Polynomial Time Low-Density Parity-Check Codes With Rates Very Close to the Capacity of the q-ary Random Deletion Channel for Large q

Michael Mitzenmacher, *Member, IEEE*

***Abstract*—We demonstrate polynomial-time deletion codes based on the verification-based decoding paradigm that come arbitrarily close to capacity. The random deletion channel takes n symbols from a q-ary alphabet, and each symbol is deleted independently with probability p. Taking advantage of recent improvements on the results of Luby and Mitzenmacher for verification-based decoding by Shokrollahi and Wang, we show how to design for any $\epsilon > 0$ and sufficiently large n and q deletion codes with the following properties: the rate is $(1-p)(1-\epsilon)$, the failure probability is $n^{O(1/\epsilon^2)}/q$, and the computational complexity for encoding and decoding is $n^{O(1/\epsilon^2)} \log q$. We also extend these schemes to obtain the same results even if the undeleted symbols are also transposed arbitrarily, and if a sufficiently small number of random symbols are inserted.**

***Index Terms*—Deletion channel, low-density parity-check codes (LDPC), q-ary symmetric channel, verification-based decoding.**

I. Introduction

The random deletion channel deletes each symbol sent independently with probability p. If the symbols sent are from a q-ary alphabet, with sufficiently large q, then by embedding sequence numbers into the transmitted symbols, one can make the deletion channel function like an erasure channel. For example, using log for $\log_2$ throughout, if each of $q = 2^b$ symbols is represented by b bits, and n symbols are sent, by using $\lceil \log n \rceil$ of the b bits as a sequence number, one can obtain a functional rate of $(1-p)(1-\lceil \log n \rceil / b) \approx (1-p)$. In fact, since for constant p only $O(\log n)$ consecutive packets are ever lost with high probability, sequence numbers that track the position modulo $c \log n$ for some sufficiently large constant c will suffice with high probability, reducing the bits required for the sequence number to $O(\log \log n)$.

An interesting question is to determine whether similar results are achievable without the embedding of sequence numbers. Diggavi and Grossglauser demonstrate that for large alphabets, choosing a random codebook is sufficient to obtain rates near $1-p$, using a Shannon-style argument [3]. Their result is nonconstructive, however, as one might expect, and decoding requires exponential time to test whether the received sequence is a subsequence of each possible codeword.

Here, we demonstrate new families of codes for the random deletion channel based on low-density parity-check (LDPC) codes using the framework of *verification-based decoding*, introduced in [6]. These codes allow encoding and decoding in time polynomial in n, albeit with an exponent of $O(1/\epsilon^2)$, where the rate is at least $(1-p)(1-\epsilon)$.

While our main focus is this capacity result, we also show that a variation of the verification-based decoding scheme allows for successful decoding even if the channel arbitrarily transposes the symbols that successfully pass through without being deleted. This of course can also be handled using $\lceil \log n \rceil$-bit sequence numbers; however, the random codebook argument of [3] does not apply to this case. Moreover, we show verification-based decoding is also robust to certain random substitution and insertion errors that sequence numbers alone cannot handle.

Because standard deletions can be handled using sequence numbers, and because our current schemes are computationally inefficient, our results at this point are primarily of theoretical interest, highlighting how LDPC codes might be used as an alternative approach in this setting. We believe, however, that this general approach, if made more efficient, could give viable alternatives in many practical situations. For example, with small packets (e.g., 64 or 128 bits), sequence numbers may be expensive in terms of bits required, and verification-based schemes may be more efficient. Also, if substitution, transposition, or insertion errors may also occur, then verification-based decoding may also offer significant advantages.

We begin by extending and elaborating on the approach summarized in [12]. In particular, we focus on the high-level connection between LDPC analysis for deletion channels and the q-ary symmetric channel (qSC). We then utilize the idea of precoding, applied by Shokrollahi and Wang [17] to obtain near–capacity verification-based decoding on the qSC, to design near-capacity codes on the deletion channel. We subsequently explain how random substitution and insertion errors as well as arbitrary transpositions can be handled.

A. Previous Work

Deletion codes, although not as well studied as erasure and error-correcting codes, have a long history; see [2], [3], [15], [18] for details and references. More recently, the case of detecting a single deletion is covered in a recent survey of Sloane [18]. For the binary channel, Diggavi and Grossglauser consider variations of Shannon's theorem to prove bounds on the capacity of a channel that deletes bits independently with probability p [3]; their bounds have been improved by Drinea and Mitzenmacher [4], [5]. Davey and MacKay design codes for channels that can insert, delete, or substitute bits, using various approaches to determine the probability of the value for each sent bit and applying an LDPC code on the end results [2].

