New Free Distance Bounds and Design Techniques for Joint Source-Channel Variable-Length Codes

Amadou Diallo, *Student Member, IEEE,* Claudio Weidmann, *Member, IEEE,*
and Michel Kieffer *Senior Member, IEEE*

***Abstract*—This paper proposes branch-and-prune algorithms for searching prefix-free joint source-channel codebooks with maximal free distance for given codeword lengths. For that purpose, it introduces improved techniques to bound the free distance of variable-length codes.**

***Index Terms*—Variable-length codes, finite state machines, source codes, channel codes, joint source-channel codes.**

I. Introduction

WHEN designing Joint Source-Channel (JSC) Variable-Length Codes (VLCs), one aims at building low-complexity codes simultaneously providing good data compression and error correction capabilities. The hope is to obtain joint codes outperforming separate codes (in terms of error rate for a given complexity or in terms of complexity for a given error rate) when the length of the codes is constrained [1], [2]. The compression efficiency of a code is measured by the ratio of the average codeword length to the source entropy [3], while its error-correction performance may be predicted with a union bound using the *distance properties* of the code, *i.e.*, its *free distance* and its *distance spectrum*, see [1].

A. Related work

JSC-VLC construction methods can be categorized according to how prefix, suffix, and distance properties, average codeword length, *etc.* enter the process. On one extreme are methods that guarantee some properties at each step of the construction. At the other extreme, one considers an exhaustive list of codebooks, whose properties are examined to find the best one.

Bidirectional or *Reversible Variable-Length Codes* (RVLCs), introduced in [4], are instantaneously decodable both in the forward and backward direction. RVLC design generally aims to minimize the average codeword length, see, *e.g.*, [5]–[11], usually without accounting for constraints on the free distance.

Manuscript received October 21, 2011; revised February 7, 2012.

A. Diallo and M. Kieffer are with L2S – CNRS - SUPELEC - Univ Paris-Sud, France. M. Kieffer is on leave at LTCI – Telecom ParisTech, France. C. Weidmann is with ETIS – CNRS UMR 8051 / ENSEA / Univ Cergy-Pontoise, France. Formerly, he was with INTHFT, Vienna University of Technology, Austria.

This work was supported by the European Commission in the framework of the FP7 Network of Excellence in Wireless COMmunications NEWCOM++ (contract n. 216715). Michel Kieffer is partly supported by the Institut Universitaire de France. Parts of this paper have been presented at MMSP 2010.

Several extensions of the design techniques for RVLCs to the construction of VLCs with larger free distance have been considered. A simple extension of [6] to VLCs with free distance greater or equal to 2 is given in [10], [12]. This is done by imposing a minimum distance between codewords of the same length. In [13], the synthesis of even-weight VLCs is considered. This guarantees minimum distance 2 or more between sequences of codewords.

Two techniques for building JSC-VLCs with a free distance larger than 2 have been proposed in [14], [15]. The first starts with a channel code, whose codewords are shortened while preserving some distance property. The second progressively builds codewords ensuring some *diverging* distance, *i.e.*, the distance between *prefixes*, which lower bounds the distance between codewords. These techniques were improved in terms of complexity by [16] and [9]. In [17], a genetic algorithm based code design is proposed, which maximizes the compression efficiency while satisfying a lower bound on the free distance. This approach complements the free distance lower bound in [14], [15] with a real-valued correction term involving a dissimilarity measure of codewords limiting the free distance bound and the codeword occurrence probability. The SAT-based approach proposed in [18], [19] may also incorporate constraints on the diverging, converging and block distances of codewords. This allows to obtain codes with the requested distance property (the lower bound is guaranteed to be satisfied), but not necessarily the code with the optimum coding efficiency.

As already mentioned, distance properties of JSC-VLCs were first evaluated in [14], [20], where a lower bound on the free distance and exhaustive (exponential complexity) algorithms for the distance spectrum were proposed. More recently, [21] considered VL-FSCs generated by variable-length finite-state encoders (VL-FSEs) and proposed a polynomial complexity matrix method to evaluate the exact distance spectrum in the code domain or an upper bound on it. In [22], Dijkstra's algorithm is applied on a product graph derived from the VL-FSE to compute the free distance of VL-FSCs. This method outperforms all previous methods in terms of complexity of distance evaluation.

B. The three main contributions of this paper

First, for given codeword lengths, the set of all JSC-VLCs is organized in a way to allow an exploration with a branch-and-prune algorithm, extending the idea proposed

in [22], [23] for JSC arithmetic codes. Second, our search criterion is the *exact* free distance (instead of a lower bound), which may be evaluated for JSC-VLCs using the techniques presented in [22]. Third, for more efficient branch pruning in the proposed search algorithm, we introduce or improve several free distance bounds for VLCs.

Section II provides definitions of structure and distance properties of JSC-VLCs and recalls the free distance evaluation technique using Dijkstra's algorithm. Section III introduces several bounds on the free distance of JSC-VLCs. Section IV describes how the search space for good JSC-VLCs may be structured using a tree and explored with a branch-and-prune algorithm. Section V provides experimental results, before drawing some conclusions in Section VI.

II. Joint Source-Channel Variable-Length Codes: JSC-VLC

Consider an M-ary memoryless source X with alphabet $\mathcal{A} = \{a_1, a_2, \ldots, a_M\}$ and associated probabilities $\boldsymbol{p} = (p_1, p_2, \ldots, p_M)$. A (JSC-)VLC encoder C maps symbol $a_i \in \mathcal{A}$ to a variable-length binary codeword $c_i = C(a_i)$. The set of codewords $\mathcal{C} = \{c_1, c_2, \ldots, c_M\}$ forms the *codebook*. The length of codeword c_i is ℓ_i and its j-th bit is c_i^j. If the code $\mathcal{C}$ is prefix-free, then the lengths $\boldsymbol{\ell} = (\ell_1, \ell_2, \ldots, \ell_M)$ satisfy Kraft's inequality $K(\boldsymbol{\ell}) = \sum_{i=1}^{M} 2^{-\ell_i} \leqslant 1$ [3, Ch. 5.2]. Henceforth $\boldsymbol{\ell}$ is called a *Kraft vector* when it satisfies Kraft's inequality.

The performance of a JSC-VLC is determined by its *redundancy* and its *error correcting capability*. The redundancy ρ is the difference between average codeword length $\ell_{\text{av}} = \sum_{i=1}^{M} p_i \ell_i$ and source entropy $H = -\sum_{i=1}^{M} p_i \log_2 p_i$. Thus $\rho = \ell_{\text{av}} - H = \sum_{i=1}^{M} p_i(\ell_i + \log_2 p_i)$. The error correcting capability is primarily characterized by the *free distance* d_{free}, which is the minimal Hamming distance between two distinct semi-infinite sequences of codewords. A finer characterization is possible through the *distance spectrum*, see [1].

To evaluate distance properties, a graphical representation of a JSC-VLC is better suited than a list of codewords. Indeed, an encoder C can be represented by a directed graph $\Gamma(\mathcal{S}, \mathcal{T})$, where $\mathcal{S}$ is a set of states (vertices) and $\mathcal{T}$ is a set of transitions (directed edges). Each transition is labeled with an input symbol and a variable-length sequence of output bits. Γ represents a *finite-state encoder* (FSE) associated to a *variable-length finite-state code* (VL-FSC). In the simple case of a JSC-VLC, $\mathcal{S}$ contains a single state s_0 from/to which all transitions leave/lead, so $\mathcal{T}$ contains M transitions associated to the symbols in $\mathcal{A}$. Each transition $u_i \in \mathcal{T}$ has an input label $I(u_i) = a_i$ and an output label $O(u_i) = c_i$.

For the purpose of distance evaluation, the FSE is transformed into a *bit-clock* FSE (B-FSE), in which each transition is labeled with exactly one output bit and may have an empty input label. Transitions with multiple output bits are replaced by chains of consecutive transitions, each with one output bit, and only the first one inheriting the input symbol.

Example 1: Fig. 1 (top) shows the FSE associated to a source $X_{(3)}$ with alphabet $\mathcal{A}_{(3)} = \{a_1 = a, a_2 = b, a_3 = c\}$ encoded using the codebook $\mathcal{C}_3 = \{c_1 = 0, c_2 = 10, c_3 = 111\}$.

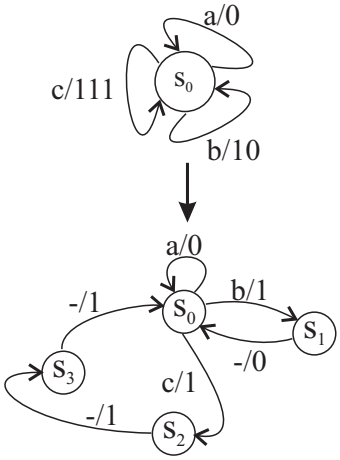

Fig. 1. Example of FSE associated to $\mathcal{A}_3$ and $\mathcal{C}_3$ (top) and its corresponding bit-clock representation (bottom)

For each transition, a slash (/) separates input symbol from output bits and a hyphen ($-$) stands for no input symbol. Fig. 1 (bottom) shows the B-FSE derived from the FSE of Fig. 1 (top).

