# Almost All Complete Binary Prefix Codes Have a Self-Synchronizing String

Christopher F. Freiling, Douglas S. Jungreis,
François Théberge, *Member, IEEE*, and Kenneth Zeger, *Fellow, IEEE*

***Abstract*—The probability that a complete binary prefix code has a self-synchronizing string approaches one, as the number of codewords tends to infinity.**

***Index Terms*—Channel noise, source coding, synchronization, variable length codes.**

## I. Introduction

Variable-length binary codes have been frequently used for communications since Huffman's important paper on constructing minimum average length codes [13]. One drawback of variable-length codes is the potential loss of synchronization in the presence of channel errors. However, many variable-length codes seem to possess a "self-synchronization" property that lets them recover from bit errors.

In particular, for some variable-length codes there exists a certain binary string (not necessarily a codeword) which automatically resynchronizes the code. That is, if a transmitted sequence of bits is corrupted by one or more bit errors, then as soon as the receiver by random chance correctly detects a self-synchronizing string, the receiver can continue properly parsing the bit sequence into codewords. Most commonly used binary prefix codes, including Huffman codes, are "complete," in the sense that the vertices in their decoding trees are either leaves or have two children. An open question has been to characterize which prefix codes and which complete prefix codes have a self-synchronizing string. In this correspondence, we prove that almost all complete prefix codes have a self-synchronizing string.

When variable-length codes are used for transmission, we say that *synchronization* is achieved when the receiver can determine the first symbol of some codeword with certainty. The *synchronization delay* is the number of code symbols which must be observed by the receiver before synchronization is achieved. The synchronization delay is a random variable depending on the source statistics and the code. A binary code is *statistically synchronizable* if the synchronization delay is finite with probability one. A string is self-synchronizing if by observing it, the receiver can achieve synchronization.[1] Some codes have self-synchronizing strings and some codes do not. Clearly, if a code has a self-synchronizing string then the code is statistically synchronizable, since the probability of not observing the string tends to zero as the number of observations increases without bound.

Gilbert [10] studied certain variable-length synchronizable codes where all codewords had common prefixes, and variable-length codes were studied in [11]. Wei and Sholtz [29] showed that fixed-length codes are statistically synchronizable if and only if they have a self-synchronizing string, as well as some other characterizations. Capocelli, Gargano, and Vaccaro [2] proved that a variable-length code is statistically synchronizable if and only if it has a self-synchronizing string. They also gave an algorithm that determines whether a code has a self-synchronizing string. It is known that if the all-zeros and all-ones words of a complete prefix code have relatively prime lengths, then the code has a self-synchronizing string [1]. Even [7] gave an algorithm for determining whether finite automata are synchronizable and showed how the algorithm can be applied to variable-length codes. Capocelli *et al.* [3] gave an algorithm for constructing prefix codes with self-synchronizing strings such that the average length of the code is close to optimal. They also provided a method for constructing prefix codes with a self-synchronizing codeword and whose rate redundancy is low.

Ferguson and Rabinowitz [8] found sufficient conditions for the existence or nonexistence of self-synchronizing strings for Huffman codes for many classes of source probabilities, but required the synchronizing strings to be Huffman codewords. They also examined the problem of finding, for a given set of codeword lengths, a Huffman code with the shortest possible self-synchronizing codeword. Escott and Perkins [6] gave an algorithm for finding such codes suggested in [8], but only if a shortest self-synchronizing codeword exists whose length is one bit longer than the shortest Huffman codeword length. Rudner [25] showed how to generate a Huffman code with a shortest synchronizing sequence for certain cases when the minimum codeword length is less than five. Montgomery and Abrahams [20] showed how to construct variable-length codes with a self-synchronizing string, whose average length is close to that of a Huffman code.

Other work in these areas can be found in [11], [12], [15]–[17], [19], [21]–[23], [26]–[28], [30].

## II. Complete Prefix Codes

A *prefix code* is a set $\mathcal{C} \subset \{0, 1\}^*$ with the property that no element of $\mathcal{C}$ is a prefix of any other element of $\mathcal{C}$. The elements of $\mathcal{C}$ are called *codewords*. A prefix code is *complete*[2] if for every $u \in \{0, 1\}^*$, the string $u0$ is a prefix of some codeword if and only if $u1$ is a prefix of some codeword. Huffman codes are examples of complete prefix codes. Given a finite set $\mathcal{A}$ with the same cardinality as $\mathcal{C}$, an *encoder* is any one-to-one mapping $f\colon \mathcal{A}^* \to \mathcal{C}^*$ with the properties that $f$ maps the empty string to the empty string, $f(\mathcal{A}) \subset \mathcal{C}$, and $f(uv) = f(u)f(v)$ for all $u, v \in \mathcal{A}^*$. Thus, a prefix code converts a sequence of symbols from $\mathcal{A}$ into a sequence of bits by replacing each symbol $s \in \mathcal{A}$ by the codeword $f(s)$.

Each encoder has an inverse, called a *decoder*. Prefix codes are attractive for communications because they have easily implementable encoders and decoders. Encoding is a table lookup operation. Decoding (i.e., parsing) a binary string involves scanning from left to right and inserting a comma each time a codeword is seen.

Manuscript received February 6, 2002; revised May 3, 2003. This work was supported by the Institute for Defense Analyses and the National Science Foundation. The material in this correspondence was presented at the IEEE International Symposium on Information Theory, Yokohama, Japan, June 2003.

C. F. Freiling is with the Department of Mathematics, California State University, San Bernardino, CA 92407-2397 USA (e-mail: cfreilin@csusb.edu).

D. S. Jungreis is with the Center for Communications Research, San Diego, CA 92121-1969 USA (e-mail: jungreis@ccrwest.org).

F. Théberge is with the Communications Security Establishment, CSE/DND, Terminal, Ottawa, ON K1G 3Z4, Canada (e-mail: theberge@ieee.org).

K. Zeger is with the Department of Electrical and Computer Engineering, University of California, San Diego, La Jolla, CA 92093-0407 USA (e-mail: zeger@ucsd.edu).

Communicated by M. Weinberger, Associate Editor for Source Coding.

Digital Object Identifier 10.1109/TIT.2003.815803

[1]In the literature, such a string has also been called a "synchronizing sequence" and a "universal synchronizing sequence." A code containing such a string has been called "completely self-synchronizing," "ergodic," and "synchronizing."