Verification-based decoding utilizes LDPC codes in the setting where erroneous values lead to a false verification with suitably small probability [6]. Earlier, similar ideas were also suggested by Metzner [1], [10], [11], including ideas corresponding to verification, although the application to deletion channels and related channels as well as the corresponding LDPC analysis we provide are all new. As previously mentioned, the analysis of [6] for the qSC was extended by Shokrollahi and Wang [17], who show that by using precoding, a simple verification-based decoding scheme can achieve near-capacity performance on the qSC.

II. The LDPC Setup

We briefly summarize the standard framework for LDPC codes, following [9]. LDPC codes can be represented by bipartite graphs. On one side, the n *variable nodes* correspond to symbols in the codeword. On the other side, the m *check nodes* correspond to constraints on the adjacent variable nodes. The design rate R is given by $R = \frac{n-m}{n}$. (The actual rate R tends to be slightly higher than the design rate R in practice, because the check nodes are not necessarily all linearly independent. We ignore this distinction henceforth.) The q-ary alphabet is assumed to have an appropriate group structure and group operation. Specifically, the symbols can be interpreted as numbers modulo q, with the constraints being on the sum of values of the variable nodes modulo q. When $q = 2^b$, as when the packet is a string

Manuscript received March 16, 2005; revised June 28, 2006. This work was supported in part by an Alfred P. Sloan Research Fellowship and the National Science Foundation under Grants CCR-9983832, CCR-0118701, and CCR-0121154. The material in this correspondence was presented in part at the IEEE International Symposium on Information Theory, Yokohama, Japan, June/July 2003.

The author is with Harvard University, Division of Engineering and Applied Sciences, Cambridge, MA 02138 USA (e-mail: michaelm@eecs.harvard.edu).

Communicated by G. Zémor, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2006.885443

of b bits, we may use the group $(\boldsymbol{Z}_2)^b$, so the sum operation is a bitwise exclusive–or and the constraints are packet-level parity-check constraints.

A family of codes can be determined by assigning degree distributions to the variable and check nodes. Let $\vec{\lambda} = (\lambda_2, \ldots, \lambda_{d_v})$ be the vector such that the fraction of edges connected to variable nodes of degree i is λ_i, and similarly let $\vec{\rho} = (\rho_2, \ldots, \rho_{d_c})$ be such that the fraction of edges connected to check nodes of degree i is ρ_i. We assume a minimum degree of two throughout. We define the polynomials

$$\lambda(x) := \sum_{i=2}^{d_v} \lambda_i x^{i-1} \quad \text{and} \quad \rho(x) := \sum_{i=2}^{d_c} \rho_i x^{i-1}.$$

The λ_i and ρ_i variables must satisfy a constraint so that the number of edges is the same on both sides of the bipartite graph. This constraint is easily specified in terms of the design rate by the equation

$$R = 1 - \frac{\int_0^1 \rho(x)dx}{\int_0^1 \lambda(x)dx}.$$

Once degrees have been chosen for each node (so that the total degree of the check nodes and the variable nodes are equal), a specific random code can be chosen by mapping the edge connections of the variable nodes to the edge connections of the check nodes via a random permutation.

When we consider message passing algorithms below, to determine their asymptotic performance of such codes, it suffices to consider the case where the neighborhood of each node is a tree (that is, no cycles) for some number of levels. Analysis in this case is greatly simplified since random variables that correspond to messages in our message passing algorithms can be treated as independent. The analysis is accurate because the graph is asymptotically tree-like, and can be made rigorous using now-standard martingale arguments, as in [7]–[13].

III. A Simple Algorithm for the Deletion Channel

To develop our approach, we begin with a simple, nonoptimal scheme. This scheme was summarized in the original publication of this work [12].

In the context of the random deletion channel, we assume that the data being sent consists of independent random symbols from the q-ary alphabet, and further that the constraints are such that the sums of the symbols associated with each check node c are also independent random symbols from the q-ary alphabet, predetermined by the sender and receiver. That is, the code consists of a single bipartite graph with variable nodes and check nodes. Initially there are (approximately) $n - m$ values to be sent, stored in specific variable nodes, and the remaining m are determined by the random constraints. Linear time encoding and decoding schemes also exist for such single-layer schemes [14]. In practice, nonrandom data can be make to appear random if the sender and receiver share a source of (pseudo)-randomness: preprocess the data by adding an independent pseudorandom value to each symbol. (These random values can be thought of as part of the description of the code.)

To analyze the decoding, we think of each variable node as being in one of two possible states: *unverified* and *verified*. When a node is unverified, the algorithm has not yet fixed the final value for that node. Hence, the decoding algorithm begins with all nodes being unverified. When a node is verified, it is given a fixed value for the remainder of the algorithm. Hence the algorithm should, with high probability, never assign a verified node an incorrect value. We shall also refer to a variable node that was deleted but becomes verified as having been *recovered*.

We first explain our scheme for the case where each check node has fixed degree $d_c > 2$. Suppose that a out of n symbols values arrive. The decoding algorithm begins by considering all $\binom{a}{d_c}$ ways of summing d_c of the a nodes. The decoding algorithm then applies the following rules in any order as much as possible.