The following definitions are for general FSEs and will be specialized to VLCs as needed. Let $\sigma(u)$ be the originating state of some transition $u \in \mathcal{T}$ and $\tau(u)$ its target state. A path $\boldsymbol{u} = (u_1, u_2, \ldots, u_k) \in \mathcal{T}^k$ on the graph is a concatenation of transitions that satisfy $\sigma(u_{i+1}) = \tau(u_i)$ for $1 \leqslant i < k$. By extension, we define $\sigma(\boldsymbol{u}) = \sigma(u_1)$ and $\tau(\boldsymbol{u}) = \tau(u_k)$, as well as $I(\boldsymbol{u})$ and $O(\boldsymbol{u})$, which are the concatenations of the input, respectively output, labels of $\boldsymbol{u}$. Finally, $\ell(\boldsymbol{x})$ is the length (in symbols or bits) of the sequence $\boldsymbol{x}$. The Hamming distance d_{H} between two equal-length sequences $\boldsymbol{x}, \boldsymbol{y}$ equals the Hamming weight, *i.e.*, the number of non-zero entries, of their elementwise difference, $d_{\text{H}}(\boldsymbol{x}, \boldsymbol{y}) = w_{\text{H}}(\boldsymbol{x} - \boldsymbol{y})$. If two paths $(\boldsymbol{u}_1, \boldsymbol{u}_2) \in \mathcal{T}^{k_1} \times \mathcal{T}^{k_2}$ are such that $\ell(O(\boldsymbol{u}_1)) = \ell(O(\boldsymbol{u}_2))$, then we will write $d_{\text{H}}(\boldsymbol{u}_1, \boldsymbol{u}_2) = d_{\text{H}}(O(\boldsymbol{u}_1), O(\boldsymbol{u}_2))$.

Definition 1: For a FSE $\Gamma(\mathcal{S}, \mathcal{T})$ representing a VLC C, let $\mathcal{P}$ be the set of all pairs of paths in $\left(\mathcal{T}^{k_1} \times \mathcal{T}^{k_2}\right)_{1 \leqslant k_1, k_2 < \infty}$ diverging from s_0 and converging for the first time in s_0 with the same output length. Then the free distance is the minimum Hamming distance in $\mathcal{P}$,

$$d_{\text{free}} = \min_{(\boldsymbol{u}_1, \boldsymbol{u}_2) \in \mathcal{P}} d_{\text{H}}(\boldsymbol{u}_1, \boldsymbol{u}_2). \tag{1}$$

The *Pairwise Distance Graph* (PDG) is a modified product graph of the B-FSE that tracks the Hamming distances between pairs of paths in $\mathcal{P}$, which we introduced in [22]. It is constructed and used as follows. Let $\mathcal{S}_b$ and $\mathcal{T}_b$ be the set of states and transitions of the B-FSE graph $\Gamma_b(\mathcal{S}_b, \mathcal{T}_b)$, yielding the directed product graph $\Gamma_b^2(\mathcal{S}_b \times \mathcal{S}_b, \mathcal{T}_b \times \mathcal{T}_b)$. The weight $w_{\text{H}}(e)$ of edge $e = (u, v) \in \mathcal{T}_b^2$ is the Hamming distance between the outputs of transitions u and v, $w_{\text{H}}(e) = d_{\text{H}}(u, v)$. A directed path $\boldsymbol{e}$ in Γ_b^2 is a sequence of edges $\boldsymbol{e} = (e_1, e_2, \ldots, e_n)$ such that $\sigma(e_{\mu+1}) = \tau(e_\mu)$ for $1 \leqslant \mu < n$; its weight is $w_{\text{H}}(\boldsymbol{e}) = \sum_{\mu=1}^{n} w_{\text{H}}(e_\mu)$.

Consider the set $\mathcal{S}_{\text{div}}$ of states $(s_i, s_i) \in \mathcal{S}_b^2$ from which distinct transitions are diverging, as well as the set $\mathcal{S}_{\text{conv}}$ of states (s_i, s_i) to which distinct transitions are converging. One

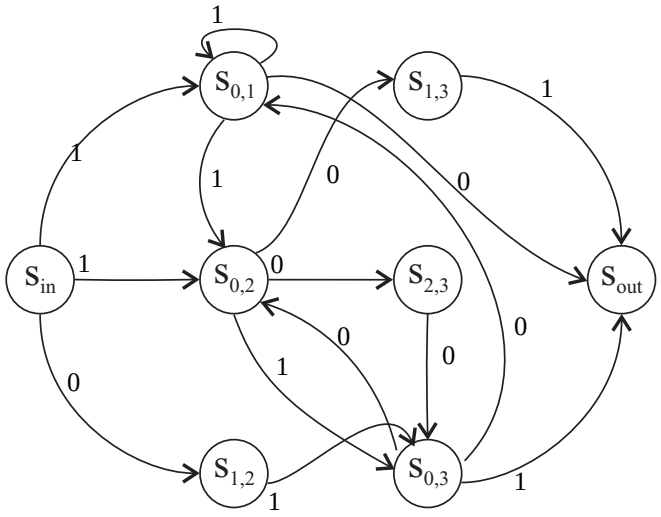

Fig. 2. Pairwise distance graph derived from the B-FSE in Fig. 1 (bottom)

obtains the PDG by merging all states in $\mathcal{S}_{\text{div}}$ into a single state s_{in} and those in $\mathcal{S}_{\text{conv}}$ into a single state s_{out}, as well as merging symmetric states (s_i, s_j) and (s_j, s_i), $i < j$, into a single state (s_i, s_j). In the simple case of VLCs, one has $s_{\text{in}} = (s_0, s_0)$ and $s_{\text{out}} = (s_0, s_0)$. Fig. 2 shows the PDG derived from the B-FSE in Fig. 1 (bottom).

Finding d_{free} with the PDG is equivalent to finding a directed minimal weight path from s_{in} to s_{out}. This is a well-known shortest weighted path problem and can be solved efficiently using Dijkstra's algorithm [24], since all weights are non-negative.

The code optimization techniques described in Section IV start from a Kraft vector $\boldsymbol{\ell}$ specifying the codeword lengths and successively determine the codebook, *i.e.*, the bits c_i^j of the codewords. Bounds on the free distance of partially determined codebooks will thus be key ingredients.

Definition 2: A codebook is *undetermined* if none of the bits c_i^j are known (determined); it is *partially determined* if some bits c_i^j are known and it is *fully determined* if all bits c_i^j are known.

Definition 3: A partially/fully determined codebook $\mathcal{C}^1$ is *deduced* from another partially determined or undetermined codebook $\mathcal{C}^0$ (denoted $\mathcal{C}^0 \prec \mathcal{C}^1$) if it is obtained by specifying some/all of the undetermined bits in $\mathcal{C}^0$.

Example 2: Consider codebooks $\mathcal{C}^0 = \{0, c_2^1 c_2^2, c_3^1 c_3^2 c_3^3\}$, $\mathcal{C}^1 = \{0, 11, c_3^1 c_3^2 c_3^3\}$, and $\mathcal{C}^2 = \{0, 10, 111\}$. $\mathcal{C}^0$ and $\mathcal{C}^1$ are partially determined codebooks, while $\mathcal{C}^2$ is fully determined. Moreover, $\mathcal{C}^0 \prec \mathcal{C}^1$ and $\mathcal{C}^0 \prec \mathcal{C}^2$, but $\mathcal{C}^1 \not\prec \mathcal{C}^2$ and $\mathcal{C}^2 \not\prec \mathcal{C}^1$.

III. Bounding the free distance of JSC-VLCs

A. Some existing bounds

Consider a (nonlinear) block code $C(n, M, d)$, where n is the block length, M is the number of codewords and d is the minimum Hamming distance of the code. For $M > 1$, Plotkin's upper bound [25, p. 167] yields

$$d \leqslant \bar{d}^{\text{p}}(n, M) = \left\lfloor \frac{nM}{2(M-1)} \right\rfloor, \tag{2}$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x. For odd M, this can be tightened to $d \leq \lfloor n(M+1)/(2M) \rfloor$. For $M \in \{0, 1\}$, we define $\bar{d}^{\text{p}}(n, M) = \infty$.

Heller used Plotkin's bound to upper bound the free distance of convolutional codes, by counting the number of sequences that lead from the zero state back to itself. Heller's approach (referred in [26, Chap. 3.5]) extends to nonlinear time-varying trellis codes (which output a fixed number of bits per transition), but it does not immediately extend to VL-FSCs, which include JSC-VLCs. This will be done in Section III-C.

For a VLC with fully determined codebook, Buttigieg [14] derived lower and upper bounds on the free distance,

$$d_{\text{block}} \geqslant d_{\text{free}} \geqslant \min(d_{\text{block}}, d_{\text{div}} + d_{\text{conv}}), \tag{3}$$

where d_{block} is the *overall minimum block distance* (between equal-length codewords), d_{div} is the *minimum diverging distance* (between prefixes of unequal-length codewords) and d_{conv} is the *minimum converging distance* (between suffixes of unequal-length codewords), respectively.

B. Simple application of Plotkin's bound

If only the Kraft vector $\boldsymbol{\ell}$ is known, one may apply Plotkin's bound to d_{block} to obtain an upper bound on d_{free}. Let L be the largest entry of $\boldsymbol{\ell}$ and let M_r, $1 \leqslant r \leqslant L$, be the number of entries in $\boldsymbol{\ell}$ equal to r. Apply (2) to the block codes $C_r(r, M_r, d_r)$, $1 \leqslant r \leqslant L$, formed by the subset of codewords of length r, and minimize to yield an extension of Plotkin's upper bound to the free distance of a JSC-VLC,

$$d_{\text{free}} \leqslant \bar{d}^{\text{pe}}_{\text{free}}(\boldsymbol{\ell}) = \min_{1 \leqslant r \leqslant L} \bar{d}^{\text{p}}(r, M_r). \tag{4}$$

Example 3: Consider the Kraft vector $\boldsymbol{\ell} = (2, 3, 5, 5)$. One has $M_0 = M_1 = M_4 = 0$, $M_2 = M_3 = 1$, and $M_5 = 2$. Applying the extension of Plotkin's bound to $\boldsymbol{\ell}$ yields $\bar{d}^{\text{pe}}_{\text{free}}(\boldsymbol{\ell}) = 5$, while the maximum free distance that can actually be achieved with $\boldsymbol{\ell}$ is $d_{\text{free}} = 2$.