[2]There is some variation in terminology in the literature for this concept. For example, Gallager [9, p. 54] and Cover and Thomas [4, p. 111] use "complete," whereas Knuth [14, p. 713] uses "extended," McEliece [18, p. 249] uses "full," Csiszár and Körner [5, p. 72] use "saturated," and Gilbert and Moore [11, p. 942] use "exhaustive," and Berstel and Perrin [1, p. 98] use "maximal." In fact, there is quite a bit of closely related literature on synchronization in the fields of automata theory and finitely presented monoids (e.g., see [1]).

A *binary tree* is a finite directed acyclic graph such that every node has out-degree zero or two, one node (called the *root node* and denoted $r$) has in-degree zero, and all other nodes have in-degree one. Whenever an edge leads from one node to another, these nodes are referred to, respectively, as the *parent* and *child*. The edges leading from a parent to its children are labeled "0" and "1," and the corresponding children are called the 0-*child* and 1-*child*. A *leaf* is a node with no children. A nonleaf node is called *internal*. A *branch* is a path from the root to a leaf, and each branch is identified with the sequence of zeros and ones that label this path. In particular, the *zero branch* is the branch associated with the all-zeros codeword. Each node of a tree is identified with the label of the path to the node from the root. In particular, the root node is identified with the empty string.

A complete prefix code can be conveniently represented by the binary tree whose branches are its codewords. To decode a binary sequence, one places a pointer at the root, and proceeds through the bit sequence. For each 0-bit, the pointer is moved to its 0-child, and for each 1-bit the pointer is moved to its 1-child. Whenever the pointer reaches a leaf, the symbol in $\mathcal{A}$ that is represented by that leaf's path is output, and the pointer is reset to the root.

A simple but useful fact is that for every node $v$ in a binary tree, there exists a nonnegative integer $k$ such that $v0^k$ is the root node, where $0^k$ is the string of $k$ zeros. This is because we can traverse down a binary tree from $v$ along 0-children, eventually hitting a leaf node.

## III. Self-Synchronizing Strings

Recall that in the decoding procedure, we keep a pointer to a node in the tree; for each bit of data, we move the pointer from a node to one of its children, and when we reach a leaf, we move the pointer back to the root. In this way, any string of data-bits moves the pointer from one node to another. We define this formally.

*Definition III.1:* Let $z$ denote the $n$-long string $b_1, \ldots, b_n$ where each $b_i \in \{0, 1\}$. For internal nodes $u$ and $v$ in a binary tree, we say that $z$ *brings* $u$ *to* $v$ if either:

- $n = 1$, $v$ is the $b_1$-child of $u$;
- $n = 1$, $v$ is the root, and the $b_1$-child of $u$ is a leaf;
- $n > 1$, $b_1$ brings $u$ to a node $w$, and $b_2, \ldots, b_n$ brings $w$ to $v$.

If $w$ is a leaf node then we say $z$ *brings* $u$ *to the root via* $w$ if $b_1, \ldots, b_{n-1}$ brings $u$ to a node whose $b_n$-child is $w$. In this circumstance, when there is no ambiguity, we may for convenience say that $z$ brings $u$ to $w$.

It is clear that for any string $z$ and any internal node $u$ there is exactly one internal node $v$ such that $z$ brings $u$ to $v$. A string is a concatenation of codewords if and only if it brings the root to itself.

In the decoding procedure, the location of the pointer depends on the entire set of bits that have been decoded; however, knowing the most recently decoded bits provides some information about the location of the pointer. In particular, there are certain strings that bring multiple nodes to one node. We say a string $z$ *coalesces* $u$ *and* $v$ if it brings $u$ and $v$ to the same node. We say two nodes $u$ and $v$ *coalesce* if there exists some string that coalesces them.

A string $z \in \{0, 1\}^*$ is *self-synchronizing* for a complete prefix code if it brings every internal node to the root. Therefore, if we are decoding a data stream and we have just decoded a self-synchronizing string, then the pointer is back at the root, regardless of what bits preceded the self-synchronizing string.

*Example III.2:* The complete prefix code represented by the binary tree in Fig. 1 has "101" as a self-synchronizing string. To see this, observe that the internal nodes are $0^i$ for $0 \leq i \leq 5$ and $10^i$ for $0 \leq i \leq 2$. The string "1" brings all but one of these to the root, the exception being that "1" brings the root to 1. The string "01" brings both 1 and the root to the root, so "101" brings every internal node to the root.

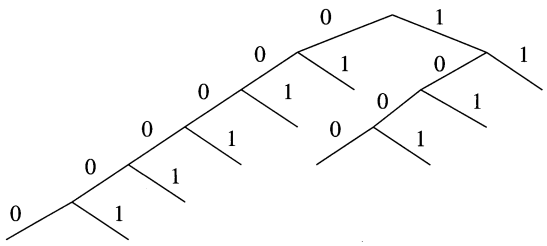

Fig. 1. Binary tree example of complete prefix code with a self-synchronizing string.

*Example III.3:* The complete prefix codes represented by the binary trees in Fig. 2 do not have self-synchronizing strings. This is because every path from the root to itself has even length, whereas each tree contains nodes which can only get to leaf nodes by paths of odd length.

Let $\mathcal{Q}$ be a collection of binary codes and let $P$ be some property that each such code may or may not have. For each positive integer $n$ let $\alpha(n)$ be the number of codes in $\mathcal{Q}$ with $n$ codewords that have property $P$, and let $\beta(n)$ be the number of codes in $\mathcal{Q}$ with $n$ codewords. We say that *almost all codes in* $\mathcal{Q}$ *have property* $P$ if $\lim_{n\to\infty} \alpha(n)/\beta(n) = 1$. The main result of this correspondence is the following theorem.

*Theorem III.4:* Almost all complete prefix codes have a self-synchronizing string.

The following general lemma about self-synchronizing strings will be useful in later sections.

*Lemma III.5:* If every internal node on the zero branch of a binary tree coelesces with the root, then the tree has a self-synchronizing string.

*Proof:* For any string $x$, let $N_x$ denote the set of nodes $v$ such that $x$ brings some node to $v$. Select $x$ so that $|N_x|$ is minimal. If $|N_x| = 1$, then there is a self-synchronizing string. Assume then that $|N_x| \geq 2$. Let $v$ and $w$ be any two nodes of $N_x$. For sufficiently large $m'$, the string $0^{m'}$ will bring both $v$ and $w$ to nodes on the zero-branch. Therefore, some string $0^m$ will bring $v$ to the root, while bringing $w$ to a node $0^i$. By assumption, there is a string $y$ that coalesces $0^i$ with the root. Hence, we have a string $z = x0^m y$ such that $|N_z| < |N_x|$, which is a contradiction. □

To exploit this lemma, it is useful to view a binary tree via the $k$-forest consisting of the union of trees attached to (and including) the 1-child of each of the $k$ internal nodes of the zero branch.

