Codes Correcting a Burst of Deletions or Insertions

Clayton Schoeny[1], Antonia Wachter-Zeh[2], Ryan Gabrys[3], Eitan Yaakobi[2]

[1]University of California, Los Angeles, CA, USA
[2]Computer Science Department, Technion—Israel Institute of Technology, Haifa, Israel
[3]Spawar Systems Center San Diego, CA, USA
emails: *cschoeny@ucla.edu, antonia@cs.technion.ac.il, ryan.gabrys@gmail.com, yaakobi@cs.technion.ac.il*

***Abstract*—This paper studies codes that correct bursts of deletions. Namely, a code will be called a b-*burst-correcting code* if it can correct a deletion of any b consecutive bits. While the lower bound on the redundancy of such codes was shown by Levenshtein to be asymptotically $\log(n) + b - 1$, the redundancy of the best code construction by Cheng *et al.* is $b(\log(n/b + 1))$. In this paper we close on this gap and provide codes with redundancy at most $\log(n) + (b - 1)\log(\log(n)) + b - \log(b)$.**

We also extend the burst deletion model to two more cases: 1. a deletion burst of at most b consecutive bits and 2. a deletion burst of size at most b (not necessarily consecutive). We extend our code construction for the first case and study the second case for $b = 3, 4$. The equivalent models for insertions are also studied and are shown to be equivalent to correcting the corresponding burst of deletions.

I. Introduction

In communication and storage systems, symbols are often inserted or deleted due to synchronization errors. These errors can be caused by a variety of disturbances such as timing defects or packet-loss. Constructing codes that correct insertions or deletions is a notoriously challenging problem since a relatively small number of edits can cause the transmitted and received sequences to be vastly different in terms of the Hamming metric.

For disconnected, intermittent, and low-bandwidth environments, the problem of recovering from symbol insertion/deletion errors becomes exacerbated [5]. From the perspective of the communication systems, these errors manifest themselves in bursts where the errors tend to cluster together. Our goal in this work is the study of codes capable of correcting from bursts of insertion/deletion errors. Such codes have many applications pertaining to the synchronization of data in wireless sensor networks and satellite communication devices [7].

In the 1960s, Varshamov, Tenengolts, and Levenshtein laid the foundations for codes capable of correcting insertions and deletions. In 1965, Varshamov and Tenengolts created a class of codes (now known as VT-codes) that is capable of correcting asymmetric errors on the Z-channel [14], [13]. Shortly thereafter, Levenshtein proved that these codes can also be used to correct a single insertion or deletion [9] and he constructed a class of codes that can correct two adjacent insertions or deletions [10].

The main goal of this work is to study codes that correct a *burst of deletions* which refers to the deletion of a fixed number of consecutive bits. A code will be called a *b-burst-deletion-correcting code* if it can correct any deletion burst of size b. For example, the codes studied by Levenshtein in [10] are two-burst-deletion-correcting codes.

Establishing tight upper bounds on the cardinality of burst-deletion-correcting codes is a challenging task since the burst deletion balls are not all of the same size. In [9], Levenshtein derived an asymptotic upper bound on the maximal cardinality of a b-burst-deletion-correcting code, given by $\frac{2^{n-b+1}}{n}$. Therefore, the minimum redundancy of such a code should be approximately $\log(n) + b - 1$.

On the other hand, the best construction of b-burst-correcting codes, that we are aware of, is Construction 1 by Cheng *et al.* [3]. The redundancy of this construction is $b(\log(n/b + 1))$ and therefore there is still a significant gap between the lower bound on the redundancy and the redundancy of this construction. One of our main results in this paper is showing how to improve the construction from [3] and deriving codes whose redundancy is approximately

$$\log(n) + (b - 1)\log(\log(n)) + b - \log(b), \quad (1)$$

which is larger than the lower bound on the redundancy by roughly $(b - 1)\log(\log(n))$.