1) For any check node c, suppose that j of its neighbors have been verified. If there exists a combination of $d_c - j$ symbols that have arrived such that the sum of these $d_c - j$ values and the values of the verified neighbors of c equals the value associated with c, then verify these $d_c - j$ remaining neighbors of c. That is, label all of the neighbors of this check node as verified, and assign the value of the kth unverified neighbor of this node to be the kth of the $d_c - j$ values in order of arrival.
2) If all but one of the neighbors of a check node is verified, the remaining neighbor becomes verified, with its final value being set so that the sum of all neighbors of the check node equals the associated value.

In this decoding process, the check nodes play two roles. First, they may *verify* that the appropriate positions of a collection of nodes are correct, according to first rule. This verification rule applies because of the following fact: if the sum of the values of a collection of i nodes matches the value associated with a check node of degree i, then with high probability these nodes must be the neighbors of that check node. Concretely, we have the following lemma.

Lemma 1: The probability of a false verification at any point in the decoding process is bounded above by $m\binom{n}{d_c}/q$.

Proof: Consider any step of the algorithm where a check node c verifies its remaining unverified neighbors. For a false verification to occur at some point in the decoding, there must be a collection of d_c variable nodes with values that sum to the value of the check node c, other than the d_c neighbors of c. But any collection of d_c variable nodes (other than the neighbors of c) includes a variable node whose value is independent of the check node, and conditioned on the values of the other $d_c - 1$ nodes in the collection, the probability that this variable node takes on the precise value that will cause c to make a false verification is at most $1/q$. The bound of $m\binom{n}{d_c}/q$ then follows by a trivial union bound. □

To see the value of the above lemma, consider the case where symbols consist of b bits, so $q = 2^b$. Thus, the probability of a failure from a false verification is exponentially small in b at each step. Hence, if d_c is constant, the packet size b needs to be only $\Omega(\log n)$ bits in order that the probability of failure due to a false verification be polynomially small in n. For codes that are ϵ-close to capacity, we will see below that we may require d_c to grow like $\Theta(1/\epsilon^2)$, and in this case b will need to be $\Omega(\log n/\epsilon^2)$.

The other role of a check node is to recover a neighboring variable node that was not received, according to Rule 2. A check node can recover a neighbor after all other neighbors have been verified and therefore are known to have the correct value with high probability.

The decoding algorithm above is easily generalized to the case where the check nodes do not all have the same degree. In this case, when dealing with a check node c of degree $\deg(c)$, one must check all combinations of $\deg(c)$ variable nodes for possible verifications. There are therefore still only $O(n^{d_c})$ possible combinations to consider. The total probability of a false verification remains $O(mn^{d_c}/q)$ by essentially the same argument as Lemma 1. It will turn out that right-regular codes, or codes where check nodes all have the same degree, will suffice to obtain near-capacity achieving codes, although our analysis holds more generally.

A. A Message-Passing Decoding Algorithm

The algorithm above is easily modified to become a message-passing algorithm, which allows us to apply standard analysis techniques to determine the asymptotic fraction of deletions tolerable under our decoding process. Consider an edge (v, c) between a variable node v and a check node c, and the associated tree describing the neighborhood of v. The tree is rooted at v, and the tree branches out from the check nodes of v excluding c.

The decoding process occurs in rounds, with each round having two stages. In the first stage, each variable node passes to each neighboring check node in parallel its current state and, if verified, a value. In the second stage, if possible any $c' \neq c$ that neighbors v verifies v and provides it an appropriate value. (In the analysis, we think of each variable node as passing on to the check node c the current value *excluding any information obtained directly from* c. This avoids the problem of a circular flow of information.) Note that in this second stage each check node c' must determine if some combination of d_c arrived or recovered variable node values (that include any verified neighbors of c') sum appropriately so that v can be verified. This can be done by, for example, keeping a sorted list of all combinations of d_c arrived or recovered variable node values.

Let a_j be the probability that v arrived but is unverified after j rounds, and let b_j be the probability that v did not arrive and is unverified after j rounds. Hence, $1 - a_j - b_j$ is the probability that v has been verified through the tree rooted at v within j rounds. We ignore the possibility in the analysis that a false verification occurs, since by Lemma 1 this occurs with negligible probability when the alphabet is sufficiently large. (Technically, we can simply condition on a false verification not happening.) Initially $a_0 = 1 - p$ and $b_0 = p$. The decoding algorithm succeeds if $a_j + b_j$ converges to 0.