## IV. Tree Counting

A $k$-forest is an ordered collection of $k$ disjoint binary trees. In this section, some results on $k$-forests are presented that will be used in later proofs. For positive integers $n$ and $k \leq n$, let $G(n, k)$ be the number of $k$-forests that have $n$ leaves.

Denote the $n$th *Catalan number* by $C_n$. The following properties are known [24]:

1) $C_{n-1} = G(n, 1)$
2) $C_n = \frac{1}{n+1}\binom{2n}{n}$
3) $C_n = C_0 C_{n-1} + C_1 C_{n-2} + \cdots + C_{n-1} C_0$.

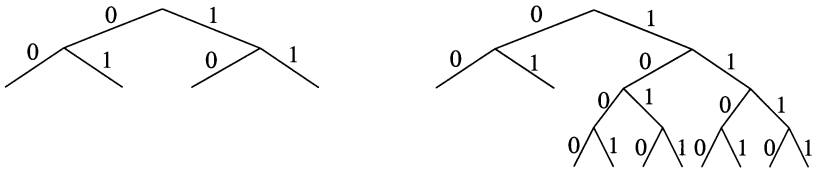

Fig. 2. Binary tree examples of complete prefix codes without self-synchronizing strings.

The quantity $C_{n-1}$ is the number of binary trees containing $n$ leaf nodes, or equivalently, the number of complete prefix codes with $n$ codewords. We now derive the values and limiting values for $G(n, k)$.

*Lemma IV.1:* $G(n, k) = \frac{k}{n}\binom{2n-k-1}{n-1}$.

*Proof:* Suppose the first of the $k$ trees is an isolated node. Then there are $n-1$ remaining leaves and $k-1$ remaining components, so the number of such $k$-forests is $G(n-1, k-1)$. Suppose instead that the first of the $k$ trees is *not* an isolated node, so the root of the first tree has children $a$ and $b$. We can now attach any trees to $a$, to $b$, and to the remaining $k-1$ roots subject to the condition that these $k+1$ trees have a total of $n$ leaves. The number of ways to do this is the number of $(k+1)$-forests with $n$ leaves, i.e., $G(n, k+1)$. Hence,

$$G(n, k) = G(n-1, k-1) + G(n, k+1) \tag{1}$$

or equivalently, replacing $k$ with $k-1$

$$G(n, k) = G(n, k-1) - G(n-1, k-2). \tag{2}$$

We now prove the lemma by induction on $k$. $G(n, 1)$ is the number of binary trees with $n$ leaves, so

$$G(n, 1) = C_{n-1} = \frac{1}{n}\binom{2n-2}{n-1}.$$

$G(n, 2)$ is the number of ordered pairs of binary trees with a total of $n$ leaves. We can count these pairs by assuming the two trees have $i$ leaves and $n-i$ leaves, respectively, and then summing over $i$. Thus, using the properties of the Catalan numbers

$$\begin{aligned} G(n, 2) &= C_0C_{n-2} + C_1C_{n-3} + \cdots + C_{n-2}C_0 \\ &= C_{n-1} \\ &= \frac{1}{n}\binom{2n-2}{n-1} = \frac{2}{n}\binom{2n-3}{n-1}. \end{aligned} \tag{3}$$

This establishes the two initial cases. The induction step follows from (2)

$$\begin{aligned} G(n, k) &= \frac{k-1}{n}\binom{2n-k}{n-1} - \frac{k-2}{n-1}\binom{2n-k-1}{n-2} \\ &= \frac{k-1}{n}\left[\binom{2n-k-1}{n-2} + \binom{2n-k-1}{n-1}\right] \\ &\quad - \frac{k-2}{n-1}\binom{2n-k-1}{n-2} \\ &= \frac{n-k+1}{n(n-1)}\binom{2n-k-1}{n-2} + \frac{k-1}{n}\binom{2n-k-1}{n-1} \\ &= \frac{1}{n}\binom{2n-k-1}{n-1} + \frac{k-1}{n}\binom{2n-k-1}{n-1} \\ &= \frac{k}{n}\binom{2n-k-1}{n-1}. \end{aligned}$$

□

*Lemma IV.2:* For all $k > 1$

$$\lim_{n\to\infty} \frac{G(n-1, k-1)}{G(n, k)} = \frac{k-1}{2k}$$
$$\lim_{n\to\infty} \frac{G(n, k+1)}{G(n, k)} = \frac{k+1}{2k}.$$

*Proof:* It follows from Lemma IV.1 that

$$\frac{G(n-1, k-1)}{G(n, k)} = \frac{n(k-1)}{k(2n-k-1)}$$

and

$$\frac{G(n, k+1)}{G(n, k)} = \frac{(k+1)(n-k)}{k(2n-k-1)}$$

from which the lemma follows. □

## V. Proof of the Main Theorem

For binary trees $S$ and $T$, we use the notation $S \prec T$ to indicate that $S$ is a subtree of $T$ and has the same root node. For any binary tree $T$, let $\mathcal{L}(T)$ denote the set of leaves of $T$. Given a binary tree $S$ and a set $L \subseteq \mathcal{L}(S)$, we define $\mathcal{T}(S, L, n)$ to be the set of $n$-leaf binary trees $T$ such that $S \prec T$ and $L \subseteq \mathcal{L}(T)$. When $S$ and $L$ are understood, we will use $L'$ to denote the complement of $L$ in $\mathcal{L}(S)$, i.e., $\mathcal{L}(S) \setminus L$.

*Lemma V.1:* $|\mathcal{T}(S, L, n)| = G(n - |L|, |L'|)$.

*Proof:* Each tree in $\mathcal{T}(S, L, n)$ consists of $S$ plus a tree attached to a subset of the nodes of $L'$. Thus, there is a one-to-one correspondence between the elements of $\mathcal{T}(S, L, n)$ and the set of all forests with $|L'|$ components and $n - |L|$ leaves. □

It is possible to select a binary tree $T$ uniformly at random from $\mathcal{T}(S, L, n)$ via a sequence of observations and outcomes. Each observation consists of selecting a node known to be in $T$, and seeing if it has children in $T$. Each step of this process reveals partial knowledge about $T$.

