for computing the CC distance spectrum. All these techniques have a complexity that is linear in the number of encoder states, due to the linearity of CCs.

For nonlinear FSCs, all pairs of codewords have to be compared to compute the free distance. For Euclidean-distance codes generated by trellis-coded modulation (TCM) [12], [13] used the product graph derived from the graph associated to the FSE. This allows to compute the distance spectrum of TCM in the code (signal) domain and to infer the free distance. For a FSC with 2^ν states, a product trellis with $(2^\nu)^2$ states is required for these evaluations [13]. For the class of *geometrically uniform* FSCs [14], which includes certain TCM codes, a modified generating function on a state diagram with only 2^ν states is sufficient to compute the spectrum [15].

All the above-described techniques are for fixed-rate codes, more precisely for FSEs defined by graphs where all transitions have input labels of the same length k, as well as output labels of the same length n, as is the case, *e.g.*, for rate k/n CCs. Distance properties for JSC-VLCs were first evaluated in [6], [16], where a lower bound for their free distance and exhaustive (exponential complexity) algorithms for their distance spectrum were proposed. Graphs that are similar to those used for distance properties of fixed-rate trellis codes have also been used in the context of JSC-VLCs, but to evaluate other figures of merit. For example, [17] defined a *testing graph*, consisting of the product graph that represents only pairs of paths at null Hamming distance to define a test for synchronizability of VLCs. The *error-state diagram* introduced by [18] for VLCs is the product graph that represents the pairs of paths at Hamming distance one to study the resynchronization properties of the decoder after a single bit error in the encoded sequence. In [19], these results were extended to JSC-IACs.

Evaluating the distance properties of nonlinear FSCs corresponding to JSC-IACs is slightly more complex than for JSC-VLCs. This is due to the fact that the FSE of a JSC-VLC has only one state in which paths can diverge and converge, while there may be many such states for a JSC-IAC. First analytical tools for JSC-IACs were proposed in [7], where the free distance is evaluated with polynomial complexity, whereas approximated distance spectra are obtained with exponential complexity as in [16]. More recently, [20] explicitly defined variable-length finite-state codes (VL-FSCs) generated by variable-length finite state encoders (VL-FSEs) and proposed a matrix method with polynomial complexity to compute the exact distance spectrum in the code domain or some upper bound on it. The definitions of VL-FSEs and VL-FSCs will be recalled in Section II.

In Section III, we first generalize the methods proposed in [13] and [10] to all FSCs, in order to be able to evaluate distance properties. A product graph inspired by that in [13] is proposed for general FSEs and is simplified to get two graphs: the *modified product graph* (MPG), which allows to compute the code domain distance spectrum using a transfer function approach, and the *pairwise distance graph* (PDG). The PDG allows to compute the free distance of the FSC by applying Dijkstra's algorithm as in [10], without computing the entire distance spectrum. This approach is much less complex than the technique for computing the free distance of a JSC-IAC proposed in [7].

Our aim is then to apply these free distance evaluation tools to the efficient optimization of JSC-IACs.

Arithmetic coding (AC) [21] is an efficient source coding method whose variants have been used in recent still image and video coders [22], [23]. Nevertheless, AC is particularly vulnerable to transmission errors. To overcome this, the most common form of JSC-AC introduces 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 *code interval* [24]. The larger the FS probability, the higher the redundancy and the robustness against errors. This idea is extended in [25], which proposes the introduction of multiple FSs (MFS). All these FS techniques can be applied to IAC [26] leading to JSC-IAC. Optimization of JSC-IAC has been considered in [7], assuming that the total probability allotted to the (M)FS and the probability of each individual FS are independent of the states of the FSE representing the JSC-IAC. However, the class of JSC-IAC with state-independent probability allocation for the FSs is significantly smaller than that with state-dependent allocation. For a fixed amount of redundancy, more robust JSC-IACs than those obtained by [7] may be obtained.

The second contribution of this paper, described in Section IV, consists in presenting some algorithms to globally optimize the free distance of some FSE by adjusting the introduction of MFS with in binary input JSC-IAC.

A JSC-IAC (and its corresponding FSE) may be described by its *characteristic parameters* (source probabilities, arithmetic precision, design rate) and by the way MFS are introduced. The set of all JSC-IACs for fixed parameter values and with state-dependent MFS is generally huge, but Section IV-B shows that it may be structured with a tree, where all JSC-IAC codes correspond to leaves of the tree. An efficient branch-and-prune algorithm is then used to explore this tree and discard large parts of it as soon as it can be shown that all JSC-IACs stemming from a given node of the tree cannot have good performances in terms of free distance. An extension to non-binary input JSC-IACs is then presented in Section IV-C. Experimental results for both binary input and non-binary input JSC-IACs are then provided in Section V.

II. Joint source-channel finite-state encoders

We briefly recall and extend some definitions from [20]. A binary-output FSE may be represented with a directed graph $\Gamma(\mathcal{S},\mathcal{T})$, where $\mathcal{S}$ is the set of states (vertices) and $\mathcal{T}$ is the set of transitions (directed edges). Each transition is labeled with a sequence of input symbols and a sequence of output bits. Let $\sigma(t)$ be the originating state of a transition $t \in \mathcal{T}$ and $\tau(t)$ its target state, while $I(t)$ denotes its input label and $O(t)$ its output label. Let $P(t)$, $t \in \mathcal{T}$, be the probability that $I(t)$ is emitted by the source, which for simplicity we assume to be memoryless. A path $\boldsymbol{t} = (t_1 \circ t_2 \circ \cdots \circ t_k) \in \mathcal{T}^k$ on the graph 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). By extension, we define $\sigma(\boldsymbol{t}) = \sigma(t_1)$ and $\tau(\boldsymbol{t}) = \tau(t_k)$, as well as $I(\boldsymbol{t})$ and $O(\boldsymbol{t})$, which are the concatenations of the input, respectively output, labels of $\boldsymbol{t}$. The probability of a path is $P(\boldsymbol{t}) = \prod_{i=1}^{k} P(t_i)$. Finally, $\ell(\boldsymbol{x})$ is the length (in symbols or bits) of the sequence $\boldsymbol{x}$.

We assume that the FSE graph is irreducible, *i.e.*, that any state can be reached from any other in a finite number of transitions, and that it is aperiodic, *i.e.*, that the state recurrence times, measured in output bits, are not multiples of an integer period $m > 1$. These assumptions imply that the FSE together with the source being encoded forms an ergodic Markov chain, which has a unique stationary state distribution. Let $P^*(s)$, $s \in \mathcal{S}$, be the stationary probability of the state s, which is computed taking into account the output label lengths as outlined in [20].

For a FSE to be a proper source encoder, for every state, the input labels of the outgoing transitions have to form a complete prefix set, which implies that their transition probabilities sum to one [20].

Given an initial state s_0, the succession of states of the FSE for all possible (semi-)infinite input sequences can be displayed with a trellis, which can be viewed as a description of the temporal evolution of the FSE. The output labels of all paths through the trellis form a FSC, whose performance is determined by its *coding rate* and its *error correcting capability*. The error correcting capability is primarily characterized by the *free distance* d_{free} (a finer characterization is possible through the *distance spectrum*). Under the assumption that the FSE is an irreducible graph, the free distance will be the same for every possible initial states.

Definition 1: The coding rate R_{c}, in bits per symbol, is the ratio between the average length of the output labels and the average length of the input labels of the transitions in $\mathcal{T}$,

$$R_{\text{c}} = \frac{\sum_{t \in \mathcal{T}} P^*(\sigma(t)) P(t) \ell(O(t))}{\sum_{t \in \mathcal{T}} P^*(\sigma(t)) P(t) \ell(I(t))}. \tag{1}$$

Definition 2: Let $\mathcal{P}_{s_0}^{k}$ be the set of all paths with k transitions starting in s_0. The FSC $\mathcal{C}\left(\Gamma, s_0\right)$ is the set of all infinite-length output sequences generated by the FSE from s_0, $\mathcal{C}\left(\Gamma, s_0\right) = \{O(\boldsymbol{t}) : \boldsymbol{t} \in \mathcal{P}_{s_0}^{\infty}\}$.

The Hamming distance d_{H} between two equal-length sequences $\boldsymbol{x}, \boldsymbol{y}$ is equal to the Hamming weight w_{H}, *i.e.*, the number of non-zero entries of their elementwise difference, $d_{\mathrm{H}}(\boldsymbol{x},\boldsymbol{y}) = w_{\mathrm{H}}(\boldsymbol{x}-\boldsymbol{y})$. If two paths $(\boldsymbol{t}_1,\boldsymbol{t}_2) \in \mathcal{T}^{k_1} \times \mathcal{T}^{k_2}$ are such that $\ell(O(\boldsymbol{t}_1)) = \ell(O(\boldsymbol{t}_2))$, then we will write $d_{\mathrm{H}}(\boldsymbol{t}_1,\boldsymbol{t}_2) = d_{\mathrm{H}}(O(\boldsymbol{t}_1), O(\boldsymbol{t}_2))$.