The evolution of the process from round to round, assuming that the neighborhood of v is given by a tree, is given by

$$a_{j+1} = a_0 \lambda(1 - \rho(1 - b_j)) \tag{1}$$

$$b_{j+1} = b_0 \lambda(1 - \rho(1 - a_j - b_j)). \tag{2}$$

We explain the derivation of (2). A node v that was deleted and remains unverified in the $(j+1)$th round was initially deleted with probability b_0. Also, it cannot be the case that there is some check node c' other than c neighboring v that has all of its children verified after j rounds, or else v would be recovered in the $(j+1)$th round. Now each c' has $k-1$ children below it with probability ρ_k, and each child is verified after j rounds with probability $1 - a_j - b_j$. The probability that v has not been recovered by a specific check node c' by round j is therefore

$$\sum_i \rho_i (1 - a_j - b_j)^{i-1} = \rho(1 - a_j - b_j).$$

As v has $k-1$ other neighboring check nodes besides c with probability λ_k, the probability that v remains unverified in round $j+1$ is

$$\sum_i \lambda_i (1 - \rho(1 - a_j - b_j))^{i-1} = \lambda(1 - \rho(1 - a_j - b_j)).$$

This yields (2); (1) is derived by similar considerations.

An important observation at this point is that the preceding analysis is entirely similar to the original analysis for verification codes for errors on the qSC given by Luby and Mitzenmacher in [6]. A deletion corresponds to an error; a deleted symbol that has not been subsequently recovered (and verified) corresponds to an erroneous symbol that has not been corrected (and verified); an undeleted symbol that has not been verified corresponds to a symbol that arrived without error but has not been verified. This mapping between the deletion setting and qSC setting gives an exact correspondence between the decoding algorithm presented above and decoding algorithm for the qSC in [6], with the only differences being the error probability for a false verification and the running time, as here we may require exhaustively considering all $O(n^{d_c})$ symbol combinations in order to find the right locations for the symbols. Based on this correspondence, we can conclude the following theorem, from [6, Theorem 2].

Theorem 1: For any rate R, with $0 < R < 1$, and a given $\epsilon > 0$, there exists a family of verification-based deletion codes of rate R that correct a $1 - \frac{R}{2} - \frac{\sqrt{4R - 3R^2}}{2} - \epsilon$ fraction of deletions with high probability.

The decoding for such codes can be done in time polynomial in n, assuming that d_c is a constant. The result of Theorem 1, however, is based on capacity achieving codes such as Tornado codes, where d_c grows like $\ln(1/\epsilon)$. Initially, $O(n^{d_c} \log q)$ work must be done to consider all relevant combinations of packets. It is convenient to sort these values, to determine whether any of them lead to verifications, leading to $O(n^{d_c} \log(n^{d_c}) \log q)$ work. After that, whenever the status of a variable node is changed, the corresponding check node must see whether it can send further useful information on to other packets. That is, once a missing value is verified, new values corresponding to combinations involving that value must be checked. The entire decoding time, however, remains $O(n^{d_c} \log(n^{d_c}) \log q)$.

As a perhaps more practical example, applying (1) and (2) to a 3-6 regular LDPC code yields a verification code for deletions of rate $1/2$ that runs in $O(n^6 \log n \log q)$ time and can handle over 13% of the symbols being deleted.

IV. Improving to Near-Capacity Performance

Shokrollahi and Wang [17] have demonstrated that near-capacity performance can be obtained on the qSC by using an improved decoding scheme suggested in [6] along with an appropriate precoding. Given this result, and the correspondence between decoding on the qSC and for deletion channels given above, it is natural to ask whether we can obtain similar near-capacity performance on deletion channels. In this section, we show that in fact this is the case. In order to make this work entirely self-contained, we provide a full argument, using the right-regular codes presented in [16]. Right-regular codes make the decoding algorithm slightly easier to describe, although again irregular degree sequences can be used with small modifications. We note that the results of [17] (using an irregular code construction) could also be used to obtain a similar result, perhaps giving a slight improvement on the dependence of the decoding complexity and error probability on ϵ. Since we have $1/\epsilon$ terms in the exponent, even such improvements would not lead to a practical scheme, and we do not attempt to optimize such terms here.

We modify the verification-based decoding algorithm as follows. Besides applying the previous rules, we consider all sums of combinations of $d_c - 1$ arrived or recovered variable nodes and one check node. In this case, if two check nodes have $d_c - 1$ neighbors that have arrived and share one neighbor that has been deleted, the corresponding sums will match. In this case, we have a match on a proposed value for a deleted node; when this occurs, the value for this node can be recovered and all neighbors of the two check nodes verified.

This modification corresponds to the improved algorithm for the qSC given in [6] and [17]. The argument that no false verification occurs is entirely similar to that of Lemma 1; as long as the variable node and check node values are random, the probability of an incorrect verification is small. (As stated in [6], we also require no cycles of length 4 in the graph representing the LDPC code.) Specifically, there are

$m\binom{n}{d_c-1}$ possible ways of combining a check node and d_c-1 message nodes; the probability any two combinations will have the same value is bounded above by

$$\frac{\left(m\binom{n}{d_c-1}\right)^2}{q}=\frac{n^{O(d_c)}}{q}.$$

Hence, $b=\Omega(d_c\log n)$ bits per packet still ensures that no false verification occurs with high probability. As we show below, using $d_c=\Theta(1/\epsilon^2)$ suffices, so we may require $b=\Omega(\log n/\epsilon^2)$ bits per packet.