Thus, for a tree $T$ and one of its nodes $v$, define an *outcome* to be the mapping

$$o(v, T) = \begin{cases} 1, & \text{if } v \in \mathcal{L}(T) \\ 0, & \text{if } v \notin \mathcal{L}(T). \end{cases} \tag{4}$$

When a node $v$ is used in this way, it will be called an *observation*.

For a sequence of observations and outcomes, $(v_0, o_0)$, $(v_1, o_1)$, $\ldots$, $(v_{m-1}, o_{m-1})$ we say that the binary tree $T$ *satisfies* such a sequence if $o(v_i, T) = o_i$ for each $i$. For any trees $T$ and $S \prec T$, and any $v \in \mathcal{L}(S)\backslash\mathcal{L}(T)$, let $S^{(v)}$ denote the tree obtained by adjoining to $S$ the two children (in $T$) of $v$ and the two corresponding edges from $v$.

*Theorem V.2:* For every $S$, $L$, and $v \in L'$, the following hold:

i) $\{T \in \mathcal{T}(S, L, n)\colon o(v, T) = 1\} = \mathcal{T}(S, L \cup \{v\}, n)$;

ii) $\{T \in \mathcal{T}(S, L, n)\colon o(v, T) = 0\} = \mathcal{T}(S^{(v)}, L, n)$.

*Proof:*

i) Suppose $T \in \mathcal{T}(S, L, n)$ and $o(v, T) = 1$. Then $T$ contains $S$ and has $n$ leaves. Also, every node of $L$ is a leaf of $T$ as is $v$, so

every node of $L \cup \{v\}$ is a leaf of $T$. Thus, $T \in \mathcal{T}(S, L \cup \{v\}, n)$. Conversely, if $T \in \mathcal{T}(S, L \cup \{v\}, n)$, then $T$ contains $S$ and has $n$ leaves. Also, every node of $L$ is a leaf of $T$, as is $v$, so $T \in \mathcal{T}(S, L, n)$ and $o(v, T) = 1$.

ii) Suppose $T \in \mathcal{T}(S, L, n)$ and $o(v, T) = 0$. Then $T$ has $n$ leaves, and every node of $L$ is a leaf of $T$. Also, $T$ contains $S$ and the children of $v$, hence, $S^{(v)}$, so $T \in \mathcal{T}(S^{(v)}, L, n)$. Conversely, if $T \in \mathcal{T}(S^{(v)}, L, n)$, then $T$ has $n$ nodes, and every node of $L$ is a leaf of $T$. Also, $T$ contains $S^{(v)}$, so $T$ contains $S$, and $o(v, T) = 0$. Hence, $T \in \mathcal{T}(S, L, n)$ and $o(v, T) = 0$. □

We now give a brief outline of the remainder of this section. We begin by considering, for some fixed $l$, only those binary trees that have $0^l$ as a leaf. In Theorem V.3, we construct a specific sequence of observations and outcomes that guarantees the existence of a string that coalesces some fixed $0^i$ with the root. We also give a lower bound on the probability that a binary tree satisfies this sequence of outcomes. In Theorem V.6, we combine many such sequences to show that for almost all binary trees, there is a string that coalesces $0^i$ with the root. This will immediately imply Theorem V.8, that if a binary tree has $0^l$ as a leaf there is almost surely a self-synchronizing string. Finally, in Theorem V.10, we show that we can drop the condition that $0^l$ is a leaf.

*Theorem V.3:* Let $i$, $l$, and $n$ be positive integers. Let $S$ be a binary tree and let $L \subsetneq \mathcal{L}(S)$ such that $0^l \in L$. Assume $n > |L| + |L'| + l$ and $l > i$. Then there exists $z \in \{0, 1\}^*$, a nonnegative integer $\gamma \leq l$, and a nonempty sequence of observations and outcomes $(v_0, 1), (v_1, 0), \ldots, (v_\gamma, 0), (v_{\gamma+1}, 1)$ such that $z$ coalesces $0^i$ with the root in every tree $T \in \mathcal{T}(S, L, n)$ that satisfies this sequence. Furthermore, if $T$ is selected uniformly at random from $\mathcal{T}(S, L, n)$, and $P$ is the probability that $T$ satisfies this sequence, then

$$P \geq \frac{G(n - |L| - 2, |L'| + l - 2)}{G(n - |L|, |L'|)}.$$

*Proof:* The main idea of the proof if as follows. First, we find a string $w$ which brings the root and $0^i$ to nodes $a$ and $b$ (not necessarily respectively), and such that $0^\beta$ and $0^\alpha$, respectively, bring $a$ and $b$ to leaves $v_0$ and $v_1$ of $S$. We guarantee that at least one of these two leaves of $S$ is not a leaf of $T$. Then we show how to construct a sequence of $0s$ which coalesces $a$ and $b$ in $T$, provided $T$ satisfies a certain sequence of observations and outcomes. The specific observations and outcomes are chosen to synchronize the paths that $a$ and $b$ follow via a sequence of $0s$, allowing them to loop around the zero branch as many times as needed. Finally, we show that the fraction of trees in $\mathcal{T}(S, L, n)$ that satisfy such observations and outcomes is lower-bounded as in the statement of the theorem.

Define the set

$$\mathcal{B} = \{u\colon u \text{ is a node of } S \text{ and } u0^k \in L' \text{ for some } k \geq 0\}$$

and let $T \in \mathcal{T}(S, L, n)$. Let $w$ be any shortest string that brings either the root $r$ or $0^i$ to a node in $\mathcal{B}$ (note that $\mathcal{B}$ may contain leaves of $T$ but not the root). $w$ brings one of $r$ or $0^i$ to some $a \in \mathcal{B}$ and brings the other to some node $b$. Neither $a$ nor $b$ is the root, but they could be leaves. Observe that $b$ must be in $S$, for otherwise the path to $b$ induced by $w$ would pass through a node in $L' \subseteq \mathcal{B}$, violating the minimal length of $w$. Now let

- $v_0 = b0^\beta$ where $\beta$ is the smallest integer such that $b0^\beta \in \mathcal{L}(S)$;
- $v_1 = a0^\alpha$ where $\alpha$ is the smallest integer such that $a0^\alpha \in L'$;
- $\gamma \in \{1, 2, \ldots l\}$ such that $\gamma \equiv (\beta - \alpha) \bmod l$;
- $v_{i+1}$ be the 0-child of $v_i$ in $T$, for $i \geq 1$.