Definition 3: The free distance, d_{free}, of the FSC $\mathcal{C}\left(\Gamma, s_0\right)$ is the minimum Hamming distance between any pair of distinct code sequences. Let $\mathcal{P}$ be the set of all pairs of paths in $\left(\mathcal{T}^{k_1} \times \mathcal{T}^{k_2}\right)_{1\leq k_1,k_2<\infty}$ diverging in some state and converging for the first time in the same or another state and with the same length of output labels. Consequently, d_{free} is also the minimum Hamming distance in $\mathcal{P}$

$$d_{\mathrm{free}} = \min_{(\boldsymbol{t}_1,\boldsymbol{t}_2)\in\mathcal{P}} d_{\mathrm{H}}\left(\boldsymbol{t}_1,\boldsymbol{t}_2\right). \tag{2}$$

Definition 4: The distance spectrum [9] in the code domain can be represented with a generating function

$$G\left(D\right) = \sum_{d=d_{\mathrm{free}}}^{\infty} A_d D^d, \tag{3}$$

where A_d is the average number of paths at Hamming distance d from a given path. In the most general case, A_d can be defined as [7], [20],

$$A_d = \sum_{\substack{(\boldsymbol{t}_1,\boldsymbol{t}_2)\in\mathcal{P} \\ d_{\mathrm{H}}(\boldsymbol{t}_1,\boldsymbol{t}_2)=d}} P^*(\sigma(\boldsymbol{t}_1))P(\boldsymbol{t}_1) \tag{4}$$

By the above definitions we see that the code $\mathcal{C}\left(\Gamma, s_0\right)$ and its free distance are independent of the source (provided all source letters have nonzero probability), while the joint source-channel nature shows that the rate and the spectrum coefficients A_d depend on the source statistics.

Finally, we introduce two notions which may help searching codes with a computer. A *complete automaton* (CA) is an FSE that can encode any source sequence. An *incomplete automaton* (IA) is an FSE having one or more terminal states without outgoing transitions, in which encoding stops, see Figure 1. Such terminal states are called *stopping states* or *unexplored states*. An IA Γ_0 generates finite-length prefixes of code sequences and possibly some infinite-length code sequences. Let the *incomplete code* $\mathcal{C}_0 = \mathcal{C}(\Gamma_0, s_0)$ contains these (finite and infinite) sequences.

Definition 5: The free distance associated to an incomplete FSE is the minimum Hamming distance between any pair of 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 (finite-length) paths ending in the same state or (infinite-length) path converging at some time instant in the same state, the free distance is infinite.

An IA or CA Γ_1 is *derived* from an IA Γ_0 if it has been obtained by exploring (adding the successor states of) one or more terminal states of Γ_0 (hence the graph Γ_0 is a subgraph of Γ_1) [27]. Let $\mathcal{C}_0$ and $\mathcal{C}_1$ be two codes generated by Γ_0 and Γ_1, respectively, with free distance $d_{\text{free}}^{(0)}$ and $d_{\text{free}}^{(1)}$, respectively. We define the relation $\mathcal{C}_0 \preceq \mathcal{C}_1$, meaning that all elements in $\mathcal{C}_0$ are prefixes of elements in $\mathcal{C}_1$. If $\mathcal{C}_0 \preceq \mathcal{C}_1$, an important property following directly from Definitions 3 and 5 is that $d_{\text{free}}^{(0)}$ is an upper bound on $d_{\text{free}}^{(1)}$.

Lemma 1: Let $\mathcal{C}_0$ and $\mathcal{C}_1$ be two (incomplete) codes such that $\mathcal{C}_0 \preceq \mathcal{C}_1$. Then $d_{\text{free}}(\mathcal{C}_0) \geqslant d_{\text{free}}(\mathcal{C}_1)$.

In [7], three types of FSE describing the coding operations 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 a bit-clock FSE (B-FSE) suited for the evaluation of distance spectra, where each transition is labeled with exactly one output bit, see Figure 1. Details on how these FSEs are obtained can be found in [7]. In the sequel, $\mathcal{S}_r$ and $\mathcal{T}_r$ are the set of states and transitions in R-FSE and $\mathcal{S}_b$ and $\mathcal{T}_b$ the set of states and transitions in B-FSE. M_r and M_b are the number of states in a R-FSE and a B-FSE, respectively.

III. Characterization of the error correcting performance

From here on, we consider B-FSE. To evaluate the free distance and the distance spectrum of some JSC-IAC described by a B-FSE $\Gamma_b(\mathcal{S}_b, \mathcal{T}_b)$, techniques inspired from [13] are applied to track the distances between pairs of paths in the B-FSE. The product graph $\Gamma_b^2 = \Gamma_b \times \Gamma_b$ labeled with Hamming distances (which would yield the product trellis considered in [13]) is considered for that purpose. The main difference with [13] is that for VL-FSE, states may have a different number of outgoing transitions. Then we show how this product graph may be simplified.

A. Efficient free distance computation

This section describes the generation of the product graph associated to a B-FSE and its simplification to efficiently compute the free distance of the FSC generated by the B-FSE.

Consider the set of states $\mathcal{S}_b = \{s_i \ : \ 0 \leqslant i < M_b\}$ and the set of transitions $\mathcal{T}_b$ of the directed graph $\Gamma_b(\mathcal{S}_b, \mathcal{T}_b)$ representing some B-FSE. The product graph associated to $\Gamma_b(\mathcal{S}_b, \mathcal{T}_b)$ is the directed graph $\Gamma_b^2(\mathcal{S}_b \times \mathcal{S}_b, \mathcal{T}_b \times \mathcal{T}_b)$ with M_b^2 states $s_{i,j}$ defined as $s_{i,j} = (s_i, s_j), \ 0 \leqslant i, j < M_b$.

For any pair of transitions (u, v) in the original graph, Γ_b^2 contains a directed edge e with $e = (u, v)$, $\sigma(e) = s_{\sigma(u),\sigma(v)}$, and $\tau(e) = s_{\tau(u),\tau(v)}$. The weight of the edge e, $w_{\mathrm{H}}(e)$ is defined as the Hamming distance between the output of the two transitions, u and v, *i.e.*, $w_{\mathrm{H}}(e) = d_{\mathrm{H}}(u, v)$.

A directed path $\boldsymbol{e}$ in Γ_b^2 from the state $\mathrm{s}_{i,j}$ to the state $\mathrm{s}_{m,n}$, is a sequence of edges $\boldsymbol{e} = (e_1 \circ e_2 \circ \cdots \circ e_N)$ such that $\sigma(e_{\mu+1}) = \tau(e_\mu)$ for $1 \leqslant \mu < N$. The weight of this directed path, $w_{\mathrm{H}}(\boldsymbol{e})$ is defined as

$$w_{\mathrm{H}}(\boldsymbol{e}) = \sum_{\mu=1}^{N} w_{\mathrm{H}}(e_\mu). \tag{5}$$

Since Γ_b is output bit clock, one has $w_{\mathrm{H}}(\boldsymbol{e}) \leqslant N$.

Thus, the weight of a directed path in Γ_b^2 from a state $\mathrm{s}_{i,j}$ to a state $\mathrm{s}_{m,n}$, is the Hamming distance between the output bits of two paths $(\boldsymbol{t}_1, \boldsymbol{t}_2) \in \mathcal{T}_b^k \times \mathcal{T}_b^k$. Hence, when we explore Γ_b^2 from the initial state $s_{0,0}$, the weights of the obtained directed paths represent all the Hamming distances of all possible pairs of codewords in $\mathcal{C}(\Gamma_b, s_0) \times \mathcal{C}(\Gamma_b, s_0)$, including d_{free} according to Definition 3. Figure 1 (b) gives an example of a graph for a B-FSE (where s_0 and s_2 are the states where paths diverge). Figure 2 shows the product graph derived from the B-FSE in Figure 1 (b). The edges in Figure 2 are labeled with their weight.