From [6], we have for this decoding algorithm

$$a_{j+1}=a_0\lambda(1-\rho(1-b_j)) \tag{3}$$

and

$$b_{j+1}=b_0[\lambda(1-\rho(1-b_j)) + \lambda'(1-\rho(1-b_j))(\rho(1-b_j)-\rho(1-a_j-b_j))]. \tag{4}$$

It is worth explaining the new (4) for the b_j, following the framework of [6]. For a variable node that has not arrived to remain unverified after $j+1$ rounds, one of the following two conditions must hold after j rounds:

1) all neighbors of v other than c have some other neighboring variable node that has not arrived and not been recovered; or
2) all but one neighbor of v other than c have some other neighboring variable node that has not arrived and not been recovered, and the one neighbor that of v for which all neighbors have either arrived or been recovered cannot have all of its neighbors verified.

The first case corresponds to the term $b_0\lambda(1-\rho(1-b_j))$. For the second case, when v has $i-1$ other neighbors, the probability that a specific set of $i-2$ neighbors other than c have at least one variable node neighbor other than v after round j that did not arrive and remains unverified is $(1-\rho(1-b_j))^{i-2}$. Ignoring (temporarily) the last neighboring check node, since there are $i-1$ possible ways of choosing the correct neighbor, we have the term

$$\sum_i \lambda_i(i-1)(1-\rho(1-b_j))^{i-2}=\lambda'(1-\rho(1-b_j))$$

where λ' is the derivative of λ. We multiply this term by $(\rho(1-b_j)-\rho(1-a_j-b_j))$. Here $\rho(1-b_j)$ is the probability that all of the other neighbors of the last check have arrived or been verified, and $\rho(1-a_j-b_j)$ is the probability that all of the other neighbors are verified.

We now apply the insight of [17]; if we precode by using a standard erasure-correcting code on the original message before coding for the deletion channel, we do not need to recover all of the variable nodes, but just a $1-\gamma$ fraction of them for some suitably small γ. Hence, we do not need to show that a_i+b_i converges to 0, but only to some suitably small constant $\gamma=O(\epsilon)$.

First, as noted in [6], we can modify (4) by replacing a_j with a_{j+1}; this corresponds to breaking each round into two subrounds, with variable nodes that having arrived being verified in the first subround, and variable nodes that have been deleted being recovered in the second subround

$$b_{j+1}=b_0[\lambda(1-\rho(1-b_j)) + \lambda'(1-\rho(1-b_j))(\rho(1-b_j)-\rho(1-a_{j+1}-b_j))]. \tag{5}$$

Now substituting the expression for a_{j+1} gives a recursion in terms of b_j

$$b_{j+1}=b_0[\lambda(1-\rho(1-b_j))+\lambda'(1-\rho(1-b_j))(\rho(1-b_j) -\rho(1-a_0\lambda(1-\rho(1-b_j))-b_j))]. \tag{6}$$

Expressing things as a single recursion makes it somewhat easier to show convergence of the b_j to a fixed point close to 0.

We now recall the degree sequences introduced in [16], which were designed to approach capacity for the erasure channel. For $a\geq 2$ and $N\geq 2$, we let $\alpha=1/(a-1)$ and use

$$\rho_a(x)=x^{a-1}$$

and

$$\lambda_{a,N}(x)=\alpha\frac{\sum_{k=1}^{N-1}\binom{\alpha}{k}(-1)^{k+1}x^k}{\alpha-N\binom{\alpha}{N}(-1)^{N+1}}$$

where $\binom{\alpha}{k}$ is the standard fractional binomial coefficient. Here $\lambda_{a,N}(x)$ is meant to be a close approximation of $\lambda_*(x)=1-(1-x)^\alpha$, which satisfies

$$\lambda_*(1-\rho_a(1-x))=x.$$

Indeed, it follows easily from [16] that

$$\lambda'_{a,N}(x)\leq\lambda'_*(x)=\alpha(1-x)^{\alpha-1}$$

and

$$\lambda_{a,N}(1-\rho_a(1-x))\leq\frac{\alpha x}{\beta}$$

where

$$\beta=\alpha-N\binom{a}{N}(-1)^{N+1}.$$

The rate of the code (see [16, Proposition 2 and Theorem 2]) is

$$1-\frac{\alpha-N\binom{\alpha}{N}(-1)^{N+1}}{\alpha-\binom{\alpha}{N}(-1)^{N+1}}\approx 1-\frac{\beta}{\alpha}$$

where the approximation can be made as close as desired by taking N sufficiently large. Hence, β/α can be made arbitrarily close to $1-R$ by choosing N sufficiently large (see [16, Theorem 2]).