C. Extension of Heller's bound

Heller's upper bound on d_{free} of convolutional codes is based on counting the number of code sequences of length n and then applying Plotkin's bound [26, Chap. 3.5]. This section extends Heller's approach to VL-FSCs, which include JSC-VLCs.

Let $\Gamma(\mathcal{S}, \mathcal{T})$ be the graph of the FSE associated to a VL-FSC with S states, $\mathcal{S} = \{0, \ldots, S-1\}$ without loss of generality. Only the length of the output label (codeword) of each transition needs to be known to count the number of code sequences of length n. Let $L = \max_{u \in \mathcal{T}} \ell(O(u))$ the longest output label of all transitions. For a JSC-VLC, $S = 1$ and $L = \max(\ell_1, \ell_2, \ldots, \ell_M)$. Consider the integer matrices $A_r \in \mathbb{N}^{S \times S}$, $1 \leqslant r \leqslant L$, counting the transitions of length r,

$$(A_r)_{i,j} = |\{u : \ell(O(u)) = r \text{ and } \sigma(u) = i \text{ and } \tau(u) = j\}|, \tag{5}$$

where $|\mathcal{A}|$ is the cardinality of the set $\mathcal{A}$. Entry $(A_r)_{i,j}$ counts the number of transitions from state i to state j with an output label of length r. For a JSC-VLC, A_r reduces to a scalar counting the number of codewords of length r.

The row vector $\mathbf{m}_n \in \mathbb{N}^S$ holds entries $m_{n,s}$ equal to the number of sequences of length n ending in state s, $s = 0, \ldots, S-1$. For $n \geqslant 1$, the *count vector* $\mathbf{m}_n$ may be expressed as a function of preceding count vectors, $\mathbf{m}_n = \sum_{r=1}^{L} \mathbf{m}_{n-r} A_r$, which may be written

$$\mathbf{m}_n = [\mathbf{m}_{n-L}, \ldots, \mathbf{m}_{n-1}] \left[A_L^{\mathsf{T}} \ldots A_1^{\mathsf{T}}\right]^{\mathsf{T}}. \tag{6}$$

The recursion is initialized with the vector $[\mathbf{m}_{-L+1}, \ldots, \mathbf{m}_0] = [\mathbf{0}, \ldots, \mathbf{0}, \mathbf{e}_i]$, having $m_{0,i} = 1$ for initial state i and all other entries equal to zero. Now, (6) may be extended to compute a *block* of L count vectors $[\mathbf{m}_n, \ldots, \mathbf{m}_{n+L-1}]$ based on the preceding block of L count vectors, thus making it possible to track the behavior on an L-block basis. One obtains

$$[\mathbf{m}_n, \ldots, \mathbf{m}_{n+L-1}] = [\mathbf{m}_{n-L}, \ldots, \mathbf{m}_{n-1}] \cdot A, \tag{7}$$

where $A = \left[A^{(0)}, \ldots, A^{(L-1)}\right]$ is a $LS \times LS$ matrix composed of $LS \times S$ blocks $A^{(k)} = \sum_{j=0}^{k} A_{\rightarrow(k-j)} B_j$. The latter expression is computed from the basic $LS \times S$ block $[A_L^{\mathsf{T}}, \ldots, A_1^{\mathsf{T}}]^{\mathsf{T}}$ shifted down iS positions,

$$A_{\rightarrow(i)} = [\underbrace{\mathbf{0}, \ldots, \mathbf{0}}_{i \text{ times}}, \; A_L^{\mathsf{T}}, \ldots, A_{1+i}^{\mathsf{T}}]^{\mathsf{T}},$$

where $\mathbf{0}$ denotes a block of $S \times S$ zeros. It also involves the sum of all products of A_r such that their indices sum to $j = 1, \ldots, L-1$, given by

$$B_j = \sum_{\substack{(r_1, \ldots, r_i): \\ r_1 + \cdots + r_i = j}} A_{r_1} \cdots A_{r_i}. \tag{8}$$

The sum (8) is over all *ordered* partitions (*i.e.*, compositions) of j into i non-negative integers $r_1, \ldots, r_i$ ($A_{r_k} = 0$ if there are no codewords of length r_k). Order is important, since matrix multiplication is non-commutative in general. By definition, $B_0 = I_S$, the $S \times S$ identity matrix. Using (7), all counts are computed from a given start state i in a recursive block fashion as[1]

$$[\mathbf{m}_{kL-L+1}(i), \ldots, \mathbf{m}_{kL}(i)] = [\mathbf{0}, \ldots, \mathbf{0}, \mathbf{e}_i] \, A^k, \;\; k = 1, 2, \ldots \tag{9}$$

The count of length-n sequences from state i to state j can also be extracted directly from the matrix A^k as $m_{n,j}(i) = \left(A^k\right)_{(L-1)S+i, \; rS+j}$, where $r = (n-1) \bmod L$ and $k = (n-1-r)/L + 1$. For VLCs this becomes $m_n = \left(A^k\right)_{L-1, \; r}$ (indices start at 0). These counts can then be inserted into Plotkin's bound to yield an upper bound on d_{free}.

Proposition 1: An extension of Heller's upper bound on the free distance of VL-FSCs is

$$d_{\text{free}} \leqslant \bar{d}_{\text{free}}^{\text{he}}(\boldsymbol{\ell}) = \min_{0 < n \leqslant n_{\max}} \min_{i,j \in \mathcal{S}} \bar{d}^{\text{p}}\left(n, m_{n,j}(i)\right), \tag{10}$$

where $n_{\max} \geq L$ is chosen according to computational constraints.

[1]Using this approach, some counts may be computed twice. Consider *e.g.* $L = 5$ and a codeword of length $\ell = 2$. Then the computations for $\mathbf{m}_{kL+3}(i)$ will include reevaluating $\mathbf{m}_{kL+1}(i)$. Thus the computational complexity could be reduced by a more elaborate scheduling of operations, at the cost of giving up the simple matrix representation (9). Alternatively, bit-clock intermediate states could be introduced, like for the code spectrum computation [21], but the corresponding matrix would be considerably larger.

Proof: The set of length-n paths between some start state i and some final state j forms a (nonlinear) block code, so Plotkin's bound applies. It remains to be shown that (9) correctly counts the number of paths $m_{n,j}(i)$, *i.e.*, that no path is counted twice. This is indeed the case in (7), since the definition of $A_{\rightarrow(i)}$ ensures that the first transition leads from the block $\left[\mathbf{m}_{(k-2)L+1}(i), \ldots, \mathbf{m}_{(k-1)L}(i)\right]$ into the block $\left[\mathbf{m}_{(k-1)L+1}(i), \ldots, \mathbf{m}_{kL}(i)\right]$, while B_j accounts for transitions within the latter block. ∎

To evaluate (10) for a VLC, it usually suffices to choose $n_{\max}$ such that $2L \leqslant n_{\max} \leqslant 3L$, since short lengths with multiplicities $A_\ell > 1$ and short concatenations tend to dominate the bound.

This technique can also be used to obtain bounds on the effective minimum distance of practical schemes transmitting finite-length blocks, where start state i and length n are known to the decoder. If there is no termination mechanism, the decoder does not know the final state and thus the bound becomes $\min_{j \in \mathcal{S}} \bar{d}^{\text{p}}(n, m_{n,j}(i))$. If the blocks are terminated in state j, the bound becomes $\bar{d}^{\text{p}}(n, m_{n,j}(i))$. This hints at the importance of a proper termination mechanism in order to prevent the minimum block distance falling below d_{free}.

Example 4: For a JSC-VLC with the Kraft vector of Example 3 and $n_{max} = 15$, one has $m_1 = 0$, $m_2 = m_3 = m_4 = 1$, $m_5 = 4$. This leads to $\bar{d}_{\text{free}}^{\text{he}} = 3$ (at $n = 5$), which is closer to the optimally achievable $d_{\text{free}} = 2$ than the bound of Example 3.

D. Bounding d_{free} using the pairwise distance graph

Given a Kraft vector $\boldsymbol{\ell}$, one may construct an undetermined codebook $\mathcal{C}^0$ in which all bits have symbolic labels c_i^j (see Section II). The edges of the PDG obtained from $\mathcal{C}^0$ will be labeled with sums of symbolic labels. The PDG of a partially determined codebook $\mathcal{C}^1$ that is deduced from $\mathcal{C}^0$ (*i.e.* $\mathcal{C}^0 \prec \mathcal{C}^1$) will have the same structure as the PDG of $\mathcal{C}^0$, with the difference that some (or all) labels c_i^j are determined to be 0 or 1. An edge $(u, v) \in \mathcal{T}_b \times \mathcal{T}_b$ with determined labels will have a determined label, while all other edges of the PDG (with at least one undetermined transition label) will have undetermined labels.

Example 5: $\mathcal{C}^1 = \{0, 10, 111\}$ is a determined codebook whose PDG is represented in Fig. 2, whereas $\mathcal{C}^0 = \left\{c_1^1, c_2^1 c_2^2, c_3^1 c_3^2 c_3^3\right\}$ is an undetermined codebook whose PDG is in Fig. 3.

The fact that all codebooks deduced from $\mathcal{C}^0$ (*i.e.* from $\boldsymbol{\ell}$) will have the same PDG structure allows to obtain a nested hierarchy of lower and upper bounds on d_{free}.

Theorem 2: The shortest weighted path obtained after replacing each undetermined edge label in the PDG of a partially determined codebook $\mathcal{C}^0$ by 0 (respectively 1) is a lower bound $\underline{d}_{\text{free}}^{\text{PDG}}(\mathcal{C}^0)$ (respectively upper bound $\overline{d}_{\text{free}}^{\text{PDG}}(\mathcal{C}^0)$) on the free distances of all JSC-VLC codebooks deduced from $\mathcal{C}^0$. This path can be found using Dijkstra's algorithm.