Since, for the evaluation of d_{free} one is only concerned with paths in Γ_b^2 belonging to $\mathcal{P}$, we can derive from Γ_b^2 a modified product graph (MPG) that represents only these paths. Consider the two sets of states $S_{\mathrm{div}} \subset \mathcal{S}_b^2$ and $S_{\mathrm{conv}} \subset \mathcal{S}_b^2$ such that

$$\mathcal{S}_{\mathrm{div}} = \left\{ s_{i,i} \in \mathcal{S}_b^2 \ : \ \exists\, (u, v) \in \mathcal{T}_b^2, u \neq v, \text{such that } \sigma(u) = \sigma(v) = s_i \right\}, \tag{6}$$

$$\mathcal{S}_{\mathrm{conv}} = \left\{ s_{i,i} \in \mathcal{S}_b^2 \ : \ \exists\, (u, v) \in \mathcal{T}_b^2, u \neq v, \text{such that } \tau(u) = \tau(v) = s_i \right\}. \tag{7}$$

$\mathcal{S}_{\mathrm{div}}$ is the set of states of Γ_b^2 in which the outgoing edges consist of pairs of diverging transitions in $\mathcal{T}_b^2$ having the same originating state in $\mathcal{S}_b$. We merge the states in $\mathcal{S}_{\mathrm{div}}$ into a single state s_{in} with only outgoing edges. $\mathcal{S}_{\mathrm{conv}}$ is the set of states of Γ_b^2 in which the incoming edges consist of pairs of distinct transitions in $\mathcal{T}_b^2$ converging in the same target state in $\mathcal{S}_b$. We merge the states in $\mathcal{S}_{\mathrm{conv}}$ into a single state s_{out} with only incoming edges. In Figure 2, $\mathcal{S}_{\mathrm{div}} = \{s_{0,0}, s_{2,2}\}$ and $\mathcal{S}_{\mathrm{conv}} = \{s_{0,0}, s_{2,2}\}$. In Γ_b^2, the set of edges $\{e = (u, u) \ : \ u \in \mathcal{T}_b\}$ corresponds to pairs of paths which have not diverged, therefore, according to Definition 3, this set will be not useful to find d_{free}. If we replace in Γ_b^2 the sets $\mathcal{S}_{\mathrm{div}}$ and $\mathcal{S}_{\mathrm{conv}}$ by respectively s_{in} and s_{out}, and we remove the set $\{e = (u, u) \ : \ u \in \mathcal{T}_b\}$, we obtain a modified product graph (MPG), in which s_{in} is the initial state and s_{out} is the final state, and which still contains all paths needed for the evaluation of d_{free}. The MPG derived from the product graph in Figure 2 is represented in Figure 3. When we explore the MPG from the initial state to the final state the weights of the paths give the Hamming distances in $\mathcal{P}$. Thus finding d_{free} amounts to determining the directed path(s) from s_{in} to s_{out} with smallest weight.

It can be seen that in the MPG, if $\boldsymbol{e}_1$ is a directed path from s_{in} to $s_{i,j}$, $i \neq j$, then, there is also a directed path $\boldsymbol{e}_2$ from s_{in} to $s_{j,i}$ such that

$$\ell(\boldsymbol{e}_1) = \ell(\boldsymbol{e}_2) \;\text{ and }\; w_{\mathrm{H}}(\boldsymbol{e}_1) = w_{\mathrm{H}}(\boldsymbol{e}_2). \tag{8}$$

This is so since d_{H} is symmetric in its arguments. Here, we say that the paths $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ in the MPG are equivalent. Thus the complexity can be further reduced by defining a pairwise distance graph (PDG) that contains a single path for each pair of equivalent paths in the MPG. For this end, the two states $s_{i,j}$ and $s_{j,i}$ in the MPG are merged and replaced with a single state $s_{\nu,\gamma}$ ($\nu = \min(i,j), \gamma = \max(i,j)$) in the PDG, and the two directed edges (u,v) and (v,u) (with $u \in \mathcal{T}_b$ and $v \in \mathcal{T}_b$) in MPG are replaced by a single edge in the PDG. The PDG derived from the MPG in Figure 3 is represented in Figure 4.

Finding d_{free} with this PDG is again the same as finding the directed path(s) from s_{in} to s_{out} with the smallest weight. This is known as the shortest weighted path problem in graph theory and can be solved efficiently using Dijkstra's algorithm [27], since all weights are non-negative. The number of states in PDG, M_{PDG}, is

$$M_{\mathrm{PDG}} = \frac{M_b \times (M_b - 1)}{2} + 2. \tag{9}$$

M_{PDG} is the maximum number of states explored when Dijkstra's algorithm is applied on PDG to compute d_{free}.

An alternative method to compute the free distance of a JSC-IAC was presented in [7]. It uses a three-dimensional array defined as $\triangle_n = (a_{k,i,j})_{0 \le k,i,j < M_b}$, where $a_{k,i,j}$ is the minimum Hamming distance between all pairs of paths of length n (in code domain) starting from the state s_k and ending respectively in the states s_i and s_j and not having converged. This method is an iterative method over the path length n. The algorithm needs to compare the paths up to n_{free}, which is the path length for which no unmerged pairs of paths with distance less than d_{free} exist. In practical implementations, some upper bound $n_{\max}$ for n has to be given to perform the algorithm in [7] in finite time. If $n_{\max}$ is too small, no guarantee may be provided that the actual value of d_{free} has been found. This may occur for some codes like catastrophic codes, since such codes contain pairs of codewords with a finite Hamming distance that correspond to never-converging paths. The computational complexity of this method is $O\left(n_{max} \times |\mathcal{T}_b|^2 \times M_r\right)$, while the worse complexity of applying Dijkstra's algorithm on a PDG is $O(M_{\text{PDG}}^2)$. With a better implementation of Dijkstra's algorithm using Fibonacci heaps, the complexity may be reduced to $O(|\mathcal{T}_{\text{PDG}}| + M_{\text{PDG}} \times \log(M_{\text{PDG}}))$, where $\mathcal{T}_{\text{PDG}}$ is the set of edges in PDG. The existence of catastrophic pairs of paths implies a zero-weight (directed) loop in the PDG and vice-versa. For instance in Figure 5 (a) the codewords obtained by $(s_0, s_1, s_3, \cdots, s_3)$ and $(s_0, s_2, s_4, \cdots, s_4)$ have Hamming distance one. Dijkstras algorithm has no problems with such codes, since during the process, each edge of the PDG is explored at most once.

B. Distance spectra in the code domain

This section briefly describes the way an MPG could be labeled such that the distance spectrum in code domain is obtained from a generating function, which may be computed by evaluating a symbolic transfer function, *e.g.*, using Mason's gain formula [28]. For this end, we assign to every edge $e = (u, v)$ a gain $g(e)$ defined as

$$g(e) = \begin{cases} P^*(\sigma(u))P(u)D^{w_{\text{H}}(e)} & : \ \sigma(u) = \sigma(v), \\ P(u)D^{w_{\text{H}}(e)} & : \ \sigma(u) \neq \sigma(v), \end{cases} \tag{10}$$

where D is a symbolic variable used to track the path weight. Then, applying Mason's gain formula between s_{in} and s_{out} in the MPG allows to obtain a transfer function $G(D)$, which represents the distance spectrum according to (4). In [20] a method to compute the distance spectrum (in code domain) directly using matrices instead of generating functions is proposed. In this method, the coefficients A_d (for $0 \leq d < \infty$) of $G(D)$ are computed one by one. The index of the first non null coefficient gives the value of the free distance. The method in [20] uses a matrix inversion with matrices of size $M_b^2 \times M_b^2$ to compute the coefficients. This is more complex than using Dijkstra's algorithm.

The method described above cannot be applied in the information (source) domain for general variable-length FSCs, since the distances between information sequences need to be expressed using the Levenshtein distance [29], which is not additive.

IV. Optimizing finite-state JSC codes based on arithmetic coding

This section describes a method to search for error-correcting codes with large free distance. Our approach is to explore a subset of the set $\mathcal{F}$ of all FSEs. $\mathcal{F}$ contains the set $\mathcal{F}_{\text{u}}$ of FSEs which generates uniquely decodable FSCs, which in turn contains the set $\mathcal{F}_{\text{JSC-IAC}}$ of all FSEs corresponding to JSC-IAC. The latter contains the set $\mathcal{F}_{\text{JSC-IAC}}^{\text{T}}$ of all FSEs representing an IAC followed by an error-correcting code (*i.e.* classical separate *tandem* encoding). It also contains the set $\mathcal{F}_{\text{JSC-IAC}}^{\text{M}}$ of all *memoryless* JSC-IAC, *i.e.*, JSC-IAC whose encoding behavior depends only on the current arithmetic encoder state. For such codes, the encoder behavior can depend on the previously encoded symbols or on the previously generated output bits only indirectly through the state. Some IAC followed by block codes belong to the set $\mathcal{F}_{\text{JSC-IAC}}^{\text{M}}$, but more general tandem schemes consisting of an IAC followed by a CC do not belong to $\mathcal{F}_{\text{JSC-IAC}}^{\text{M}}$. For the sake of simplicity, we restrict our exploration to the set $\mathcal{F}_{\text{JSC-IAC}}^{\text{M}}$.