Using $\lambda=\lambda_{a,N}$ and $\rho=\rho_a$ in (3) yields

$$a_{j+1}\leq\frac{\alpha(1-b_0)}{\beta}b_j. \tag{7}$$

Similarly, applying these facts to (5) yields

$$\begin{aligned}b_{j+1}&\leq\frac{\alpha b_0}{\beta}b_j \\ &\quad+b_0\alpha\rho_a(1-b_j)^{\alpha-1}(\rho_a(1-b_j)-\rho_a(1-a_{j+1}-b_j)) \\ &=\frac{\alpha b_0}{\beta}b_j+b_0\alpha\rho_a(1-b_j)^\alpha\left(1-\frac{\rho_a(1-a_{j+1}-b_j)}{\rho_a(1-b_j)}\right) \\ &=\frac{\alpha b_0}{\beta}b_j+b_0\alpha(1-b_j)\left(1-\left(1-\frac{a_{j+1}}{1-b_j}\right)^{a-1}\right) \\ &\leq\frac{\alpha b_0}{\beta}b_j+b_0\alpha.\end{aligned}$$

We have terminated the equations using a seemingly very loose bound, but it proves sufficient here. Choose N sufficiently large and $R\geq 1-p-\epsilon/2$, so that $\alpha b_0/\beta=\alpha p/\beta\leq 1-\nu\epsilon$ for a suitable constant ν. Further, choose a sufficiently large so that $\alpha\leq\mu\epsilon^2$ for a suitable constant μ. Then we have that $b_{j+1}\leq b_j$ as long as

$$b_j\geq(1-\nu\epsilon)b_j+b_0\mu\epsilon^2$$

or equivalently as long as

$$b_j \geq \frac{b_0 \mu \epsilon}{\nu}.$$

Hence, by choosing $a = \Theta(1/\epsilon^2)$ and N sufficiently large, we can guarantee that b_j (and hence $a_j + b_j$) does not reach its fixed point until $a_j + b_j$ is at most $\epsilon/2$.

We can therefore state the following theorem.

Theorem 2: Over a q-ary deletion channel with deletion probability p, an erasure precode and an LDPC code with the right-regular degree distribution based on fractional binomial coefficients can together achieve a rate of $(1-\epsilon)(1-p)$. The computational complexity of the decoder is $n^{O(1/\epsilon^2)} \log q$, and the decoding error probability is $n^{O(1/\epsilon^2)}/q$.

Proof: By choosing $a = d_c = O(1/\epsilon^2)$ and N sufficiently large, we can set the rate of the erasure code to be at least $1 - \epsilon/2$ and the rate of initial verification-based code to be at least $(1-p)(1-\epsilon/2)$. The rate of the combined code is then at least

$$\left(1 - \frac{\epsilon}{2}\right)(1-p)\left(1 - \frac{\epsilon}{2}\right) \geq (1-\epsilon)(1-p).$$

The corresponding bounds on the complexity and the decoding error probability follow. □

V. Further Improvements and Variations

A. Reducing the Decoding Time

The running time of verification-based codes for random deletions can be improved by not considering all $O(n^{d_c})$ combinations of d_c symbols. Each received packet can be associated with its expected position in the sent stream of packets, and it will lie within $O(\sqrt{n \log n})$ packets of its expected position with all but polynomially small probability. Taking advantage of this can reduce the decoding time to $O(n^{d_c/2}(\mathrm{polylog}(n)))$.

There appears to be further room for improvements of this type. In particular, one could arrange for the neighbors of a check node to lie within a small neighborhood, rather than be distributed nearly uniformly among the variable nodes. As long as the number of small cycles is kept sufficiently small, the analysis will still hold. If one could arrange for the symbols associated with each check node to be in a block of size $s \ll n$ initially, then by trying $O(\sqrt{n \log n})$ positions for the initial packet as in the preceding paragraph, one should be able to reduce the decoding time to $O\left(\sqrt{n}(\mathrm{polylog}(n))\binom{s}{d_c}\right)$. Similarly, the alphabet size could be correspondingly reduced. It is not clear how small s can be chosen, or whether such an approach can be made practical; this remains a point for future work.

B. Insertion and Transposition Errors

The verification-based decoding scheme is robust against substitution errors or insertions of packet data, as long as such errors or insertions maintain the property that no false verification occurs with high probability. Hence, if random errors are made (so a symbol takes on a uniform incorrect value) or symbols with random values are inserted, as long as q is sufficiently large, the decoding process will still be successful. For example, suppose n random symbols are interleaved among the n encoding symbols before passing through the deletion channel. This gives at most $m\binom{2n}{d_c}$ possible additional combinations involving the extraneous symbols, and thus an additional probability of $m\binom{2n}{d_c}/q$ for a false verification. In this setting, it is still the case that a suitable $b = \Omega(d_c \log n)$ bits per packet ensures that no false verification occurs with high probability. More general error or insertion models can be handled, as long as one can bound the overall error of a false verification step under the model.