Proof: The proof is for the lower bound, the upper bound follows along the same lines. Let $\boldsymbol{e}_{\min}$ be a path with the smallest weight from state s_{in} to s_{out} in the PDG of $\mathcal{C}^0$, when replacing undetermined labels by 0. Let $d_{\text{free}}^{\min} = w_{\text{H}}(\boldsymbol{e}_{\min})$.

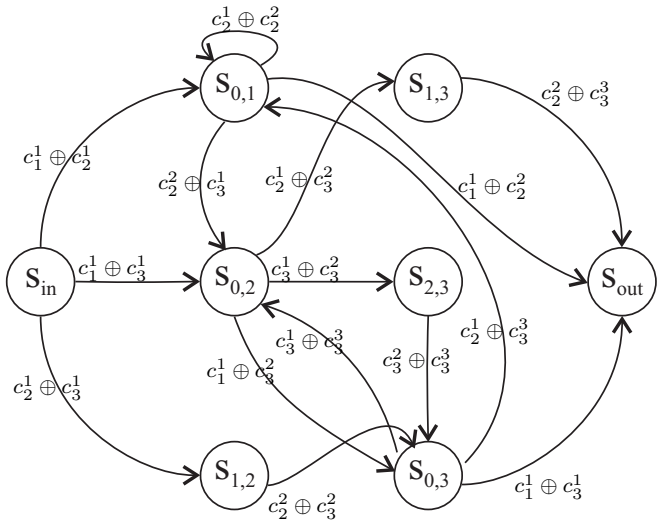

Fig. 3. Pairwise distance graph derived from the B-FSE of $\mathcal{C}^0 = \left\{c_1^1, c_2^1c_2^2, c_3^1c_3^2c_3^3\right\}$

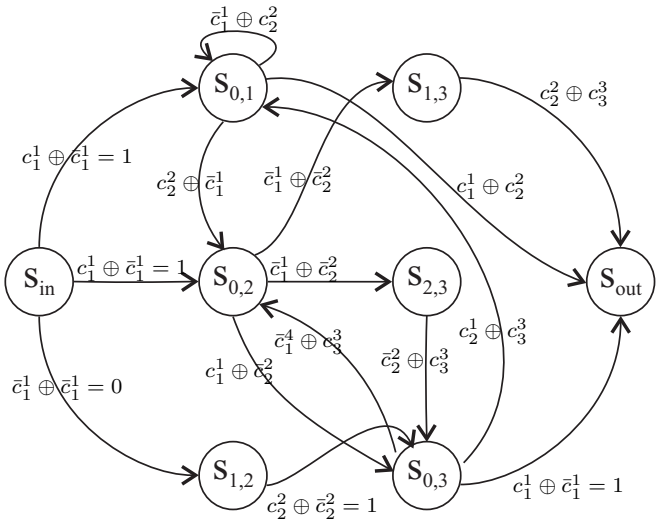

Fig. 4. Pairwise distance graph derived from the B-FSE of $\mathcal{C}^1 = \left\{c_1^1, \bar{c}_1^1c_2^2, \bar{c}_1^1\bar{c}_2^1c_3^3\right\}$, accounting for the prefix condition

Consider a deduced codebook $\mathcal{C}^1 \succ \mathcal{C}^0$, for which more PDG edge labels are determined compared to the PDG of $\mathcal{C}^0$. Since edge labels are non-negative, the weight of the path $\boldsymbol{e}_{\min}$ can only remain equal or increase. Hence, $d_{\text{free}}^{\min}$ is a lower bound on the free distance of $\mathcal{C}^1$. ■

A direct consequence of Theorem 2 is that the bounds enclosing the free distance of a deduced codebook $\mathcal{C}^1$ will be at least as tight as the bounds for the original codebook $\mathcal{C}^0 \prec \mathcal{C}^1$.

Corollary 3: If two codebooks $\mathcal{C}^0$ and $\mathcal{C}^1$ satisfy $\mathcal{C}^0 \prec \mathcal{C}^1$, then

$$\left[\underline{d}_{\text{free}}^{\text{PDG}}(\mathcal{C}^1), \overline{d}_{\text{free}}^{\text{PDG}}(\mathcal{C}^1)\right] \subseteq \left[\underline{d}_{\text{free}}^{\text{PDG}}(\mathcal{C}^0), \overline{d}_{\text{free}}^{\text{PDG}}(\mathcal{C}^0)\right]. \qquad (11)$$

The more bits are determined in a partially determined codebook, the more likely the bounds provided by Corollary 3 will be close to each other.

Example 6: Consider the undetermined code $\mathcal{C}^0 = \left\{c_1^1, c_2^1c_2^2, c_3^1c_3^2c_3^3\right\}$ whose PDG is represented in Fig. 3. Applying Theorem 2 to $\mathcal{C}^0$ leads to $\left[\underline{d}_{\text{free}}^{\text{PDG}}(\mathcal{C}^0), \overline{d}_{\text{free}}^{\text{PDG}}(\mathcal{C}^0)\right] = [0, 2]$. Thus, the free distance of any code deduced from $\mathcal{C}^0$ will be upper bounded by 2.

E. Bounding d_{free} of prefix-free JSC-VLCs

The above methods for obtaining free distance bounds do not take into account the condition that no codeword may be a prefix of another. Any prefix-free JSC-VLC codebook can be represented by a labeled binary tree with leaves mapped to the M source symbols, such that the codeword for a symbol can be read off as the concatenation of the binary labels from the root to the corresponding leaf, see [3].

Consider the set $\Theta(\boldsymbol{\ell})$ of unlabeled binary trees that may be associated to prefix-free JSC-VLC codebooks with Kraft vector $\boldsymbol{\ell}$, that is, the set of binary trees with M leaves at depths $\ell_1, \ell_2, \ldots, \ell_M$. The structure of a tree $T \in \Theta(\boldsymbol{\ell})$ determines which codewords have common prefixes as well as the length of those prefixes.

The free distance of any JSC-VLC codebook $\mathcal{C}$ admitting a tree T as representation can be bounded directly from the structure of the tree, without specifying any code bit. For that purpose, start by numbering the internal nodes of the tree, including the root. Then, for each internal node i, label one branch leaving the node with b_i and the other branch (if it exists) with $\bar{b}_i$. This labeling reflects the prefix condition on any VLC codebook $\mathcal{C}$ that can be obtained by labeling T. Now, when constructing the PDG for bounding the free distance of $\mathcal{C}$, the knowledge that $d_{\text{H}}(b_i, \bar{b}_i) = b_i \oplus \bar{b}_i = 1$ may be applied to edges of the PDG labeled $b_i \oplus \bar{b}_i$, while all other edge labels in the PDG are replaced with 0 or 1, depending on the type of bound being computed (see Section III-D). The resulting free distance bounds will be denoted $\underline{d}_{\text{free}}^{\text{T}}(T)$ and $\overline{d}_{\text{free}}^{\text{T}}(T)$.

Example 7: Consider again the undetermined code $\mathcal{C}^0 = \left\{c_1^1, c_2^1c_2^2, c_3^1c_3^2c_3^3\right\}$. Its PDG shown in Fig. 3 does not reflect the prefix condition. Accounting for the prefix condition, one would get, *e.g.*, $\mathcal{C}^1 = \left\{c_1^1, \bar{c}_1^1c_2^2, \bar{c}_1^1\bar{c}_2^1c_3^3\right\}$. The resulting PDG is represented in Fig. 4. Applying Theorem 2 to $\mathcal{C}^1$ leads to $[\underline{d}_{\text{free}}^{\text{T}}(\mathcal{C}^1), \overline{d}_{\text{free}}^{\text{T}}(\mathcal{C}^1)] = [1, 2]$, which is better than the enclosure provided by $\underline{d}_{\text{free}}^{\text{PDG}}$.

IV. Designing JSC-VLCs

Next we outline three methods for structuring JSC-VLCs in trees, *i.e.*, for creating hierarchies of partial codebooks. This allows using efficient branch-and-prune algorithms to search for good JSC-VLCs. In the remainder of this section, we assume that the codeword lengths in a Kraft vector $\boldsymbol{\ell}$ are in non-decreasing order, $\ell_1 \leqslant \ell_2 \leqslant \cdots \leqslant \ell_M$.

The choice of the Kraft vector leading to a code with minimum redundancy for a given target d_{free}, or with maximum d_{free} for a given constraint on the redundancy, is not addressed in this paper. Candidate Kraft vectors satisfying necessary conditions may be generated using the bounds provided in Section III and the concept of *dominant length vectors* referenced in [18].

TABLE I
GENERIC CODEBOOK OPTIMIZATION ALGORITHM

$\mathcal{C}_{\text{opt}}$ =Optimize($\mathcal{C}_0$)

1 $\mathcal{L} = \{\mathcal{C}_0\}$; $\mathcal{S} = \emptyset$; $\underline{d}_{\text{free}} = \underline{d}_{\text{free}}(\mathcal{C}_0)$; $\overline{d}_{\text{free}} = \overline{d}_{\text{free}}(\mathcal{C}_0)$;
2 while $\mathcal{L} \neq \emptyset$
3 take $\mathcal{C}$ out of $\mathcal{L}$;
4 if $\underline{d}_{\text{free}}(\mathcal{C}) = \overline{d}_{\text{free}}$ then $\mathcal{C}_{\text{opt}} = \mathcal{C}$; end.
5 if all bits of $\mathcal{C}$ are determined,
6 $\mathcal{S} = \mathcal{S} \cup \mathcal{C}$;
7 else
8 $\left\{\mathcal{C}^{(1)}, \ldots, \mathcal{C}^{(k)}\right\}$ =Deduce($\mathcal{C}$) ;
9 insert $\mathcal{C}^{(1)}, \ldots, \mathcal{C}^{(k)}$ in $\mathcal{L}$;
10 $\overline{d}_{\text{free}} = \max_{\mathcal{C} \in \mathcal{L} \cup \mathcal{S}} \overline{d}_{\text{free}}(\mathcal{C})$;
11 $\underline{d}_{\text{free}} = \max_{\mathcal{C} \in \mathcal{L} \cup \mathcal{S}} \underline{d}_{\text{free}}(\mathcal{C})$;
12 eliminate all $\mathcal{C} \in \mathcal{L}$ such that $\overline{d}_{\text{free}}(\mathcal{C}) < \underline{d}_{\text{free}}$;
13 sort $\mathcal{L}$;
14 $\mathcal{C}_{\text{opt}} = \arg\max_{\mathcal{C} \in \mathcal{S}} d_{\text{free}}(\mathcal{C})$;end.