As the set $\mathcal{F}_{\text{JSC-IAC}}^{\text{M}}$ still remains large, it will be interesting to structure it in a way that allows to promptly explore it for the largest d_{free}. This may be done with a tree in which the leaves correspond to complete JSC-IACs (CAs) and internal nodes correspond to incomplete JSC-IACs (IAs). From the root which is the initial IA determined by the values of the characteristic parameters, the tree is generated by successively extending all intermediate IAs as described in Sections IV-B and IV-C. Using Lemma 1 in a branch-and-prune algorithm substantially reduces the time needed to find the best JSC-IAC. The idea of the branch-and-prune algorithm is to successively eliminate large parts of the tree which cannot lead to the optimum d_{free}. This is done by iteratively updating a lower bound $\underline{d}_{\text{free}}$, of the largest free distance which may be obtained for given values of the characteristic parameters. When exploring the tree, if an IA is reached, its d_{free} is compared to $\underline{d}_{\text{free}}$. If it is larger, the IA is extended, in the other case, the IA is no more explored, since according to Lemma 1 all IAs and CAs derived from it will have a d_{free} smaller or equal to $\underline{d}_{\text{free}}$. If a CA is reached and its d_{free} is larger than $\underline{d}_{\text{free}}$, then $\underline{d}_{\text{free}}$ is updated. Three ways for exploring the tree are considered: *depth first exploration*, *breadth first exploration*, and a *sort method* which extends the IA with the largest d_{free}. Their respective efficiency is compared in Section V. Section IV-A recalls some bases of AC which may be helpful for understanding the tree construction methods. Sections IV-B and IV-C describe the tree construction method for binary input and non-binary input JSC-IAC respectively.

A. Finite-state integer arithmetic coding

The basic idea of binary AC is to assign to every sequence of source symbols a unique subinterval of the unit interval $[0, 1)$. 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 iteratively partitioning the code interval $[0, 1)$ according to the probabilities of the source symbols. Let K be the size of the source alphabet $\{a_1, a_2, \ldots, a_K\}$. At the end of some iteration, assume that the code interval is $[l, h)$. At the next iteration, $[l, h)$ is partitioned into K non-overlapping subintervals $\{I_1, I_2, \ldots, I_K\}$, the width of I_i being proportional to the probability of the symbol a_i. The subinterval corresponding to the symbol to encode is then selected as the new code interval. Partitions and selections continue until the last symbol has been processed. The encoder chooses a value in the current code 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 IAC.

Binary IAC, also called quasi-arithmetic coding (QAC) [30], [26] works as 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 interval $[l, h)$ are renormalized as follows

- If $h \leqslant T/2$, l and h are doubled.
- If $T/2 \leqslant l$, l and h are doubled after subtracting $T/2$.
- If $T/4 \leqslant l$ and $h \leqslant 3T/4$, 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 f (for *follow*). If the current interval before renormalization lies 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* [21].

Since the IAC encoding process can be characterized by $[l, h)$ and f (and the source probabilities), the state of the automaton representing the encoder 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 FSE. In general, f may grow without bounds, but it can be easily limited to $f \leqslant f_{\max}$, as in [31]. The present work takes the approach of [7]: whenever $f = f_{\max}$ and the current source interval is such that f could be further incremented, the symbol probabilities are temporarily modified to force a follow-on after encoding the current symbol.

B. A tree of binary input JSC-IAC automata

Consider a source X with alphabet $\mathcal{A} = \{a_0, a_1\}$ and pr$(X = a_0) = p_0$ and pr$(X = a_1) = p_1$. Let (l, h, f) be the current state of the encoder and $w = h - l$ be the width of the current code interval. During encoding, $[l_i, h_i)$ is the subinterval of width $w_i = h_i - l_i$ assigned to $a_i, i = 0, 1$. As mentioned in Section I, a JSC-IAC may be derived from an IAC by considering FSs. In the case of a single FS, let p_ε be the "probability" of the FS and w_ε be the width of the subinterval assigned to it. Given p_ε and p_0, the widths of the subintervals of the code interval $[l, h)$ are computed as follows

$$w_\varepsilon = \langle p_\varepsilon \times w \rangle \,, \tag{11}$$

$$w_0 = \langle p_0 \times (h - l - w_\varepsilon) \rangle \,, \tag{12}$$

$$w_1 = h - l - w_0 - w_\varepsilon, \tag{13}$$

where $\langle \cdot \rangle$ means rounding towards the nearest integer. In a more general case, the probability of the FS may be split among up to three FSs $\{\varepsilon_0, \varepsilon_1, \varepsilon_2\}$, with corresponding probabilities $\{p_{\varepsilon_0}, p_{\varepsilon_1}, p_{\varepsilon_2}\}$ such that $p_{\varepsilon_0} + p_{\varepsilon_1} + p_{\varepsilon_2} = p_\varepsilon$. Figure 6 shows how the code interval may be partitioned during the coding process in the case of a JSC-IAC with 3 FSs. The way p_ε is distributed among $\{\varepsilon_0, \varepsilon_1, \varepsilon_2\}$ may be *state-independent, i.e.*, independent of the values of l, h, and f, or it may be *state-dependent*, in which case, the order of the subintervals assigned to source symbols may also change. Considering a state-dependent FS probability assignment provides a large design freedom allowing to build automata potentially leading to better codes than those obtained with a state-independent design, already considered in [7]. However, to the best of our knowledge, no analytical method is known to find the optimal state-dependent probability assignment.

The set of all encoders can be obtained by iteratively exploring the successors of all states, starting from the IA with state $(l = 0, h = T, f = 0)$, for every admissible configuration of the subintervals $[l_0, h_0)$ (associated to a_0) and $[l_1, h_1)$ (associated to a_1) of $[l, h)$. This may be done by letting both l_0 and l_1 vary from l to $h-1$ in steps of one. Then one may check whether one of the following conditions is satisfied.

$$l \leqslant l_0 < (h_0 = l_0 + w_0) \leqslant l_1 < (h_1 = l_1 + w_1) \leqslant h, \tag{14}$$

$$l \leqslant l_1 < (h_1 = l_1 + w_1) \leqslant l_0 < (h_0 = l_0 + w_0) \leqslant h. \tag{15}$$

Each time (14) or (15) are satisfied, a new IA or CA is derived from the previous IA by supplementing it with two states (obtained from $[l_0, h_0)$ and $[l_1, h_1)$ after appropriate renormalizations) if they do not already exist. The resulting IAs are then explored in turn. From the initial IA with state $(l = 0, h = T, f = 0)$, the expansion of IAs gives a tree of automata in which all internal nodes corresponds to IAs and leaves correspond to CAs and each edge corresponds to the exploration of an unexplored state.

Figure 7 shows how all possible automata for a given value of the characteristic parameters and with a state-dependent FS probability assignment may be described by a tree. The initial IA consists of the unexplored state $(0, T, 0)$ shared by all automata. It forms the root of the tree. Each node in the first layer of internal nodes represents an IA for which a given configuration of the subintervals of the initial code interval have been considered.

Figure 8 shows an example of a tree of automata for the characteristic parameters $T = 8$, $f_{\max} = 1$, $p_0 = \frac{1}{4}$, and $p_\varepsilon = \frac{1}{2}$. The circles labeled $s_0, s_1, \dots$ represent the states of the IA or CA. They are shaded for unexplored states. The initial incomplete automaton IA_0 consists of the initial state $s_0 = (0, 8, 0)$. One possible manner to extend the initial state is to assign to a_0 the interval $[0, 1)$ and to a_1 the interval $[1, 4)$ leading to the second incomplete automaton IA_1. The last possible extension of the initial state assigns to a_0 the interval $[7, 8)$ and to a_1 the interval $[4, 7)$ yielding the incomplete automaton IA_N. Iterating this approach, one may get CAs such as the one shown shaded on the bottom left side of the tree.

C. Extension to non-binary input JSC-IAC

For sources with $K > 2$ symbols, extending the interval subdivision presented in Section IV-B would require to consider at most $K + 1$ FSs. Allowing state-dependent assignment of the probabilities of the FSs may lead to a very high number of automata for given values of the characteristic parameters. Nevertheless, in practice most (if not all) source coding standards involving arithmetic codes introduce a binarization step of the non-binary symbols to encode before binary input arithmetic coding [32]. This allows the use IAC with reduced precision, compared to non-binary input ACs. Instead of a single source probability model, several (adaptive) probability models are considered and chosen depending on some *context*, corresponding here to the index of the bit to encode in a binarized symbol. The methods described in Section IV-B are thus extended for binarized sources with $K > 2$ symbols, accounting for some simple context, with non-adaptive probability model.

Let X be a memoryless source with alphabet $\mathcal{A}_K = \{a_1, \dots, a_K\}$ and corresponding set of symbol probabilities $\mathcal{P}_a = \{p_{a_1}, \dots, p_{a_K}\}$. Without loss of generality, assume that $p_{a_i} \geq p_{a_{i+1}}$, $i = 1, \dots, K$.

A binarization of $\mathcal{A}_K$ may be done as follows. Consider first some $x \in \mathbb{N}$ and $L \in \mathbb{N}^*$ such that $L \geq \lceil \log_2(x) \rceil$. Let $\mathrm{B}_L(x)$ be the binary representation of the integer x on L digits. For instance $\mathrm{B}_3(1) = 001$. Now consider $L_K \in \mathbb{N}^*$ such that $L_K \geq \lceil \log_2(K) \rceil$. For each symbol $a_i \in \mathcal{A}_K$, $\mathrm{B}_{L_K}(i-1)$ is a possible binary representation.