The codes are also obviously robust against small amounts of packet-reordering (transpositions). As long as packets associated with every check node are received in the correct order, the decoding algorithms we have described above will function correctly.

In fact, arbitrary reordering can be handled as well with only slightly more work, as we now sketch. In the case of arbitrary reordering, when all the neighbors of a check node have arrived or been recovered, the check node can verify these values, but because the check node does not know the proper order, it can only provide a list of the possible values to a variable node. If a variable node receives such a list from two (or more) distinct check nodes, with high probability only a single value will occur on both lists (assuming no cycles of length four in the code graph), providing a variable node with the appropriate unique value. Let us think of a variable node that has determined its unique value as *fully verified*, and a variable node that knows its value as being in a list of up to d_c values as being *partially verified*. We allow a deleted variable node to be recovered if a neighboring check node has all of its other neighboring variable nodes fully verified.

Now, as previously, we consider the additional modification where all sums of combinations of $d_c - 1$ arrived or recovered variable nodes and one check node yield proposed values for an unrecovered neighbor of a variable node. If two check nodes with one common unrecovered variable node neighbor offer the same proposed value, so that we have a match on a proposed value for a deleted node, then the value for this node can be recovered. Note that in this case the value for this node is in fact fully verified, since we know exactly which node takes on the prescribed value. The other neighbors of the check nodes will become either partially or fully verified.

We claim that with this understanding, we again have the equation

$$b_{j+1} = b_0[\lambda(1 - \rho(1 - b_j)) + \lambda'(1 - \rho(1 - b_j))(\rho(1 - b_j) - \rho(1 - a_j - b_j))] \quad (8)$$

where a_j is the fraction of symbols that have arrived but have not been fully verified. That is, we again have that for a variable node that has not arrived to send a message that it is still unverified after $j+1$ rounds to a neighboring check node c, one of the following two conditions must hold after j rounds:

1) all neighbors of v other than c have some other neighboring variable node that has not arrived and not been recovered; or
2) all but one neighbor of v other than c have some other neighboring variable node that has not arrived and has not been recovered, and the one neighbor that of v for which all neighbors have either arrived or been recovered cannot have all of its neighbors fully verified.

From (8), it is straightforward to repeat the argument of Section IV, showing that Theorem 2 holds even if we allow arbitrary reordering of the undeleted symbols.

VI. Conclusion

We have demonstrated a fundamental correspondence between verification-based decoding on the qSC and the deletion channel, and used this correspondence to modify recent analysis for near-capacity codes for the qSC to obtain similar codes for the deletion channel. Variations on these codes can handle arbitrary reordering of the arriving symbols as well. Although the resulting decoding algorithms are polynomial in

n, they are exponential in d_c, which grows large for near-capacity performance. While the codes we present here could possibly be improved in various ways, the key goal for future work would be to remove the exponential dependence on d_c; ideally, we would aim for a code whose running time was polynomial in both n and d_c. Our hope is that this framework can either directly be improved in such a way to yield effective algorithms, or will give insight into alternative efficient algorithms.

ACKNOWLEDGMENT

The author would like to thank Amin Shokrollahi and Wei Wang for providing the submitted version of [17], which was instrumental for developing the arguments of this correspondence. The author also thanks the anonymous reviewers for helpful comments.

REFERENCES

[1] J. M. Chung and J. J. Metzner, "Theoretical analysis of the error correction performance of majority-logic-like vector symbol codes," *IEEE Trans. Commun.*, vol. 49, no. 6, pp. 979–987, Jun. 2001.

[2] M. C. Davey and D. J. C. MacKay, "Reliable communication over channels with insertions, deletions, and substitutions," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 671–686, Feb.. 2001.

[3] S. Diggavi and M. Grossglauer, "On transmission over deletion channels," in *Proc. 39th Annu. Allerton Conference on Communication, Control and Computing*, Monticello, IL, 2001, pp. 573–582.

[4] E. Drinea and M. Mitzenmacher, "On lower bounds for the capacity of deletion channels," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4648–4657, Oct. 2006.

[5] ——, "Improved lower bounds for i.i.d. deletion and insertion channels," in *Proc. 42nd Annu. Allerton Conf. Communication, Control and Computing*, Monticello, IL, 2004, submitted for publication.

[6] M. Luby and M. Mitzenmacher, "Verification-based decoding for packet-based low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 120–127, Jan. 2005.