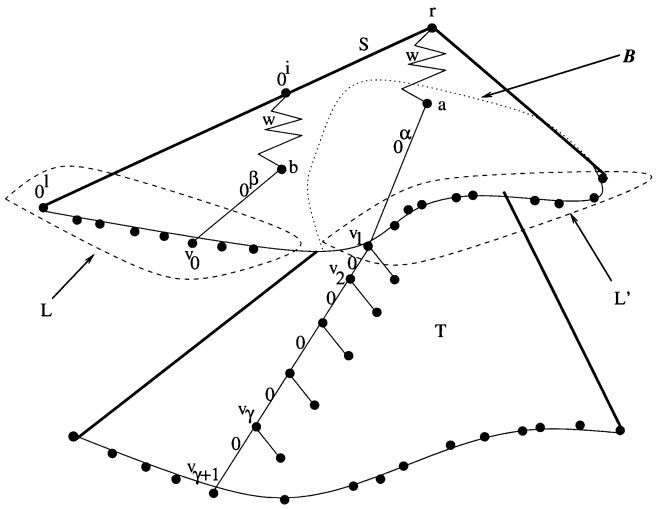


Fig. 3. Proof illustration, when $v_0 \in L$, and $w$ brings $r$ to $a$.

If $v_0 \in L$ and $T$ satisfies the sequence

$$(v_1, 0), (v_2, 0), \ldots, (v_\gamma, 0), (v_{\gamma+1}, 1)$$

or $v_0 \in L'$ and $T$ satisfies the sequence

$$(v_0, 1), (v_1, 0), (v_2, 0), \ldots, (v_\gamma, 0), (v_{\gamma+1}, 1)$$

then the string $0^{\max(\beta, \alpha+\gamma)}$ coalesces $a$ and $b$ in $T$. In both cases, $v_0$ is a leaf of $T$ and $v_1$ is $\gamma$ steps away from a leaf by following 0-children. Thus, if $T$ satisfies the sequence

$$(v_0, 1), (v_1, 0), (v_2, 0), \ldots, (v_\gamma, 0), (v_{\gamma+1}, 1)$$

then the string $z = w0^{\max(\beta, \alpha+\gamma)}$ coalesces $r$ and $0^i$. This is illustrated in Fig. 3.

We now derive the bound for the probability that a randomly chosen tree in $\mathcal{T}(S, L, n)$ satisfies this sequence. The probability $P$ that tree $T$ satisfies the observations and outcomes is the ratio of the number of possibilities for $T$ before the observations to the number of possibilities after the observations.

Define

$$\begin{aligned} &S_0 = S \\ &S_j = S_{j-1}^{(v_j)}, && \text{for } j = 1, \ldots, \gamma + 1 \\ &L_j = L \cup \{v_0, v_{j+1}\}, && \text{for } j = 0, \ldots, \gamma. \end{aligned} \tag{5}$$

If $v_0 \in L'$, then $L'_\gamma$ (i.e., $\mathcal{L}(S_\gamma) \backslash L_\gamma$) is the union of $L'$ and the 1-children of $v_1, v_2, \ldots, v_\gamma$ minus $\{v_0, v_1\}$. Therefore, by Lemma V.1

$$\begin{aligned} |\mathcal{T}(S_\gamma, L_\gamma, n)| &= G(n - |L_\gamma|, |L'_\gamma|) \\ &= G(n - |L| - 2, |L'| + \gamma - 2). \end{aligned}$$

If $v_0 \in L$, then $L'_\gamma$ is the union of $L'$ and the 1-children of $v_1, v_2, \ldots, v_\gamma$ minus $\{v_1\}$. Therefore, by Lemma V.1

$$\begin{aligned} |\mathcal{T}(S_\gamma, L_\gamma, n)| &= G(n - |L_\gamma|, |L'_\gamma|) \\ &= G(n - |L| - 1, |L'| + \gamma - 1). \end{aligned}$$

The pair $(S_\gamma, L_\gamma)$ represents the partial knowledge of the tree after the given sequence of observations and outcomes. The probability that $T$ satisfies the observations and outcomes is

$$P = \frac{|\mathcal{T}(S_\gamma, L_\gamma, n)|}{|\mathcal{T}(S, L, n)|}.$$

From (1), the numerator is smaller in the case when $v_0 \in L'$, and from (2), is smallest when $\gamma$ is as large as possible, namely, $\gamma = l$. By Lemma V.1, the denominator is

$$|\mathcal{T}(S, L, n)| = G(n - |L|, |L'|).$$

This achieves the lower bound in the theorem. □

In Theorem V.3, we gave a sequence of observations and outcomes that guarantee that a particular node coalesces with the root. We also gave a lower bound for the probability of these outcomes. We next show that for large $n$, this lower bound converges to a positive value.

*Lemma V.4:* For all $l \geq 2$, $k$, and $m$

$$\lim_{n\to\infty} \frac{G(n-k-2, m+l-2)}{G(n-k, m)} \geq 2^{-l-2}.$$

*Proof:* Let $n' = n - k$ and let

$$P(n) = \frac{G(n'-2, m+l-2)}{G(n', m)}.$$

Then $P(n)$ can be written as a telescoping product

$$P(n) = \frac{G(n'-1, m-1)}{G(n', m)} \left( \prod_{i=0}^{l-1} \frac{G(n'-1, m+i)}{G(n'-1, m+i-1)} \right) \cdot \frac{G(n'-2, m+l-2)}{G(n'-1, m+l-1)}.$$

As $n \to \infty$, the limits of the fractions in this product are given by Lemma IV.2. Thus,

$$\begin{aligned} \lim_{n\to\infty} P(n) &= \frac{m-1}{2m} \left( \prod_{i=0}^{l-1} \frac{m+i}{2(m+i-1)} \right) \frac{m+l-2}{2(m+l-1)} \\ &= \frac{1}{2^{l+2}} \cdot \binom{m+l-2}{m} \\ &\geq \frac{1}{2^{l+2}}. \end{aligned}$$

□

Note that Lemma V.4 implies in particular, that

$$\frac{G(n-k-2, m+l-2)}{G(n-k, m)} > 2^{-l-3}$$

for sufficiently large $n$. An immediate consequence of Theorem V.3 and Lemma V.4 is the following corollary.