Example 1: Consider the 26 letters of the English alphabet $\mathcal{A}_{26}$. Then $K = 26$ and $L_K = 5$ bits. Table I shows the probability of occurrence [16] and the bit assignment for each symbol in $\mathcal{A}_{26}$. The entropy of a memoryless source X generating symbols according to the probabilities given in Table I is $H(X) = 4.175$ bits/symbol.

Now denote a_i^j, $i = 1, \dots, K$ and $j = 1, \dots, L_K$ be the j-th bit in the binary representation of a_i. To avoid confusion with the output bits of the AC, a_i^j is called a symbol. Let p_b^j for $b \in \{0, 1\}$ and $j = 1, \dots, L_K$ be the probability that $a_i^j = b$. One has $p_b^j = \sum_{i=1}^K p_{a_i} \delta(a_i^j - b)$, where $\delta(x)$ is the indicator function ($\delta(x) = 1$ if $x = 0$ and $\delta(x) = 0$ else). One has of course $p_0^j + p_1^j = 1$. The entropy of a memoryless binary source X_2^j generating the j-th bit of a binarized symbol with probability model $\{p_0^j, p_1^j\}$ may easily be evaluated. The entropy of the source after binarization is then $\sum_1^{L(K)} H(X_2^j)$.

Example 2: For $\mathcal{A}_{26}$, $p_0^1 = 0.0963$; $p_0^2 = 0.269$; $p_0^3 = 0.379$; $p_0^4 = 0.432$; $p_0^5 = 0.455$ and $\sum_1^5 H(X_2^j) = 4.237$ bits/non-binarized symbol, which is greater than $H(X)$. Using an AC involving five independent probability models for memoryless binary sources leads thus to some loss in efficiency compared to an AC directly handling a K-valued source.

With a context corresponding to the index of the bit to encode in the binarized source symbol, the symbol a_i^j will be arithmetic coded with the probability model $\{p_0^j, p_1^j\}$. In the FSE, this requires keeping track of the context by supplementing the state $\{l, h, f\}$ with the value of the context to get the new state $\{l, h, f, j\}$, with $j = 1, \ldots, L_K$. As a consequence, the number of states of the FSE is multiplied by L_K when considering such context. When designing an JSC-IAC, one usually tries to reach some target source-channel coding rate R_c. For that purpose, a "probability"

$$p_\epsilon^j = 1 - 2^{-(R_\mathrm{c} - H(X_2^j))} \quad (16)$$

is assigned to each context j. The characteristic parameters of a JSC-IAC are then T, $f_{\max}$, $\mathcal{P}_a$, and R_c. As in Section IV-B, the set of all encoders which may be obtained by a state-dependent assignment of the FS probabilities may be obtained by iteratively exploring the successors of all states of IAs, starting with the IA having the single state $(l = 0, h = T, f = 0, j = 1)$, for every admissible configurations of the subintervals of a unexplored state.

D. Complexity issues

It would be useful for the branch-and-prune algorithm to find a relation between the characteristic parameters and the computational 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 required by the algorithm for computing the free distance of an automaton. Again, this is extremely difficult to estimate from the parameters without building the actual FSE.

V. Experimental results

Two sets of experiments are conducted. First, simple binary sources are considered, allowing an easy evaluation of the efficiency of the branch-and-prune algorithm compared to an exhaustive search for the best FSE. Dijkstra's algorithm-based free distance evaluation is compared to the method proposed in [7]. Various tree traversal algorithms are then compared. Second, a JSC-IAC for the binarized English alphabet is provided.

A. JSC-IAC for binary sources

A first binary-input JSC-IAC with characteristic parameters $T = 8$, $f_{\max} = 1$, $p_0 = 0.1$, and a design $R_\mathrm{c} = 0.62$ bits/symbol ($p_\varepsilon = 0.1$) is considered first. The time needed to generate all possible automata with a state-dependent assignment of the FS probabilities, to compute their free distance and select the best one is 6300 s. Using the branch-and-prune algorithm with breadth-first exploration, the time needed to find the largest d_{free} is only 25 s, resulting in a time saving of 99.6%. In both cases, free distances are evaluated with the technique of [7]. This first experiment has been done on an Intel Core 2 Duo at 2.66 Ghz with 1 Gb memory.

A second binary-input JSC-IAC with characteristic parameters $T = 16$, $f_{\max} = 1$, $p_0 = 0.1$, and a design $R_{\mathrm{c}} = 0.91$ bits/symbol ($p_\varepsilon = 0.27$) is now considered. An exhaustive exploration for such JSC-IAC would be unreasonably time-consuming. Table II shows the time (in seconds) needed with a depth-first exploration, a breadth-first exploration, and the sort method described in Section IV to find the best automaton. The free distance evaluation method of [7] is compared to that presented in Section III. $|\mathcal{S}_r|$ and $|\mathcal{T}_r|$ denote the number of states and the number of transitions of the corresponding R-FSE. These numbers depend on the exploration method, since several FSCs may have the same d_{free}, without necessarily having the same number of states or transitions. The coding rate is expressed in bits/symbol. The sort method is the best in terms of computing time. 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 $\underline{d}_{\mathrm{free}}$ may rapidly increase. Having a large value of $\underline{d}_{\mathrm{free}}$ at the beginning of the algorithm facilitates pruning large parts of the tree without exploring them. It can also be seen that using Dijkstra's algorithm to compute d_{free} is much more efficient than using the method in [7].

Compared to an equivalent tandem scheme (IAC followed by a Convolutional Code (CC)) with the same coding rate, the free distance of the obtained JSC-IAC remains suboptimal. For the example of Table II, consider a $R_{\mathrm{c}} = 91$ bits/symbol equivalent tandem scheme with an IAC 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 (respectively 3), the best d_{free} of a rate 1/2 CC is 4 (respectively 5) [33, Chapter 8]. The weakness of the JSC-IAC 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}. The minimum number of states for the best obtained FSE in the JSC-IAC case is 2 (Table II), while in the tandem scheme, the total number of states is the product of the number of states of the IAC and the number of states of the CC(at least 2). Hence, the joint scheme will be less complex than the tandem scheme. In Table II one notes a slight variation of the effective coding rate, which are due to rounding effects in finite-precision IAC. This second experiment and the followings have been done on an Intel Xeon E5420 at 2.50GHz with 64 Gb memory.

B. JSC-IAC for non-binary sources

Now, our aim is to optimize a JSC-IAC for the binarized English alphabet $\mathcal{A}_{26}$ given in Table I. To reduce the number of IAs and CAs to build in the tree of automata, only two values of the context are considered. The first corresponds to the first bit index and the second to the remaining bit indexes. Two probability models are then considered, namely $\{p_0^1, p_1^1\}$ and

$$p_0^{2:5} = \frac{\sum_{j=2}^{5} p_0^j}{4} \text{ and } p_1^{2:5} = 1 - p_0^{2:5}. \tag{17}$$

The probability $p_\varepsilon^{2:5}$ assigned to the context $2:5$ can be obtained with (16). To further simplify the search for a good code, the FS probability assignment is state-dependent, but is not allowed to vary for a given value of the context. This significantly reduces the amount of different automaton which may be built for a given value of the characteristic parameters taken now as $T = 32$, $f_{\max} = 1$, $\{p_0^1, p_1^1\}, \{p_0^{2:5}, p_1^{2:5}\}$, and a design $R_\text{c} = 14$ bits/symbols.

Dijkstra's algorithm is used to compute d_free and the *sort method* is used in the branch-and-prune algorithm. The best code has $d_\text{free} = 6$ and $R_\text{c} = 13.9$ bits/symbols. The time needed to find this code is 1425 s. JSC-IAC code design is thus possible even for large alphabet sizes, provided that a binarization process is considered. However, the reduced number of contexts and the constraints imposed on the FSs lead to a JSC-IAC which is less efficient than that proposed in [16], where a JSC-VLC for $\mathcal{A}_{26}$ with $d_\text{free} = 5$ and $R_\text{c} = 10.41$ bits/symbols is obtained.

Improvements may be obtained by considering more contexts and by allowing more variations of the state-dependent assignments of the FS probabilities. However, the price to be paid is a higher computational complexity. The considered branch-and-prune algorithm may be strongly parallelized, which may help addressing this issue.

VI. Conclusion

This paper has shown how established graph transfer function methods for fixed-rate channel codes can be generalized to compute the free distance and the distance spectrum of VL-FSC. The resulting method for computing the free distance is much more efficient than the method for JSC-IAC presented in [7] and does not have problems dealing with catastrophic codes.

It also shows that the proposed branch-and-prune algorithm (using the Sort Method) is a fast way to find the JSC-IAC with largest d_{free} for binary sources. Using an appropriate binarization process prior to AC and using several probability models, this method may be extended to the design of JSC-IAC for non-binary sources.