A. Tree of codebooks

Section III-D showed that the free distance bounds for a codebook $\mathcal{C}^1$ deduced from a partially determined codebook $\mathcal{C}^0$ are tighter. This suggests to use a *search tree* to organize the search for JSC-VLCs with large free distance. Every leaf of the search tree corresponds to a fully determined codebook, while internal nodes correspond to (partially) undetermined codebooks, from which children nodes (codebooks) may be deduced by specific rules to be described. A generic branch-and-prune algorithm to efficiently organize the search is provided in Section IV-B.

For a given Kraft vector $\boldsymbol{\ell}$, Sections IV-C and IV-D consider an undetermined codebook (such as $\mathcal{C}^0$ in Example 6) as the root of the search tree: this codebook is uniquely defined by $\boldsymbol{\ell}$. Section IV-E considers undetermined codebooks in which the prefix condition has been explicitly taken into account, as is the case in Example 7.

B. Generic branch-and-prune algorithm

The algorithm provided in Table I performs a generic branch-and-prune exploration of a code search tree to get a code $\mathcal{C}_{\text{opt}}$ with optimum distance properties starting from a fully or partially undetermined codebook $\mathcal{C}_0$.

The working list $\mathcal{L}$ is initialized with $\mathcal{C}_0$, $\mathcal{S}$ is the temporary solution list, lower and upper bounds for the free distance are evaluated using any of the methods described in Section III. At Steps 5 and 6, good fully determined codebooks are stored in the temporary solution list $\mathcal{S}$, these codebooks are such that $d_{\text{free}}(\mathcal{C}) < \overline{d}_{\text{free}}$, better codes may be found later. At Steps 8 to 10, several prefix-free codebooks deduced from $\mathcal{C}$ are built, stored in $\mathcal{L}$, and bounds for d_{free} are updated. Step 12 uses results of Corollary 3. Step 13 sorts the list of codebooks to be explored according to the free distance bound. The codebook with the largest upper bound is explored first, since it has the potential to give the largest free distance.

There are various ways to deduce codebooks from a given codebook (Step 8), leading to various organizations of the search tree, as detailed in the next sections.

C. Construction by codewords

A first method to structure the tree of codebooks introduced in Section IV-A is to instantiate all bits of one codeword at each iteration, ensuring the prefix condition. One starts at the tree root with a fully undetermined codebook for which no bit is instantiated. The child nodes, representing the deduced codebooks at Step 8 of Table I inherit the (partially) undetermined codebook $\mathcal{C}$ associated to their parent node and have one more instantiated codeword. A search tree with M levels is thus traversed. This construction is reminiscent of the A*-based algorithm proposed in [11] to design RVLCs with minimum average codeword length. In our case, the organization of the tree and the optimization metric are different.

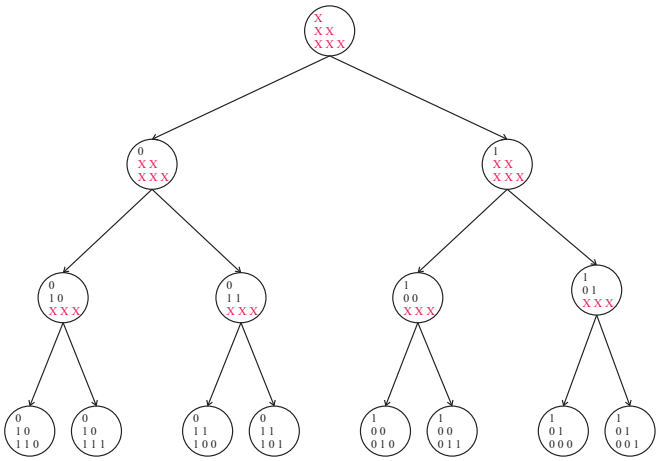

Fig. 5. Example of a search tree of JSC-VLCs for the Kraft vector $\boldsymbol{\ell} = (1, 2, 3)$ when the deduction is made by codeword

Fig. 5 shows the search tree with Kraft vector $\boldsymbol{\ell} = \{1, 2, 3\}$ when the construction is by codewords. Undetermined bits are represented by 'x'. Since $\ell_1 = 1$ is the smallest length, the children of the root (containing a fully undetermined codebook) correspond to 2^{ℓ_1} codebooks containing a single determined codeword of ℓ_1 bits. The resulting codebooks are denoted $\mathcal{C}^{(1)}$ to $\mathcal{C}^{(2^{\ell_1})}$. Then, in each codebook $\mathcal{C}^{(k)}$, $1 \leqslant k \leqslant 2^{\ell_1}$, the codeword of length $\ell_2 = 2$ is specified. The resulting codebooks are extended with codewords of length $\ell_3 = 3$. Hence, the JSC-VLCs with Kraft vector $\boldsymbol{\ell}$ correspond to the leaves of the search tree.

Inverting all bits of a codebook does not change its distance properties. Enforcing that the first codeword starts with 0 exploits this property to halve the search complexity. Moreover, if two codewords have equal lengths ℓ_i and ℓ_{i+1}, exchanging the codewords i and $i+1$ does not change the free distance of the corresponding codebook.[2] Hence one may impose a lexicographic order in the construction of the codebooks. This may again substantially reduce the time needed to find the best JSC-VLCs.

D. Construction by bitplanes

When deducing a codebook, instead of instantiating all bits of a single codeword as done in Section IV-C, one instantiates here all bits of the same bitplane: the first bit for all codewords

[2]However, codeword order does affect distance spectrum properties and may thus lead to a secondary optimization.

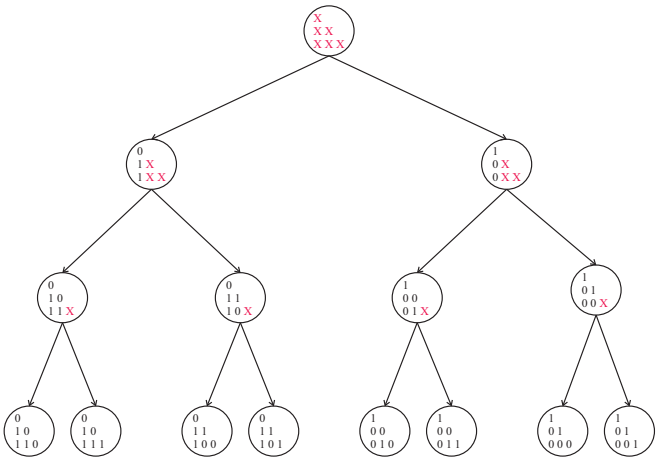

Fig. 6. Example of a search tree of JSC-VLCs for the Kraft vector $\boldsymbol{\ell} = (1, 2, 3)$ when the deduction is made by bitplane

is instantiated, then the second bit (where present), and so on. When doing this, one has to make sure that the suffixes of the codewords with a common prefix satisfy Kraft's inequality to ensure that the overall codebook remains prefix-free.

Fig. 6 shows an example search tree generation when the codebook deduction is made by bitplane. Child nodes are stemming from the root, which contains a fully undetermined codebook. The child nodes correspond to all possible combinations of the first bit for each codeword, taking care that the resulting codebooks have chances to remain prefix-free.

As in the construction of Section IV-C, the number of codebooks to consider may be reduced by exploiting the fact that inverting all bits of a codebook does not change its distance properties, and by considering a lexicographic order of the codewords of the same length.

E. Construction using canonical code trees

The structure of the unlabeled code tree already gives some information about the codes it represents, yielding free distance bounds. To efficiently exploit this fact, code trees are grouped into equivalence classes containing trees that differ only by their labeling. This is a classic tree isomorphism problem [27], [28]. The classes are represented by *canonical trees* and can be arranged on a search tree (not to be confused with the code tree). Each class can then be explored using variants of the two methods outlined in Sections IV-C and IV-D.

Equivalence is defined inductively by stating that two trees S and T are equivalent ($S \equiv T$) if they consist of a single node, or if they have the same number of immediate subtrees $S_1, S_2, \ldots, S_m$ (rooted in the direct children of S) and $T_1, T_2, \ldots, T_m$, which can be ordered such that $S_i \equiv T_i$ for all $1 \leqslant i \leqslant m$ (adapted from [28]). In the simple case at hand, two binary code trees are equivalent if there exists an isomorphism that transforms one into the other by transposing the direct children of internal nodes including the root. Assuming the tree is drawn top-down from the root, all nodes stay at their level, only their horizontal position

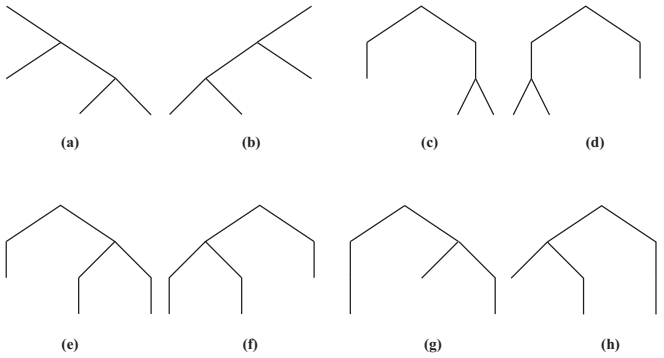

Fig. 7. Eight representations of trees associated to Kraft vector $\boldsymbol{\ell} = (2, 3, 3)$

changes. Given a Kraft vector $\boldsymbol{\ell}$, different binary trees corresponding to prefix-free codebooks may be drawn. For example, Fig. 7 provides eight tree representations for $\boldsymbol{\ell} = (2, 3, 3)$. Representations (a) and (b) show the same (equivalent) tree, while (a) and (c) are not equivalent. Trees (c) and (d) are equivalent, trees (e) and (f) are also equivalent and finally, trees (g) and (h) are equivalent. Codebooks corresponding to equivalent trees have the same prefix structure. For example, codebooks associated to representation (a) are of the form $\mathcal{C}^{(a)} = \{c_1^1 c_1^2, c_1^1 \bar{c}_1^2 c_2^3, c_1^1 \bar{c}_1^2 \bar{c}_2^3\}$, while those for representation (e) are of the form $\mathcal{C}^{(e)} = \{c_1^1 c_1^2, \bar{c}_1^1 c_2^2 c_2^3, \bar{c}_1^1 \bar{c}_2^2 c_3^3\}$.