*Corollary V.5:* Let $i$ and $l > i$ be positive integers, let $S$ be a binary tree, and let $L \subsetneq \mathcal{L}(S)$ such that $0^l \in L$. If $T$ is selected from $\mathcal{T}(S, L, n)$ uniformly at random, then for sufficiently large $n$

$$\boldsymbol{P}(\text{the root and } 0^i \text{ do not coalesce in } T) < 1 - 2^{-l-3}.$$

*Theorem V.6:* Let $i$ and $l > i$ be positive integers, let $S$ be a binary tree, and let $L \subsetneq \mathcal{L}(S)$ such that $0^l \in L$. If $T$ is selected from $\mathcal{T}(S, L, n)$ uniformly at random, then

$$\lim_{n\to\infty} \boldsymbol{P}(\text{the root and } 0^i \text{ coalesce in } T) = 1.$$

*Proof:* We will show by induction that for all $k \geq 1$ and sufficiently large $n$

$$\boldsymbol{P}(\text{the root and } 0^i \text{ coalesce in } T) > 1 - (1 - 2^{-l-3})^k. \quad (6)$$

This expression converges to 1 as $k$ grows, for fixed $l$, so the result will follow. Equation (6) holds in the case $k = 1$ by Corollary V.5. Now assume (6) is true for all $k \leq k'$ (where $k' \geq 1$). Let

$$V_{\gamma+1} = \{(v_0, 1), (v_1, 0), (v_2, 0), \ldots, (v_\gamma, 0), (v_{\gamma+1}, 1)\}$$

be a sequence of observations and outcomes as guaranteed by Theorem V.3. In particular, if $T$ satisfies $V_{\gamma+1}$ then $0^i$ and the root coalesce in $T$. Define the prefixes of $V_{\gamma+1}$ as

$$\begin{aligned} V_{-1} &= \emptyset \\ V_j &= \{(v_0, o_0), \ldots, (v_j, o_j)\}, \qquad \text{for } j = 0, \ldots, \gamma+1 \end{aligned}$$

where

$$\begin{aligned} o_0 &= o_{\gamma+1} = 1 \\ o_j &= 0, \qquad \text{for } j = 1, \ldots, \gamma. \end{aligned}$$

Define $S_j$ and $L_j$ for $j = 1, \ldots, \gamma$ as in (5), and define the events

$$F_j = \{T \in \mathcal{T}(S, L, n) \colon T \text{ satisfies } V_{j-1} \cup \{(v_j, 1 - o_j)\}\}$$

for $j = 0, \ldots, \gamma+1$. That is, $F_j$ is the event that $T$ satisfies the first $j$ observations and outcomes, but not the $(j+1)^{st}$ in the sequence $V_{\gamma+1}$. Let $H$ denote the event that $T$ does not satisfy $V_{\gamma+1}$. Then $H$ is a disjoint union of events, namely

$$H = \bigcup_{j=0}^{\gamma+1} F_j.$$

Let $U_i$ denote the event that the root and $0^i$ do not coalesce in $T$. If $U_i$ occurs, then $T \in H$. Thus,

$$\begin{aligned} P(U_i) &= P(T \in H, U_i) \\ &= \sum_{j=0}^{\gamma+1} P(T \in F_j, U_i) \\ &= P(T \in F_0) P(U_i \mid T \in \mathcal{T}\left(S^{(v_0)}, L, n\right)) \\ &\quad + \sum_{j=1}^{\gamma} P(T \in F_j) P(U_i \mid T \in \mathcal{T}(S_{j-1}, L_{j-1}, n)) \\ &\quad + P(T \in F_{\gamma+1}) P(U_i \mid T \in \mathcal{T}(S_{\gamma+1}, L \cup \{v_0\}, n)) \quad (7) \\ &< \sum_{j=0}^{\gamma+1} P(T \in F_j)(1 - 2^{-l-3})^{k'} \quad (8) \\ &= P(T \in H)(1 - 2^{-l-3})^{k'} \\ &< (1 - 2^{-l-3})^{k'+1} \quad (9) \end{aligned}$$

where (7) follows from Theorem V.2 assuming all conditioning events are nonempty (if $v_0 \in L$ then $P(T \in F_0) = 0$ and (7) trivially follows), (8) follows from the induction hypothesis (which applies to any fixed $S$ and $L$ in the statement of Theorem V.6), and (9) follows from Corollary V.5 (since $H \subset \mathcal{T}(S, L, n)$). □

*Lemma V.7:* For all $l \geq 1$, if $S$ is the binary tree corresponding to the code $\{0^l\} \cup \{0^i 1 \colon 0 \leq i \leq l-1\}$ and $L = \{0^l\}$, then $\mathcal{T}(S, L, n)$ is the set of $n$-leaf trees containing the codeword $0^l$.

*Proof:* Suppose $T$ is an $n$-leaf binary tree. If $0^l$ is a leaf of $T$, then $T$ must contain $S$, and $L \subset \mathcal{L}(T)$. Therefore, $T \in \mathcal{T}(S, L, n)$. Conversely, if $T \in \mathcal{T}(S, L, n)$, then $T$ has $0^l$ as a leaf. □

*Theorem V.8:* For all $l \geq 1$, if $T$ is selected uniformly at random from the set of $n$-leaf binary trees that have $0^l$ as a leaf, then

$$\lim_{n\to\infty} \boldsymbol{P}(T \text{ has a self-synchronizing string}) = 1.$$

*Proof:* By Lemma V.7, it is equivalent to say that $T$ is selected uniformly at random from $\mathcal{T}(S, L, n)$, where $S$ and $L$ are as defined in the statement of that lemma. Let $E_i$ denote the event that $0^i$ and the root coalesce in $T$. Clearly, $P(E_0) = 1$ since the empty string $0^0$ is the root. By Theorem V.6, for $0 < i < l$, $\lim_{n\to\infty} \boldsymbol{P}(E_i) = 1$. For all $l \geq 1$, it follows that

$$\lim_{n\to\infty} \boldsymbol{P}(E_0 \cap E_1 \cap \cdots \cap E_{l-1}) = 1.$$

By Lemma III.5, the event $E_0 \cap E_1 \cap \cdots \cap E_{l-1}$ implies that $T$ has a self-synchronizing string. □

We have proved the main result, Theorem III.4, in the special case that $0^l$ is a leaf of $T$. In order to deduce the main result from this special case, we need the following lemma.