[7] M. Luby, M. Mitzenmacher, and M. A. Shokrollahi, "Analysis of random processes via and-or tree evaluation," in *Proc. 9th Annu. ACM-SIAM Symp. Discrete Algorithms*, San Francisco, CA, Jan. 1998, pp. 364–373.

[8] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. Spielman, "Efficient erasure correcting codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 569–584, Feb. 2001.

[9] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 585–598, Feb. 2001.

[10] J. J. Matzner, "Majority-Logic-like decoding of vector symbols," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1227–1230, Oct. 1996.

[11] J. J. Metzner, "Majority-logic-like vector symbol decoding with alternative symbols," *IEEE Trans. Commun.*, vol. 48, no. 12, pp. 2005–2013, Dec. 2000.

[12] M. Mitzenmacher, "Verification codes for deletion channels," in *Proc. 2003 IEEE Int. Symp. Information Theory*, Yokohama, Japan, 2003, p. 217.

[13] T. Richardson and T. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[14] ——, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.

[15] L. Schulman and D. Zuckerman, "Asymptotically good codes correcting insertions, deletions, and transpositions," *IEEE Trans. Inf. Theory*, vol. 45, no. 7, pp. 2552–2557, Nov. 1999.

[16] A. Shokrollahi, "New sequences of linear time erasure codes approaching the channel capacity," in *Proc. 13th Int. Symp. Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes (Lecture Notes in Computer Science)*, M. Fossorier, H. Imai, S. Lin, and A. Poli, Eds. Berlin, Germany: Springer-Verlag, 1999, vol. 1719, pp. 65–76.

[17] A. Shokrollahi and W. Wang, "Low-density parity-check codes with rates very close to the capacity of the q-ary symmetric channel for large q," in *Proc. IEEE Int. Symp. Information Theory*, Chicago, IL, Jun./Jul. 2004, p. 275.

[18] N. J. A. Sloane, "On single-deletion-correcting codes," in *Codes and Designs (Ray-Chaudhuri Festschrift)*, K. T. Arasu and A. Seress, Eds. Berlin, Germany: de Guyter, 2002, pp. 273–291.

An Upper Bound on the Minimum Distance of Serially Concatenated Convolutional Codes

Alberto Perotti, *Member, IEEE*, and Sergio Benedetto, *Fellow, IEEE*

***Abstract*—This correspondence presents an upper bound on the minimum distance of serially concatenated codes with interleaver where the inner code is a systematic recursive convolutional encoder and the outer code is any convolutional encoder. The resulting expression shows that the minimum distance of the concatenated code cannot grow more than $O(K^{1-1/d_f^{(O)}})$, where K is the information word length, and $d_f^{(O)}$ is the free distance of the outer code. The obtained upper bound is shown to agree with and, in some cases, improve over previously known results.**

***Index Terms*—Combinatorial, concatenated, convolutional, minimum distance, performance bounds, serial.**

I. INTRODUCTION

Serially concatenated convolutional codes with interleavers [1] are known to perform better than parallel turbo codes in the error floor region [2]. However, when heavy puncturing is applied to obtain higher code rates, it is not uncommon to observe a significant error floor. In order to estimate the error probability in such a region, the minimum distance is a crucial parameter. In this correspondence, a method to obtain an upper bound on the minimum distance of serially concatenated convolutional codes (SCCCs) is described. The present result applies to serial concatenations where the outer code is any convolutional encoder and the inner code is generated by a rate k_0/n_0 systematic recursive convolutional encoder.

Results on the minimum distance of serially concatenated codes have been presented in [3] and [4]. Both papers show that the minimum distance grows with the block length as $O(K^x)$: in [3] the exponent x depends on the minimum distance of the outer code, while, in [4], it depends on the memory and rate of the outer encoder. Our result, although similar to the cited ones, improves over [4], where a broader class of constituent encoders is considered. Moreover, it is coherent with [3]. The method used here to derive the upper bound has been inspired by that used in [5] and [6] to obtain an upper bound on the minimum distance of parallel turbo codes.

Section II summarizes the previously known results and Section III describes the coding scheme used as a reference in order to derive the bound. In Section IV, the upper bound is derived. Finally, in Section V, the obtained upper bound is applied to different coding schemes and compared with the previously known results.

We will interchangeably use the following notations for symbol vectors:

$$\boldsymbol{v} = \{v_i, i = 0, \ldots, N-1\} \quad \longleftrightarrow \quad v(D) = \sum_{i=0}^{N-1} v_i D^i.$$

Manuscript received February 16, 2004; revised June 22, 2006. This work has been funded by Regione Piemonte under Research Project E 4. The material in this correspondence was presented in part at the IEEE International Symposium on Information Theory, Chicago, IL, June/July 2004.

The authors are with the Center for Multimedia Radio Communications at Politecnico di Torino, I-10129 Torino, Italy (e-mail: alberto.perotti@polito.it; sergio.benedetto@polito.it).

Communicated by G. Zémor, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2006.885447