As stated above, equivalent trees are grouped in an equivalence class, which can be represented by an appropriately defined *canonical tree*. The main problem becomes that of enumerating representations of all canonical trees associated to a given Kraft vector $\boldsymbol{\ell}$. This is a classic problem in graph isomorphism; however, to the best of our knowledge, no algorithm is directly (and efficiently) applicable to the case when a Kraft vector is given.

Let T be a binary tree, $\mathrm{left}(T)$ its left immediate subtree and $\mathrm{right}(T)$ its right immediate subtree (for notational compactness, T denotes both a tree and its root node). The function $\boldsymbol{\ell}^*(T)$ yields the Kraft vector associated to T in non-increasing order (a star * will denote vectors in non-increasing order). For example, when $\boldsymbol{\ell}^*(T) = (3, 3, 2, 1)$, T represents four codewords of length $1, 2, 3$ and 3. When $\boldsymbol{\ell}^*(T) = (0)$, T is a tree consisting of the root node only.

Define the order $\preceq$ on trees as follows: $T_1 \preceq T_2$ if and only if $\boldsymbol{\ell}^*(T_1) \preceq_{\text{lex}} \boldsymbol{\ell}^*(T_2)$, where $\preceq_{\text{lex}}$ is the lexicographic order on integer vectors. For example, $(1) \preceq_{\text{lex}} (1)$, $(3) \preceq_{\text{lex}} (3, 2, 2)$, $(0) \preceq_{\text{lex}} (1)$, and $() \preceq_{\text{lex}} (0)$, where $()$ stands for no tree, *i.e.* an empty tree.

Definition 4: A binary tree is *canonical* if $\mathrm{left}(T_i) \preceq \mathrm{right}(T_i)$ at all its internal nodes T_i.

This inductive definition guarantees that there is a unique canonical tree in each equivalence class, namely the minimal tree in the order $\preceq$. A canonical tree may be represented as a list by traversing it in any well-defined order that visits each internal node T_i once and listing $(\boldsymbol{\ell}^*(\mathrm{left}(T_i)), \boldsymbol{\ell}^*(\mathrm{right}(T_i)))$. A quite compact string representation of a tree T using the symbols '(', '0', ')' and ',' is obtained by recursively substituting $\boldsymbol{\ell}^*(T)$ with $(\boldsymbol{\ell}^*(\mathrm{left}(T)), \boldsymbol{\ell}^*(\mathrm{right}(T)))$.

The representations just outlined suggest the following

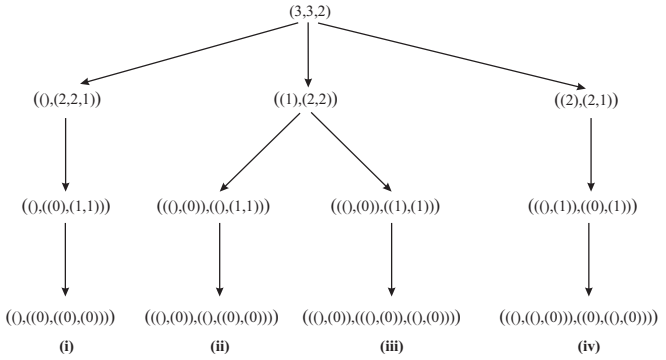

Fig. 8. Tree of canonical representations for $\boldsymbol{\ell}^* = (3, 3, 2)$

method to enumerate all canonical trees for a Kraft vector $\boldsymbol{\ell}^*$. The enumeration will take the form of a *tree of canonical representations* Λ of height $L = \max\{\ell_i^*\} = \ell_1^*$, whose leaves correspond to canonical trees, and in which going from a parent node to its children involves enumerating splittings of Kraft vectors into two parts (subtrees) ordered by $\preceq$. Starting with the Kraft vector $\boldsymbol{\ell}^*$ at the root of Λ, split $\boldsymbol{\ell}^* - \mathbf{1}$ (subtracting 1 from each length, to go down one level in the tree) into two parts $\boldsymbol{\ell}_1^*$ and $\boldsymbol{\ell}_2^*$, such that $\boldsymbol{\ell}_1^* \preceq_{\text{lex}} \boldsymbol{\ell}_2^*$ and $K(\boldsymbol{\ell}_i^*) \leqslant 1$ $(i = 1, 2)$. Note that since $\boldsymbol{\ell}_1^*$ may be empty (corresponding to an absent leaf, *i.e.* a -1 entry in $\boldsymbol{\ell}^* - \mathbf{1}$) one needs to define $K(()) = 1$. The first level below the root of Λ contains all splittings of $\boldsymbol{\ell}^*$, the second level contains pairs of splittings of $(\boldsymbol{\ell}_1^*, \boldsymbol{\ell}_2^*)$, and so on; the recursive splitting stops with leafs '(0)' and empty nodes '$()$'.

Example 8: Consider the Kraft vector $\boldsymbol{\ell} = (2, 3, 3)$. Fig. 8 shows a part of the tree of canonical representations for $\boldsymbol{\ell}^* = (3, 3, 2)$. Starting from the root node $\boldsymbol{\ell}^*$, one has $\boldsymbol{\ell}^* - \mathbf{1} = (2, 2, 1)$. There are three choices for $(\boldsymbol{\ell}_1^*,\ \boldsymbol{\ell}_2^*)$ at the first level: $(\boldsymbol{\ell}_1^* = (),\ \boldsymbol{\ell}_2^* = (2, 2, 1))$, $(\boldsymbol{\ell}_1^* = (1),\ \boldsymbol{\ell}_2^* = (2, 2))$ and $(\boldsymbol{\ell}_1^* = (2),\ \boldsymbol{\ell}_2^* = (2, 1))$. Each choice leads to one or more possible canonical trees. At the second level, there are *e.g.* two ways to split the pair $((1),\ (2, 2))$. This process is repeated for all internal nodes. This leads to leaf (i) in Fig. 8, which is the canonical tree representation of Fig. 7 (a). Leaves (ii), (iii) and (iv) in Fig. 8 correspond to Fig. 7 (c), (e) and (g), respectively.

As illustrated by Example 8, the internal nodes of the tree of canonical representations correspond to *partially determined canonical representations*. For example, the representation $((1), (2, 2))$ in the middle branch of Fig. 8 indicates that the second and third codewords have a common prefix, leading to codebooks of the form $\mathcal{C}^1 = \left\{c_1^1 c_1^2, \bar{c}_1^1 c_2^2 c_2^3, \bar{c}_1^1 c_3^2 c_3^3\right\}$, while the representation $(((), (0)), ((), ((0), (0))))$ leads to codebooks of the form $\mathcal{C}^2 = \left\{c_1^1 c_1^2, \bar{c}_1^1 c_2^2 c_2^3, \bar{c}_1^1 c_2^2 \bar{c}_2^3\right\}$. By extending the notion of codebook deduction (Definition 3) to inverted symbolic labels, one sees that $\mathcal{C}^1 \prec \mathcal{C}^2$. Thus free distance bounds for $\mathcal{C}^1$ will hold for $\mathcal{C}^2$ and all codes deduced from it. In fact, there is a one-to-one mapping between a partially determined canonical representation T^i and the corresponding codebook $\mathcal{C}^i$ with symbolic labels.[3] Thus, if tree T^1 is a child of tree T^0 in Λ, then by Corollary 3 the free distance bounds from Section III-E satisfy

$$[\underline{d}_{\text{free}}^{\text{T}}(T^1), \overline{d}_{\text{free}}^{\text{T}}(T^1)] \subseteq [\underline{d}_{\text{free}}^{\text{T}}(T^0), \overline{d}_{\text{free}}^{\text{T}}(T^0)]. \qquad (12)$$

The upper bound can sometimes be improved as follows. A partially determined canonical tree T^i is represented by recursive splittings of Kraft vectors. Each such vector corresponds to a subtree S (subcodebook) with a common prefix. Thus the extension of Plotkin's bound of Section III-B may be applied to yield $\overline{d}_{\text{free}}^{\text{Te}}(T^i) = \min\left(\overline{d}_{\text{free}}^{\text{T}}(T^i), \min_{S \subset T^i} \overline{d}_{\text{free}}^{\text{pe}}(\boldsymbol{\ell}(S))\right)$.