*Lemma V.9:* If $T$ is an $n$-leaf binary tree selected uniformly at random, then

$$\lim_{n\to\infty} P(0^l \in \mathcal{L}(T)) = \frac{l}{2^{l+1}}.$$

*Proof:* The total number of $n$-leaf binary trees is $G(n, 1)$. Let $S$ and $L$ be defined as in the statement of Lemma V.7. By that lemma, the number of $n$-leaf binary trees that have $0^l$ as a leaf is $|\mathcal{T}(S, L, n)|$. By Lemma V.1

$$|\mathcal{T}(S, L, n)| = G(n - |L|, |L'|) = G(n - 1, l).$$

Thus,

$$P(0^l \in \mathcal{L}(T)) = \frac{G(n-1, l)}{G(n, 1)}.$$

By Lemma IV.1, this is equal to

$$\frac{l \cdot n \cdot (n-1) \cdots (n-l)}{(2n-2) \cdot (2n-3) \cdots (2n-l-2)}.$$

We then take the limit as $n \to \infty$. □

*Theorem V.10:* Almost all $n$-leaf binary trees have a self-synchronizing string.

*Proof:* Let $q_n$ be the probability that an $n$-leaf binary tree $T$, chosen uniformly at random, has a self-synchronizing string. For $l < n$, define the quantities

$$a_{n,l} = \boldsymbol{P}(T \text{ has a self-synchronizing string} \,|\, 0^l \in \mathcal{L}(T))$$
$$b_{n,l} = \boldsymbol{P}(0^l \in \mathcal{L}(T))$$

and for $n \leq l$, let $a_{n,l} = b_{n,l} = 0$ ($0^l \in \mathcal{L}(T)$ is impossible in that case). Therefore,

$$\begin{aligned}
\lim_{n\to\infty} q_n &= \lim_{n\to\infty} \sum_{l=1}^{\infty} a_{n,l} b_{n,l} \\
&\geq \sum_{l=1}^{\infty} \lim_{n\to\infty} a_{n,l} b_{n,l} \\
&\quad (\text{by Fatou's lemma since } a_{n,l},\, b_{n,l} \geq 0) \\
&= \sum_{l=1}^{\infty} \left(\lim_{n\to\infty} a_{n,l}\right)\left(\lim_{n\to\infty} b_{n,l}\right) \\
&= \sum_{l=1}^{\infty} (1) \left(\frac{l}{2^{l+1}}\right) \\
&\quad (\text{by Theorem V.8 and Lemma V.9}) \\
&= 1.
\end{aligned}$$

□

Theorem V.10 immediately implies Theorem III.4.

## ACKNOWLEDGMENT

The authors wish to thank Al Hales for many helpful comments, and Gadiel Seroussi for pointing out Knuth's use of the terminology "extended tree."

## REFERENCES

[1] J. Berstel and D. Perrin, *Theory of Codes*. New York: Academic, 1985.
[2] R. M. Capocelli, L. Gargano, and U. Vaccaro, "On the characterization of statistically synchronizable variable-length codes," *IEEE Trans. Inform. Theory*, vol. 34, pp. 817–825, July 1988.
[3] R. M. Capocelli, A. A. De Santis, L. Gargano, and U. Vaccaro, "On the construction of statistically synchronizable codes," *IEEE Trans. Inform. Theory*, vol. 38, pp. 407–414, Mar. 1992.
[4] T. M. Cover and J. A. Thomas, *Introduction to Information Theory*. New York: Wiley, 1991.
[5] I. Csiszár and J. Körner, *Information Theory: Coding Theorems for Discrete Memoryless Systems*. New York: Academic, 1981.
[6] A. E. Escott and S. Perkins, "Binary Huffman equivalent codes with a short synchronizing codeword," *IEEE Trans. Inform. Theory*, vol. 44, pp. 346–351, Jan. 1998.
[7] S. Even, "Test for synchronizability of finite automata and variable length codes," *IEEE Trans. Inform. Theory*, vol. IT-10, pp. 185–189, July 1964.
[8] T. J. Ferguson and J. H. Rabinowitz, "Self-synchronizing Huffman codes," *IEEE Trans. Inform. Theory*, vol. IT-30, pp. 687–693, July 1984.
[9] R. Gallager, *Information Theory and Reliable Communication*. Englewood Cliffs, NJ: Prentice-Hall, 1968.
[10] E. N. Gilbert, "Synchronization of binary messages," *Proc. IRE*, vol. 48, pp. 470–477, Sept. 1960.
[11] E. N. Gilbert and E. F. Moore, "Variable-length binary encodings," *Bell Syst. Tech. J.*, vol. 38, pp. 933–967, July 1959.
[12] S. W. Golomb and B. Gordon, "Codes with bounded synchronization delay," *Inform. Contr.*, vol. 8, pp. 355–372, 1965.
[13] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, pp. 1098–1101, Sept. 1952.
[14] D. E. Knuth, *The Art of Computer Programming*. Reading, MA: Addison-Wesley, 1973, vol. 3.
[15] W.-M. Lam and R. Kulkarni, "Extended synchronizing codewords for binary prefix codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 984–987, May 1996.
[16] V. I. Levenshtein, "Certain properties of code systems," *Soviet Phys.–Dokl.*, vol. 6, no. 6, pp. 858–860, 1962.
[17] J. C. Maxted and J. P. Robinson, "Error recovery for variable length codes," *IEEE Trans. Inform. Theory*, vol. IT-31, pp. 794–801, Nov. 1985.
[18] R. McEliece, *The Theory of Information and Coding*. Reading, MA: Addison-Wesley, 1977.
[19] M. E. Monaco and J. M. Lawler, "Corrections and additions to "error recovery for variable-length codes"," *IEEE Trans. Inform. Theory*, vol. IT-33, pp. 454–456, May 1987.
[20] B. L. Montgomery and J. Abrahams, "Synchronization of binary source codes," *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 849–854, Nov. 1986.
[21] H. Morita, A. J. van Wijngaarden, and A. J. Han Vinck, "On the construction of maximal prefix-synchronized codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 2158–2166, Nov. 1996.
[22] P. G. Neumann, "Efficient error-limiting variable length codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 292–304, July 1962.
[23] M. Rahman and S. Misbahuddin, "Effects of a binary symmetric channel on the synchronization recovery of variable length codes," *Comput. J.*, vol. 32, pp. 246–251, 1989.