Nevertheless, at fixed code rate, the codes obtained for the time being remain less efficient than equivalent tandem schemes. Future work will consider the extension of JSC-IAC with an m-bit memory which may improve d_{free} by separating paths that would lead to small distances. The memory holds an integer $0 \leq \lambda \leq 2^m - 1$, so that the FSE state can be represented as (l, h, f, λ). The set of FSEs of JSC-IAC 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] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423 and 623–656, 1948.

[2] Y. Zhong, F. Alajaji, and L. L. Campbell, "On the joint source-channel coding error exponent for discrete memoryless systems," *IEEE Trans. Inform. Theory*, vol. 52, no. 4, pp. 1450–1468, 2006.

[3] T. M. Cover and J. M. Thomas, *Elements of Information Theory*. New-York: Wiley, 1991.

[4] A. J. Viterbi and J. Omura, *Principles of Digital Communication and Coding*. New-York: McGraw-Hill, 1979.

[5] M. Bernard and B. Sharma, "Some combinatorial results on variable-length error-correcting codes," *ARS Combinatoria*, vol. 25B, pp. 181–194, 1988.

[6] V. Buttigieg and P. Farrell, "On variable-length error-correcting codes," in *Proc. IEEE Int. Symposium on Inform. Theory.*, 1994, p. 507.

[7] S. Ben-Jamaa, C. Weidmann, and M. Kieffer, "Analytical tools for optimizing the error correction performance of arithmetic codes," *IEEE Trans. Commun.*, vol. 56, no. 9, pp. 1458–1468, September 2008.

[8] P. Elias, "Coding for noisy channels," *IRE National Convention Record*, vol. 3, no. 4, pp. 37–47, 1955.

[9] A. J. Viterbi, "Convolutional codes and their performance in communication systems," *IEEE Trans. Commun. Technol*, vol. 19, no. 5, pp. 751–772, 1971.

[10] J. Astola, "Convolutional code for phase-modulated channels," *Cybernetics and systems*, vol. 17, no. 1, pp. 89–101, 1986.

[11] M. Cedervall and R. Johannesson, "A fast algorithm for computing distance spectrum of convolutional codes," *IEEE trans. Inform. Theory*, vol. 35, no. 6, pp. 1146–1159, 1989.

[12] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory*, vol. 28, no. 1, pp. 55–67, 1982.

[13] E. Biglieri, "High-level modulation and coding for nonlinear satellite channels," *IEEE Trans. Commun*, vol. 32, no. 5, pp. 616–626, 1984.

[14] G. D. Forney, "Geometrically uniform codes," *IEEE Trans. Inform. Theory*, vol. 37, no. 5, pp. 1241–1260, 1991.

[15] E. Zehavi and J. K. Wolf, “On the performance evaluation of tellis codes,” *IEEE Trans. Inform. Theory*, vol. 33, no. 2, pp. 616–202, 1987.

[16] V. Buttigieg, “Variable-length error correcting codes,” PhD dissertation, University of Manchester, Univ. Manchester, U.K., 1995.

[17] S. Even, “Test for synchronizability of finite automata and variable length codes,” *IEEE Trans. Inform. Theory*, vol. 10, no. 3, pp. 185–189, 1964.

[18] J. C. Maxted and J. P. Robinson, “Error recovery for variable length code,” *IEEE Trans. Inform Theory*, vol. 31, no. 6, pp. 794–801, 1985.

[19] S. Malinowski, H. Jegou, and C. Guillemot, “Error recovery properties and soft decoding of quasi-arithmetic codes,” *EURASIP Journal on Advances in Signal Processing*, vol. 2008, no. 1, pp. 1–12, 2008.

[20] C. Weidmann and M. Kieffer, “Evaluation of the distance spectrum of variable-length finite-state codes,” *accepted for publication in IEEE. Trans. Commun*, 2009.

[21] I. H. Witten, R. M. Neal, and J. G. Cleary, “Arithmetic coding for data compression,” *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[22] C. Christopoulos, A. Skodras, and T. Ebrahimi, “The JPEG-2000 still image coding system: An overview,” *IEEE Trans. Consumer Electron.*, vol. 46, no. 4, pp. 1103–1127, 2000.

[23] I. E. G. Richardson, “H.264 and MPEG-4 video compression: Video coding for next generation multimedia,” *Chichester, West Sussex, UK: John Wiley & Sons*, 2003.

[24] C. Boyd, J. Cleary, I. Irvine, I. Rinsma-Melchert, and I. Witten, “Integrating error detection into arithmetic coding,” *IEEE Trans. Commun.*, vol. 45, no. 1, pp. 1–3, 1997.

[25] J. Sayir, “Arithmetic coding for noisy channels,” *Proc. IEEE Inform. Theory Workshop*, pp. 69–71, 1999.

[26] P. G. Howard and J. S. Vitter, “Practical implementations of arithmetic coding.” *Image and Text Compression*, vol. 13, no. 7, pp. 85–112, 1992.

[27] M. Gondran and M. Minoux, *Graphs and algorithms*. Chichester, UK: Wiley, 1984.

[28] S. J. Mason, “Feedback theory : Further properties of signal flow graphs,” *Proceedings of the IRE*, vol. 44, no. 7, pp. 920–926, 1956.

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

[30] R. C. Pasco, “Source coding algorithms for fast data compression,” Ph.D. Thesis, Stanford University, Dept. of EE Stanford University CA, USA, 1976.

[31] 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.*, pp. 143–152, 2006.

[32] I. Richardson, *H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia*. John Wiley and Sons, 2003.

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

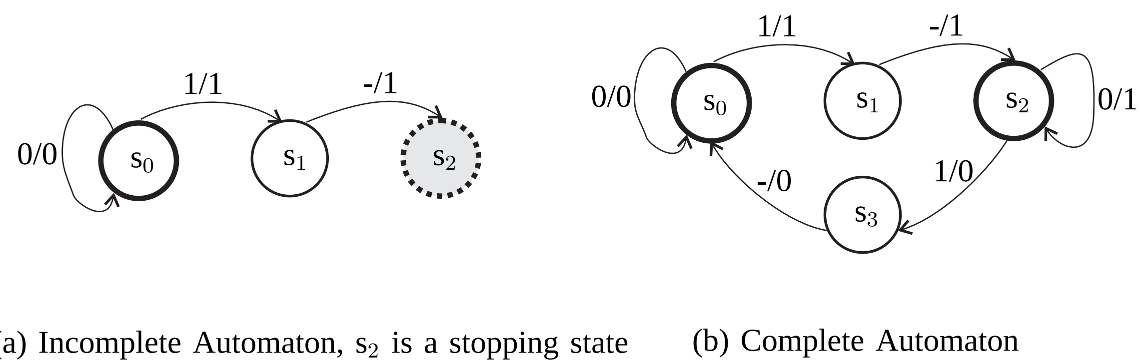

Fig. 1. Example of an incomplete automaton (a) and a complete automaton (b) derived from the automaton (a)

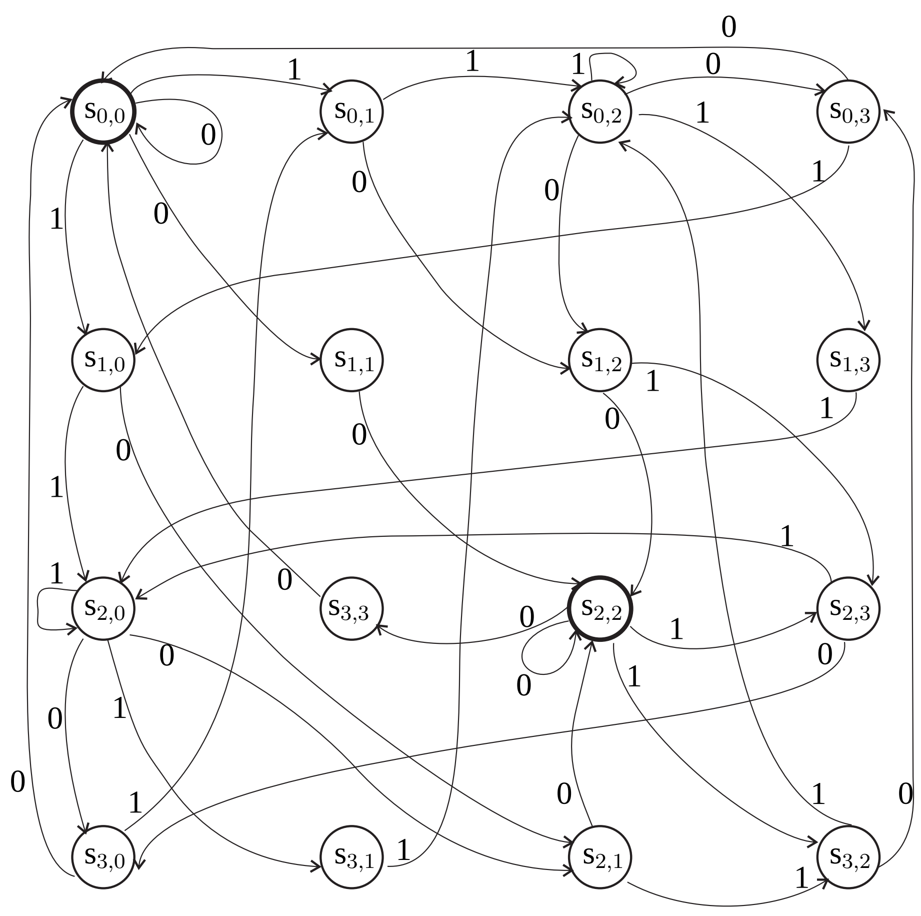

Fig. 2. Product graph derived from the B-FSE of Figure 1 (b)