For a given Kraft vector $\boldsymbol{\ell}$, we explore the tree of canonical representations (Fig. 8) with a branch-and-prune algorithm. Whenever a fully determined canonical tree (a leaf) is reached, all possible labelings of that canonical tree are explored in branch-and-prune fashion using variations of the methods in Sections IV-C and IV-D. The number of labelings to be examined may be further reduced by imposing fixed labels for each internal node of the canonical tree that has children satisfying $\boldsymbol{\ell}_1^* = \boldsymbol{\ell}_2^*$, since the further splittings of $\boldsymbol{\ell}_1^*$ and $\boldsymbol{\ell}_2^*$ will contain transpose-symmetric solutions, of which only one needs to be tested (*e.g.* $\boldsymbol{\ell}_1^*$ is split into $(\boldsymbol{\ell}_{1,1}^*, \boldsymbol{\ell}_{1,2}^*)$, while $\boldsymbol{\ell}_2^*$ is split into $(\boldsymbol{\ell}_{2,1}^*, \boldsymbol{\ell}_{2,2}^*)$; then $(\boldsymbol{\ell}_{2,1}^*, \boldsymbol{\ell}_{2,2}^*)$ is also a valid split of $\boldsymbol{\ell}_1^*$, as is $(\boldsymbol{\ell}_{1,1}^*, \boldsymbol{\ell}_{1,2}^*)$ for $\boldsymbol{\ell}_2^*$).

V. Experimental results

This section presents three sets of experiments. First, the various branch-and-prune algorithms presented in Section IV are compared to an exhaustive search for short codes. Then, the evolution of the search complexity as a function of the size of the Kraft vectors is briefly considered. Finally, the design of JSC-VLCs for relatively large source alphabets (with a maximum of 26 symbols) is considered. Experiments were performed on a single processor of an Intel Xeon E5420 at 2.50 GHz with 64 GB memory.

A. Branch-and-prune versus exhaustive search

The first Kraft vector considered is $\boldsymbol{\ell} = (4, 5, 6, 7)$. Table II shows the time needed for the methods detailed in Section IV to find a JSC-VLC with largest free distance. The generic optimization algorithm is used with codebook expansion by bitplane (*Bp*, see Section IV-D) and by codeword (*Cw*, see Section IV-C). Canonical tree representations are considered next with codebook expansion by bitplane (*C-Bp*) and by codeword (*C-Cw*), see Section IV-E. *#Codebooks* is the number of processed complete and incomplete JSC-VLCs. *#Trees* and *#Expl. trees* are respectively the number of tree representations considered and explored. The best JSC-VLCs obtained by each method is also represented.

Table II shows that all branch-and-prune algorithms are much more efficient than exhaustive search in terms of computing time (the best method is more than 600 times faster)

[3] A partially determined canonical representation T^i may be seen as a binary code tree containing nontrivial Kraft vectors in some leaves (while leaves with vector '(0)' correspond to codewords). The Kraft vectors are mapped to symbolic codeword labels for the corresponding subtree, like for an undetermined codebook.

TABLE II
COMPARISON BETWEEN EXHAUSTIVE SEARCH AND THE BRANCH-AND-PRUNE ALGORITHM WITH KRAFT VECTOR $\boldsymbol{\ell} = (4, 5, 6, 7)$ (EXTENSION OF HELLER'S BOUND YIELDS $\bar{d}_{\text{free}}^{\text{he}} = 6$ AT $n = 10$)

Method	**Exhaustive**	**Bp**	**Cw**	**C-Bp**	**C-Cw**
#Codebooks	1,586,880	83,400	5,862	28,566	12,144
d_{free}	6	6	6	6	6
Time (s)	234	37	0.37	5	1
#Trees	–	–	–	970	970
#Expl. trees	–	–	–	108	108
4	0111	0001	0000	0111	0000
5	11001	11110	01111	11000	11110
6	000000	101011	110101	101011	101011
7	1001011	0100100	1001101	1001100	1001101

and number of examined JSC-VLCs (more than 250 times less for the best method). Five different best JSC-VLC codebooks have been obtained, all with $d_{\text{free}} = 6$.

When considering the number of intermediate JSC-VLCs that need to be examined, the expansion *by codeword* is much more efficient, alone or performed on canonical tree representations. With this expansion method, more accurate bounds for the free distance are obtained at earlier stages of the search tree traversal, which helps pruning more efficiently.

Other experiments showed that C-Cw becomes more efficient than Cw alone for larger codebooks, especially codebooks with many codewords of the same length. Consider for instance the Kraft vector $\boldsymbol{\ell} = (4, 5, 5, 6, 6, 7, 7)$. C-Cw examines $682,322$ JSC-VLCs in 746 s, while Cw considers $1,874,127$ JSC-VLCs in $3,152$ s to obtain codebooks with the same free distance. The remaining experiments are thus performed with C-Cw.

Table III shows the evolution of the complexity of the C-Cw algorithm when the alphabet size increases, where new symbols have codewords that are at least as long as the previously longest codeword. It also compares the free distance of the best code obtained using the SAT-based approach presented in [19]. The proposed approach has a much larger computational complexity. Nevertheless, since we are considering the exact free distance (and not a bound) as the design criterion, better codes may be obtained, as in the present example.

B. Design of JSC-VLCs for larger alphabets

For designing JSC-VLCs for sources with larger alphabets (typically more than 10 symbols), we introduce a heuristic to reduce the number of canonical tree representations to explore, hence the number of codebooks to examine. The number of trees can be reduced by considering only canonical trees which maximize the upper bound on the free distance in (12), *i.e.*, trees in which equal-length codewords have a common prefix as short as possible. A reasonable heuristic to ensure this, when splitting Kraft vectors during the construction of canonical trees, is to make sure that the two new vectors have the same number of equal-length codewords (or their numbers differ only by one). This choice can be justified by Plotkin's bound (2), which shows that the free distance upper bound is decreasing in the number of equal-length codewords.

The first experiment in Table IV considers the 26 symbols of the English alphabet, $X_{(26)}$, with $\boldsymbol{\ell} = (2@5, 4@7, 8@8, 12@10)$, $m_i@\ell_i$ indicating as in [14] that there are m_i codewords of length ℓ_i. C-Cw finds a maximum free distance of 4 and the average code rate is $\ell_{\text{av}}^J = 7.3375$ bits/symb. The entropy is $H(X_{(26)}) = 4.1752$ bits/symb. An equivalent tandem scheme using a single-letter Huffman VLC followed by a rate 1/2 convolutional code (CC) with constraint length 2 leads to an average code length $\ell_{\text{av}}^T = 8.4090$ bits/symb with $d_{\text{free}} = 4$ (the Huffman code has an average length 4.2045 bits/symb). Thus, for the same error correcting capability, the joint scheme yields a gain of code-rate of $\ell_{\text{av}}^T - \ell_{\text{av}}^J = 1.0715$ bits/symb compared to the considered tandem scheme.

The second experiment in Table IV is made for the 16 most probable symbols of the English alphabet, $X_{(16)}$. The Kraft vector is now $\boldsymbol{\ell} = (2@6, 2@7, 4@8, 4@9, 4@10)$, leading to $d_{\text{free}} = 5$ and to $\ell_{\text{av}}^J = 7.750$ bits/symb. The entropy $H(X_{(16)}) = 3.821$ bits/symb. The equivalent tandem scheme using a single-letter Huffman VLC followed by a rate $1/2$ CC with constraint length $L_c = 3$ has $d_{\text{free}} = 5$ and an overall average code length $\ell_{\text{av}}^T = 7.705$ bits/symb. For the same free distance, a small loss in coding efficiency of $\ell_{\text{av}}^J - \ell_{\text{av}}^T = 0.045$ bits/symb is obtained.

In all cases, the bound $\bar{d}_{\text{free}}^{\text{he}}$ coincides with the optimal free distance, showing its efficiency for larger alphabets.

The search complexity appears to be dominated by the number of d_{free} (bound) evaluations using Dijkstra's algorithm, whose complexity with the current implementation is $O(|\mathcal{S}_b|^4)$. The computing time for $X_{(26)}$ is smaller than that for $X_{(16)}$ due to the fact that for $X_{(26)}$, a single tree was explored to obtain a code with free distance matching the extension of Heller's upper bound, whereas four trees were explored for $X_{(16)}$, needing many more distance evaluations.

For a JSC-VLC with Kraft vector $\boldsymbol{\ell}^J = \left(\ell_1^J, \ldots, \ell_M^J\right)$, the complexity can be evaluated as the number of states in B-FSE at the decoder side, $S_b^J = \sum_{i=1}^M \ell_i^J - M + 1$. For an equivalent tandem scheme composed of a Huffman VLC with $\boldsymbol{\ell}^T = \left(\ell_1^T, \ldots, \ell_M^T\right)$ followed by a rate $1/n$ CC with constraint length L_c, the complexity of a joint decoder on the product graph $\Gamma_b^{\text{VLC}} \times \Gamma_b^{\text{CC}}$ can be considered, which has $S_b^T = S_b^{\text{VLC}} \cdot S^{\text{CC}}$ states, where $S_b^{\text{VLC}} = \sum_{i=1}^M \ell_i^T - M + 1$ and $S^{\text{CC}} = 2^{L_c - 1}$. For the example source $X_{(26)}$ considered above, $S_b^J = 197$ and $S_b^T = 468$, while for source $X_{(16)}$, $S_b^J = 119$ and $S_b^T = 104$. Alternatively, one may consider separate decoders, and put all complexity into the CC. The performance of the CC decoder would be improved, however, this would be at the price of a worse error propagation behavior after Huffman decoding than when considering joint decoding.

VI. CONCLUSION

This paper introduces several new bounds for the free distance of JSC-VLC codes. These bounds involve an extension of Heller's bound to JSC-VLC codes, and several bounds that can be obtained using the pairwise distance graph introduced in Section II, considering that not all bits are specified in a given codebook. For the latter bounds, knowing the structure of the code tree associated to the codebook may significantly improve the tightness of the bound.