[24] K. H. Rosen, Ed., *Handbook of Discrete and Combinatorial Mathematics*. Boca Raton, FL: CRC, 2000.
[25] B. Rudner, "Construction of minimum-redundancy codes with an optimum synchronization property," *IEEE Trans. Information Theory*, vol. IT-17, pp. 478–487, July 1971.
[26] P. F. Swaszek and P. DiCicco, "More on the error recovery for variable-length codes," *IEEE Trans. Inform. Theory*, vol. 41, pp. 2064–2071, Nov. 1995.
[27] Y. Takishima, M. Wada, and H. Murakami, "Error states and synchronization recovery for variable length codes," *IEEE Trans. Commun.*, vol. 42, pp. 783–792, Feb. 1994.
[28] M. R. Titchener, "The synchronizaation of variable-length codes," *IEEE Trans. Inform. Theory*, vol. 43, pp. 683–691, Mar. 1997.
[29] V. K. W. Wei and R. A. Scholtz, "On the characterization of statistically synchronizable codes," *IEEE Trans. Inform. Theory*, vol. IT-26, pp. 733–735, Nov. 1980.
[30] G. Zhou and Z. Zhang, "Synchronization receovery of variable-length codes," *IEEE Trans. Inform. Theory*, vol. 48, pp. 219–227, Jan. 2002.

# A Coding Theorem for Lossy Data Compression by LDPC Codes

Yuko Matsunaga and Hirosuke Yamamoto, *Member, IEEE*

***Abstract*—In this correspondence, low-density parity-check (LDPC) codes are applied to lossy source coding and we study how the asymptotic performance of MacKay's LDPC codes depends on the sparsity of the parity-check matrices in the source coding of the binary independent and identically distributed (i.i.d.) source with $\Pr\{x = 1\} = 0.5$. In the sequel, it is shown that an LDPC code with column weight $O(\log n)$ for code length $n$ can attain the rate-distortion function asymptotically.**

***Index Terms*—Lossy data compression, low-density parity-check (LDPC) codes, rate-distortion function.**

## I. Introduction

Recently, low-density parity-check (LDPC) codes, which were originally discovered by Gallager [1], [2] in 1962, have been studied actively because of their very good performance in error correction. It was shown by many researchers [3]–[8] that LDPC codes can attain high performance near the Shannon limit by iterative decoding with belief propagation [9], [10]. Furthermore, MacKay [3] and Miller and Burshtein [11] proved that the LDPC codes can asymptotically achieve channel capacity.

On the other hand, it is well known that channel coding can be considered as the dual problem of lossy source coding in rate-distortion theory [12], and a good error-correcting code can be used for efficient lossy data compression. Csiszár and Körner [13] showed that the rate-distortion function can be achieved by an error-correcting code that can attain error probability $P_e = 1 - \epsilon$ with rate close to the rate-distortion function for any sufficiently small fixed $\epsilon > 0$. However, for LDPC codes or other practical error-correcting codes, the existence of such codes is not obvious, and hence we have to prove the existence if we want to use such codes for lossy data compression. Actually, for example, Goblick [14] and Berger [15] showed that a general linear error-correcting code can attain the rate-distortion function asymptotically for a binary independent and identically distributed (i.i.d.) source with $\Pr\{x = 1\} = 0.5$. Furthermore, Viterbi and Omura [16] proved a lossy source coding theorem for i.i.d. sources by using trellis codes with Viterbi decoding. However, no *practically good* lossy source coding scheme has been found because the encoding methods used in the lossy source coding schemes are not practical in the sense of memory and time complexity.

Based on the good performance of the LDPC codes and the practically feasible belief propagation decoding, we may expect that LDPC codes are good candidates for practical lossy source coding. However, it is known that belief propagation decoding does not work well in lossy source coding because the noise level of the test channel in lossy source coding is much larger than that of usual channels used in practical error correction. Nevertheless, it is still worth to clarify if the LDPC codes can attain the rate-distortion function asymptotically because some cleverly modified version of belief propagation decoding or an entirely new decoding algorithm may be devised exploiting the sparsity of parity-check matrix in the future.

In this correspondence, we treat LDPC codes on MacKay's ensemble [3] for the binary i.i.d. source with $\Pr\{x = 1\} = 0.5$, and we consider the relation between the sparsity of parity-check matrices and the asymptotic performance in LDPC codes. Furthermore, we show that for code length $n$, an LDPC code with column weight $O(\log n)$ can attain the rate-distortion function asymptotically.

The correspondence is organized as follows. Section II is devoted to some preliminaries of the rate-distortion theory and the lossy source coding by LDPC codes. The main coding theorem is also described in Section II, but the proof is given in Section III. It is also shown in Section III how the sparsity of parity-check matrices affects the performance of LDPC codes in the lossy source coding.

## II. Lossy Data Compression by LDPC Codes

Let source X be a binary i.i.d. source which takes values in $\mathcal{X} = \{0, 1\}$ with $q = \Pr\{x = 1\}$. The distortion between single letters is measured by the Hamming distortion defined by

$$d_{\mathrm{H}}(x, \hat{x}) = \begin{cases} 0, & \text{if } x = \hat{x} \\ 1, & \text{if } x \neq \hat{x} \end{cases}$$

and the distortion between $n$-bit sequences[1] $\boldsymbol{x} = x_1, x_2, \cdots, x_n^T$ and $\hat{\boldsymbol{x}} = \hat{x}_1, \hat{x}_2, \cdots, \hat{x}_n^T$ is measured by the averaged single-letter distortion as follows:

$$d(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \frac{1}{n} \sum_{i=1}^{n} d_{\mathrm{H}}(x_i, \hat{x}_i).$$

Then the rate-distortion function $R(D)$ of the binary i.i.d. source is given by

$$R(D) = \min_{\hat{X}:\, \mathrm{E}[d_{\mathrm{H}}(X, \hat{X})] \leq D} I(X; \hat{X})$$
$$= h(q) - h(D), \qquad 0 \leq D \leq q \leq 0.5$$

[1]In this correspondence, a bold-faced letter represents a column vector and $\boldsymbol{T}$ stands for transposition of a vector or a matrix

Manuscript received November 21, 2002; revised May 13, 2003. The work of H. Yamamoto was supported in part by JSPS under Grant-in-Aid for Scientific Research 14550347. The material in this correspondence was presented at the IEEE International Symposium on Information Theory, Lausanne, Switzerland, June/July 2002.

Y. Matsunaga is with Internet Systems Research Laboratories, NEC Corporation, 4-1-1 Miyazaki, Miyamae-ku, Kawasaki, Kanagawa, 216-8555, Japan (e-mail: y-matsunaga@da.jp.nec.com).

H. Yamamoto is with the Department of Mathematical Informatics, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656, Japan (e-mail: Hirosuke@ieee.org).

Communicated by R. Koetter, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2003.815805