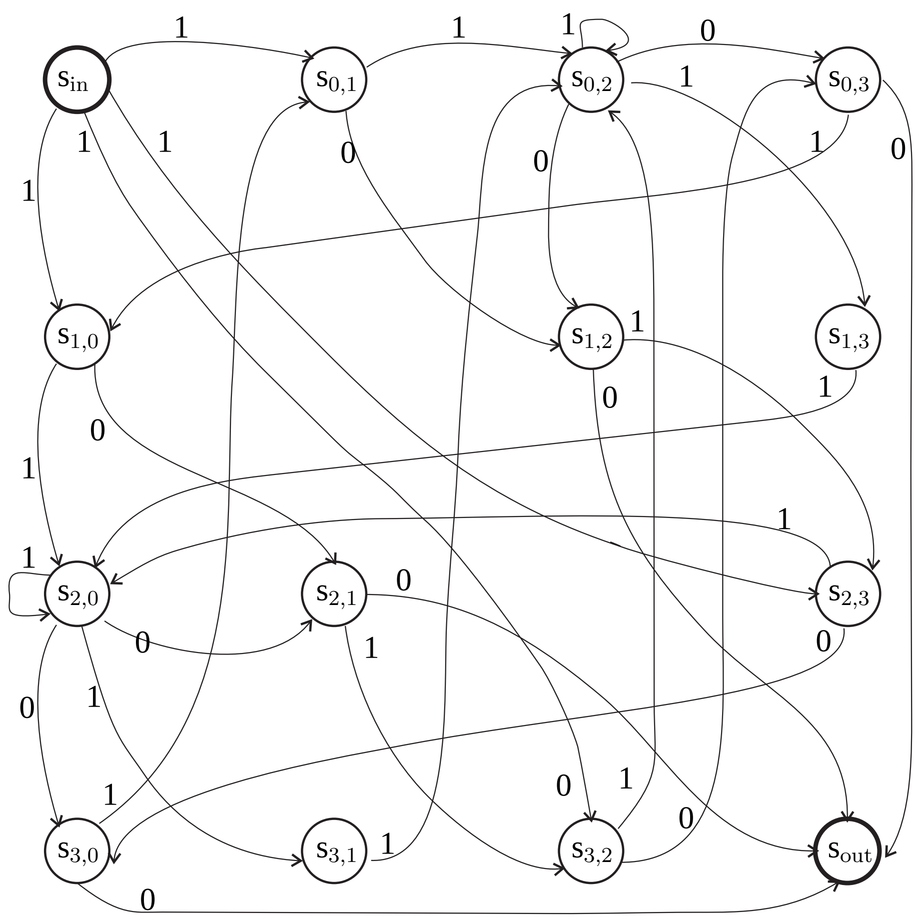

Fig. 3. The Modified Product Graph (MPG)

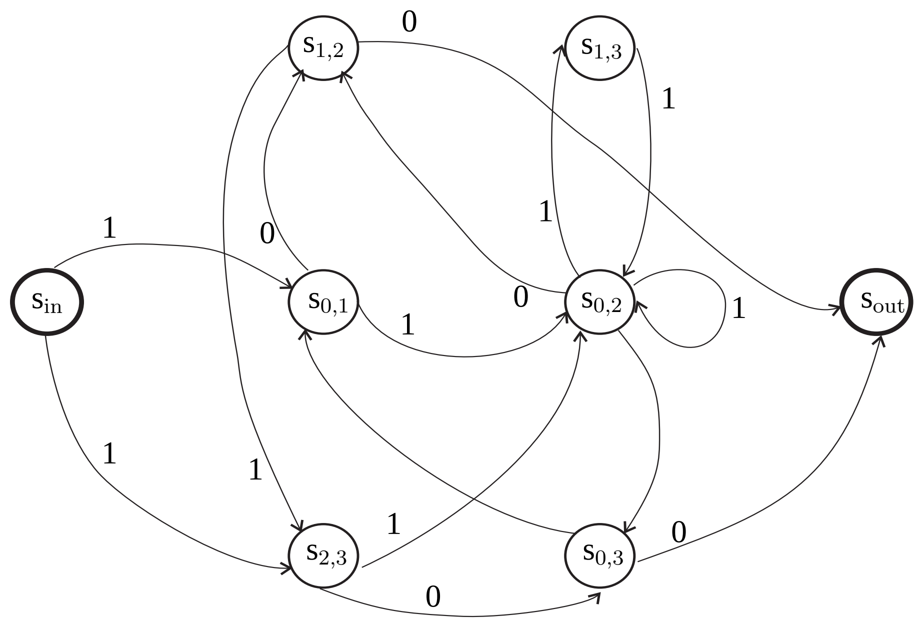

Fig. 4. The Pairwise Distance Graph (PDG)

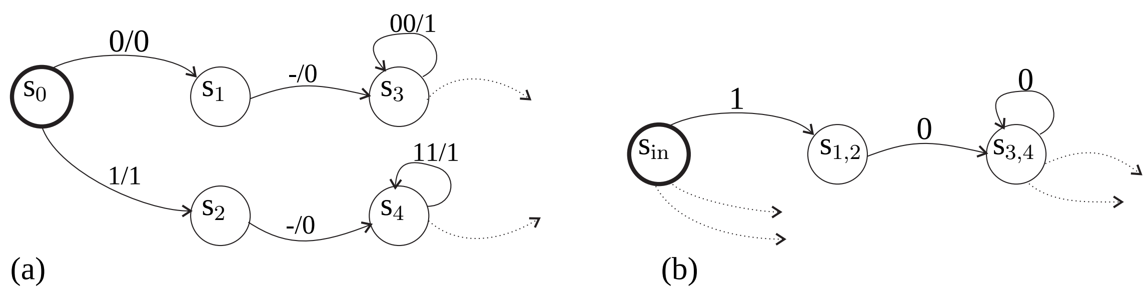

Fig. 5. Part of a bit clock finite state encoder (a) and the part of the corresponding pairwise distance graph (b)

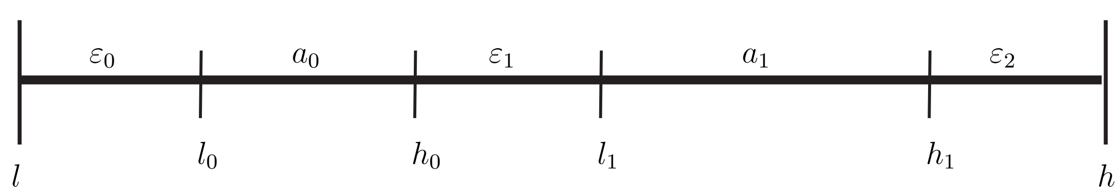

Fig. 6. Partitioning the coding interval in the more general binary input JSC-IAC case