A second contribution consists in the proposition of a branch-and-prune algorithm to optimize the free distance of

TABLE III
FREE DISTANCE AND SEARCH COMPLEXITY OF THE BEST CODE OBTAINED USING [19], AND USING C-CW, AS A FUNCTION OF THE KRAFT VECTOR

			Using [19]		Proposed approach				
$\boldsymbol{\ell}$	$d_{\text{free}}^{\text{pe}}$	$d_{\text{free}}^{\text{he}}$	d_{free}	Time (s)	d_{free}	Time (s)	#Codebooks	#Expl. trees	#Trees
$(3,7,8,9)$	∞	8	6	0.03	7	5	15,228	23	50
$(3,7,8,9,11)$	∞	7	5	0.12	7	56	45,672	26	209
$(3,7,8,9,11,12)$	∞	7	5	0.4	7	478	140,204	83	1,595
$(3,7,8,9,11,12,13)$	∞	7	5	0.4	7	857	76,416	32	6,578
$(3,7,8,9,11,12,13,14)$	∞	7	4	97	7	12,553	566,660	209	37,953
$(3,7,8,9,11,12,13,14,15)$	∞	7	4	339	7	131,302	1,039,980	419	501,782

TABLE IV
DESIGN OF JSC-VLCS FOR THE ENGLISH ALPHABET USING C-CW AND THE PROPOSED HEURISTIC FOR CANONICAL TREE REPRESENTATIONS; PROBABILITIES TAKEN FROM [14]

Symbols	Probabilities	$\boldsymbol{\ell}$	Codeword	$\boldsymbol{\ell}$	Codeword
$a_1 = E$	$p_{a_1} = 0.1270$	5	00000	6	010110
$a_2 = T$	$p_{a_2} = 0.0906$	5	11110	6	101001
$a_3 = A$	$p_{a_3} = 0.0817$	7	0110000	7	0110101
$a_4 = O$	$p_{a_4} = 0.0751$	7	0011101	7	1001010
$a_5 = I$	$p_{a_5} = 0.0697$	7	1001000	8	00001100
$a_6 = N$	$p_{a_6} = 0.0674$	7	1100110	8	01100111
$a_7 = S$	$p_{a_7} = 0.0633$	8	01010101	8	11110000
$a_8 = H$	$p_{a_8} = 0.0609$	8	01110010	8	10011011
$a_9 = R$	$p_{a_9} = 0.0599$	8	00011000	9	001111011
$a_{10} = D$	$p_{a_{10}} = 0.0425$	8	00101110	9	011101100
$a_{11} = L$	$p_{a_{11}} = 0.0403$	8	10100101	9	110000111
$a_{12} = C$	$p_{a_{12}} = 0.0278$	8	10000010	9	100010000
$a_{13} = U$	$p_{a_{13}} = 0.0276$	8	11101000	10	0010000000
$a_{14} = M$	$p_{a_{14}} = 0.0241$	8	11011011	10	0111110011
$a_{15} = W$	$p_{a_{15}} = 0.0236$	10	0100101100	10	1101111100
$a_{16} = F$	$p_{a_{16}} = 0.0223$	10	0101110011	10	1000001011
$a_{17} = G$	$p_{a_{17}} = 0.0202$	10	0110110101		
$a_{18} = Y$	$p_{a_{18}} = 0.0197$	10	0111111000		
$a_{19} = P$	$p_{a_{19}} = 0.0193$	10	0000101011		
$a_{20} = B$	$p_{a_{20}} = 0.0149$	10	0001001101		
$a_{21} = V$	$p_{a_{21}} = 0.0098$	10	0011010100		
$a_{22} = K$	$p_{a_{22}} = 0.0077$	10	0010010011		
$a_{23} = J$	$p_{a_{23}} = 0.0015$	10	1011110010		
$a_{24} = X$	$p_{a_{24}} = 0.0015$	10	1010101101		
$a_{25} = Q$	$p_{a_{25}} = 0.0010$	10	1001110101		
$a_{26} = Z$	$p_{a_{26}} = 0.0007$	10	1000111000		
		$\ell_{\text{av}} = 7.3375$	$d_{\text{free}} = 4$	$\ell_{\text{av}} = 7.750$	$d_{\text{free}} = 5$
d_{free} bounds			$\overline{d}_{\text{free}}^{\text{he}} = 4$		$\overline{d}_{\text{free}}^{\text{he}} = 5$
Computing time			310 h		554 h
Trees generated			3,130		43,172
Trees explored			1		4
d_{free} evaluations			384,336		3,121,150

a JSC-VLC with given codeword lengths. The resulting codebook maximizes the free distance, and not a lower bound on it, as was the case for other codebook optimization techniques. Among the proposed algorithms, the most efficient relies on the exploration of canonical tree representations, which describe the structure of equivalent prefix-free codebooks. For a given canonical tree, the best codebook is then searched. Compared to state-of-the-art search techniques, better codebooks in terms of free distance may be obtained, for certain parameter choices.

This algorithm considers two types of search trees: one for the canonical trees and, for each canonical representation, a search tree for the best codebook. A combined exploration of both trees is likely to significantly improve the search efficiency. Implementing Dijkstra's algorithms using Fibonacci heaps would reduce the complexity of evaluating the free distance from $O(N^2)$ to $O(M\log(N))$, where N and M are the number of vertices and edges of the PDG, respectively. Better heuristics to select equal-length codewords with good minimal distance properties such as those introduced in [14], [16] could also be considered. Finally, one may try to adapt the search tree using the A* algorithm proposed in [11] and introduce constraints on the free distance of the code to be designed.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their helpful remarks.

REFERENCES

[1] A. Hedayat and A. Nosratinia, "Performance analysis and design criteria for finite-alphabet source-channel codes," *IEEE Trans. Commun.*, vol. 52, no. 11, pp. 1872– 1879, 2004.

[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. S. Fraenkel and S. T. Klein, "Bidirectional Huffman coding," *The Computer Journal*, vol. 33, no. 4, pp. 296–307, 1990.
[5] Y. Takishima, M. Wada, and H. Murakami, "Reversible variable length codes," *IEEE Trans. Commun.*, vol. 43, no. 2/3/4, pp. 158–162, 1995.
[6] C.-W. Tsai and J.-L. Wu, "On-constructing the Huffman-codes based reversible variable-length codes," *IEEE Trans. Commun.*, vol. 49, no. 9, pp. 1506–1509, 2001.
[7] K. Lakovic and J. Villasenor, "An algorithm for construction of efficient fix-free codes," *IEEE Commun. Lett.*, vol. 7, no. 8, pp. 391– 393, 2003.
[8] H.-W. Tseng and C.-C. Chang, "Construction of symmetrical reversible variable length codes using backtracking," *The Computer Journal*, vol. 46, no. 1, pp. 100–105, 2003.
[9] J. Wang, L.-L. Yang, and L. Hanzo, "Iterative construction of reversible variable-length codes and variable-length error-correcting codes," *IEEE Commun. Lett.*, vol. 8, no. 11, pp. 671– 673, 2004.
[10] C.-W. Lin, J.-L. Wu, and Y.-J. Chuang, "Two algorithms for constructing efficient Huffman-code based reversible variable length codes," *IEEE Trans. Commun.*, vol. 56, no. 1, pp. 81–89, 2008.
[11] Y.-M. Huang, T.-Y. Wu, and Y. Han, "An A*-based algorithm for constructing reversible variable length codes with minimum average codeword length," *IEEE Trans. Commun.*, vol. 58, no. 11, pp. 3175 –3185, Nov. 2010.
[12] K. Lakovic and J. Villasenor, "On design of error-correcting reversible variable length codes," *IEEE Commun. Lett.*, vol. 6, no. 8, pp. 337–339, 2002.
[13] R. Thobaben and J. Kliewer, "An efficient variable-length code construction for iterative source-channel decoding," *IEEE Trans. Commun.*, vol. 57, no. 7, pp. 2005–2013, Jul. 2009.
[14] V. Buttigieg, "Variable-length error correcting codes," PhD dissertation, University of Manchester, Manchester, U.K., 1995.
[15] V. Buttigieg and P. Farrell, "Variable-length error-correcting codes," *IEE Proceedings on Communications*, vol. 147, no. 4, pp. 211–215, Aug. 2000.
[16] C. Lamy and J. Paccaut, "Optimised constructions for variable-length error correcting codes," in *Proc. Information Theory Workshop*, 2003, pp. 183–186.
[17] R. Maunder and L. Hanzo, "Genetic algorithm aided design of component codes for irregular variable length coding," *IEEE Trans. Commun.*, vol. 57, no. 5, pp. 1290 –1297, May 2009.
[18] S. Savari, "On optimal reversible-variable-length codes," in *Proc. Information Theory and Applications Workshop*, Feb. 2009, pp. 311–317.
[19] N. Abedini, S. P. Khatri, and S. A. Savari, "A SAT-based scheme to determine optimal fix-free codes," in *Proc. Data Compression Conference*, 2010, pp. 169–178.
[20] V. Buttigieg and P. Farrell, "On variable-length error-correcting codes," in *Proc. IEEE ISIT*, 1994, p. 507.
[21] C. Weidmann and M. Kieffer, "Evaluation of the distance spectrum of variable-length finite-state codes," *IEEE Trans. Commun.*, vol. 58, no. 3, pp. 724–728, 2010.
[22] A. Diallo, C. Weidmann, and M. Kieffer, "Efficient computation and optimization of the free distance of variable-length finite-state joint source-channel codes," *IEEE Trans. Commun.*, vol. 59, no. 4, pp. 1043 – 1052, 2011.
[23] ——, "Optimizing the search of finite-state joint source-channel codes based on arithmetic coding," in *Proc. European Signal Processing Conference*, 2009.
[24] M. Gondran and M. Minoux, *Graphs and algorithms*. Chichester, UK: Wiley, 1984.
[25] R. G. Gallager, *Information theory and reliable communication*. New York: Wiley, 1968.
[26] R. G. Johannesson and K. S. Zigangirov, *Fundamentals of convolutional coding*. IEEE Press, 1999.
[27] A. W. Aho, J. E. Hopcroft, and J. D. Ullman, *The design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
[28] S. R. Buss, "Alogtime algorithms for tree isomorphism, comparison, and canonization," in *Proc. of the 5th Kurt Gödel Colloquium on Computational Logic and Proof Theory*. London, UK: Springer-Verlag, 1997, pp. 18–33.