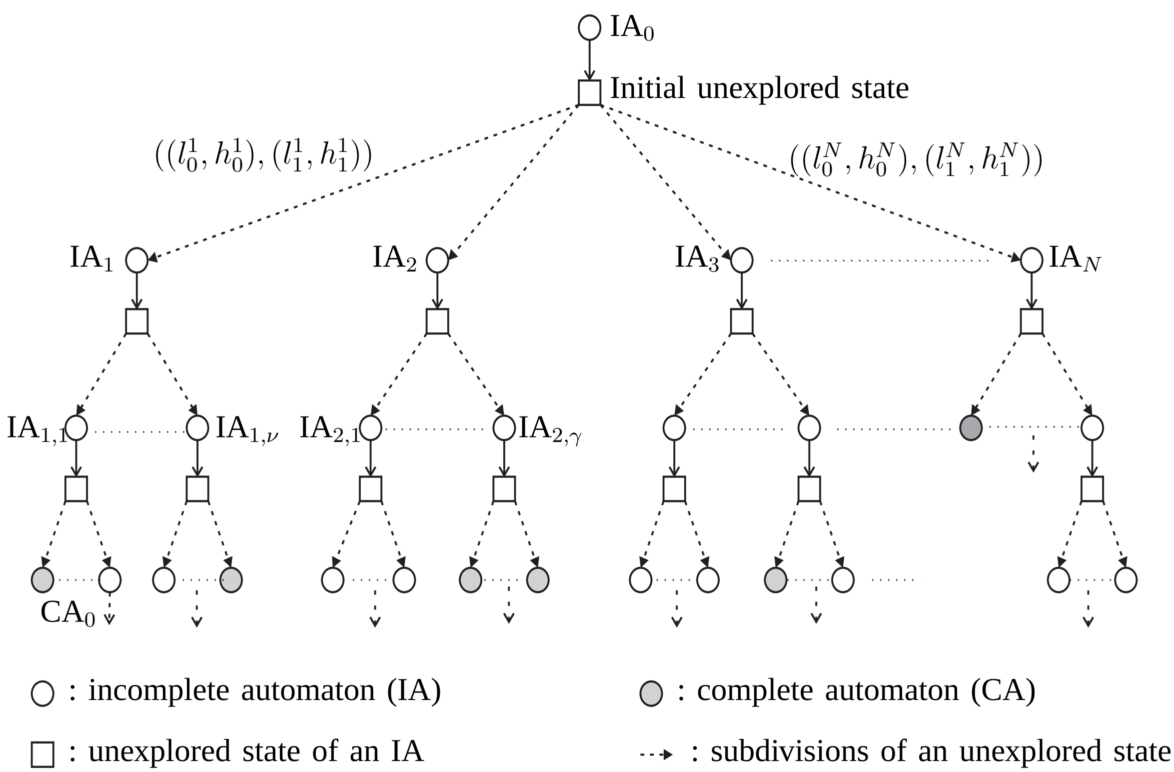

Fig. 7. All automata for given values of characteristic parameters on a tree

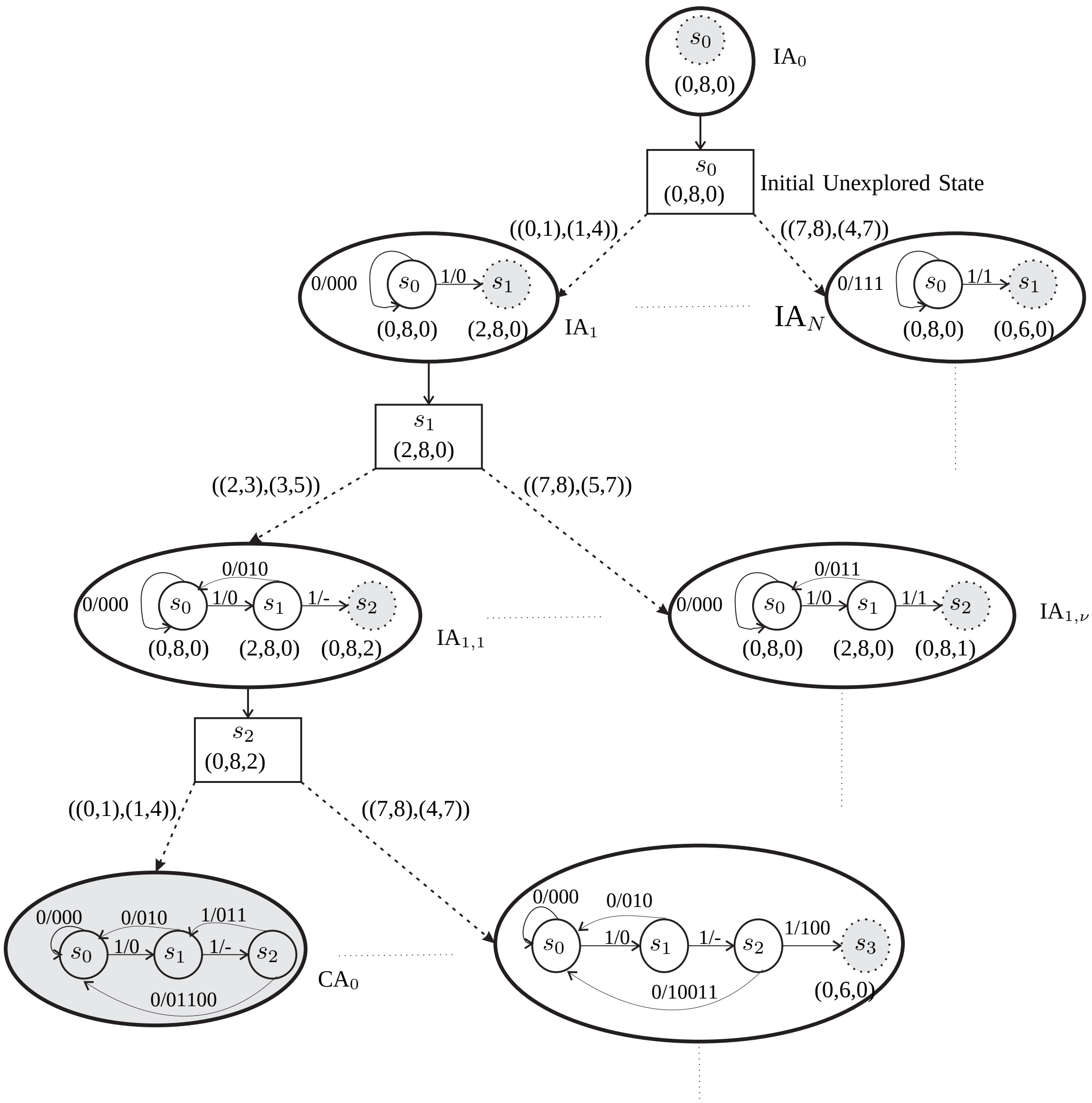

Fig. 8. Part of the tree of automata for the characteristic parameter values $T = 8$, $f_{\max} = 1$, $p_0 = \frac{1}{4}$, $P_\varepsilon = \frac{1}{2}$; the conventions of Figure 7 are used, the labels on the dotted arrows represent the intervals allotted $((l_0, h_0), (l_1, h_1))$ to the symbols a_0 and a_1.

Symbols	Probabilities	Bits assignment
$a_1 = E$	$p_{a_1} = 0.1270$	0 0 0 0 0
$a_2 = T$	$p_{a_2} = 0.0906$	0 0 0 0 1
$a_3 = A$	$p_{a_3} = 0.0817$	0 0 0 1 0
$a_4 = O$	$p_{a_4} = 0.0751$	0 0 0 1 1
$a_5 = I$	$p_{a_5} = 0.0697$	0 0 1 0 0
$a_6 = N$	$p_{a_6} = 0.0674$	0 0 1 0 1
$a_7 = S$	$p_{a_7} = 0.0633$	0 0 1 1 0
$a_8 = H$	$p_{a_8} = 0.0609$	0 0 1 1 1
$a_9 = R$	$p_{a_9} = 0.0599$	0 1 0 0 0
$a_{10} = D$	$p_{a_{10}} = 0.0425$	0 1 0 0 1
$a_{11} = L$	$p_{a_{11}} = 0.0403$	0 1 0 1 0
$a_{12} = C$	$p_{a_{12}} = 0.0278$	0 1 0 1 1
$a_{13} = U$	$p_{a_{13}} = 0.0276$	0 1 1 0 0
$a_{14} = M$	$p_{a_{14}} = 0.0241$	0 1 1 0 1
$a_{15} = W$	$p_{a_{15}} = 0.0236$	0 1 1 1 0
$a_{16} = F$	$p_{a_{16}} = 0.0223$	0 1 1 1 1
$a_{17} = G$	$p_{a_{17}} = 0.0202$	1 0 0 0 0
$a_{18} = Y$	$p_{a_{18}} = 0.0197$	1 0 0 0 1
$a_{19} = P$	$p_{a_{19}} = 0.0193$	1 0 0 1 0
$a_{20} = B$	$p_{a_{20}} = 0.0149$	1 0 0 1 1
$a_{21} = V$	$p_{a_{21}} = 0.0098$	1 0 1 0 0
$a_{22} = K$	$p_{a_{22}} = 0.0077$	1 0 1 0 1
$a_{23} = J$	$p_{a_{23}} = 0.0015$	1 0 1 1 0
$a_{24} = X$	$p_{a_{24}} = 0.0015$	1 0 1 1 1
$a_{25} = Q$	$p_{a_{25}} = 0.001$	1 1 0 0 0
$a_{26} = Z$	$p_{a_{26}} = 0.0007$	1 1 0 0 1

TABLE I

PROBABILITY OF OCCURENCE OF EACH LETTER IN ENGLISH ALPHABET TAKEN FROM [16] AND EXAMPLE OF BINARIZATION

Methods	depth-first	breadth-first	sort method
$\|\mathcal{S}_r\|$	8	2	3
$\|\mathcal{T}_r\|$	28	9	12
d_{free}	3	3	3
R_{c}	0.93	0.92	0.92
Time with [7]	451266 s	31338 s	12431 s
Time with PDG	734 s	219 s	45 s

TABLE II

COMPARISON BETWEEN THE THREE METHODS TO EXPLORE THE TREE FOR $T = 16$, $\mathrm{P}_0 = 0.1$, $P_\varepsilon = 0.26$