Our contributions are organized as follows. In Section II we define the common terms used throughout the paper and we detail the previous results that will be used as a comparison. In Section III, we construct b-burst-correcting codes with the redundancy stated in (1). In Sections IV and V, we present code constructions that correct a deletion burst of size at most b and codes that correct a non-consecutive burst of size at most three and four, respectively. Due to space restrictions, some proofs are omitted and can be found in the long version [11].

II. Preliminaries and Previous Work

A. Notations and Definitions

Let $\mathbb{F}_q$ be a finite field of order q, where q is a power of a prime and let $\mathbb{F}_q^n$ denote the set of all vectors (sequences) of length n over $\mathbb{F}_q$. Throughout this paper, we restrict ourselves to binary vectors, i.e., $q = 2$. A *subsequence* of a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is formed by taking a subset of the symbols of $\mathbf{x}$ and aligning them without changing their order. Hence, any vector $\mathbf{y} = (x_{i_1}, x_{i_2}, \ldots, x_{i_m})$ is a subsequence of $\mathbf{x}$ if $1 \le i_1 < i_2 < \cdots < i_m \le n$, and in this case we say that $n - m$ *deletions* occurred in the vector $\mathbf{x}$ and $\mathbf{y}$ is the result.

A *run* of length r of a sequence $\mathbf{x}$ is a subvector of $\mathbf{x}$ such that $x_i = x_{i+1} = \cdots = x_{i+r-1}$, in which $x_{i-1} \ne x_i$ if $i > 1$, and if $i + r - 1 < n$, then $x_{i+r-1} \ne x_{i+r}$. We denote by $r(\mathbf{x})$ the number of runs of a sequence $\mathbf{x} \in \mathbb{F}_2^n$.

We refer to a *deletion burst of size b* when exactly b consecutive deletions have occurred, i.e., from $\mathbf{x}$, we obtain a subsequence $(x_1, \ldots, x_i, x_{i+b+1}, \ldots, x_n) \in \mathbb{F}_2^{n-b}$. Similarly, a *deletion burst of size at most b* results in a subsequence $(x_1, \ldots, x_i, x_{i+a+1}, \ldots, x_n) \in \mathbb{F}_2^{n-a}$, for some $a \le b$. More generally, a *non-consecutive deletion burst of size at most b* means that within b consecutive symbols of $\mathbf{x}$, there were $a \le b$ deletions, i.e., we obtain a subsequence $(x_1, \ldots, x_i, x_{i+i_1}, x_{i+i_2}, \ldots, x_{i+i_{b-a}}, x_{i+b+1}, \ldots, x_n) \in \mathbb{F}_2^{n-a}$, for some $a \le b$, where $1 \le i_1 < i_2 < \cdots < i_{b-a} \le b$.

The *b-burst-deletion ball* of a vector $\mathbf{x} \in \mathbb{F}_2^n$, is denoted by $D_b(\mathbf{x})$, and is defined to be the set of subsequences of $\mathbf{x}$ of length $n-b$ obtained by the deletion of a burst of size b. Similarly, $D_{\le b}(\mathbf{x})$ is defined to be the set of subsequences of $\mathbf{x}$ obtained from a deletion burst of size at most b.

A *b-burst-deletion-correcting code* $\mathcal{C}$ is a set of codewords in $\mathbb{F}_2^n$ such that there are no two codewords in $\mathcal{C}$ where deletion bursts of size b result in the same word of length $n-b$. That is, for every $\mathbf{x}, \mathbf{y} \in \mathcal{C}$, $D_b(\mathbf{x}) \cap D_b(\mathbf{y}) = \emptyset$.

Similarly, we will use the following notations for bursts of insertions, namely: *insertions burst of size (at most) b*, *b-burst-insertion ball*, and *b-burst-insertion-correcting code*. Usually correcting deletions and insertions is an equivalent task and indeed this is not an exception in this work, i.e., the following equivalence holds.

Theorem 1 *A code $\mathcal{C}$ is a b-burst-deletion-correcting code if and only if it is a b-burst-insertion-correcting code.*

Similar statements hold for consecutive and non-consecutive deletion bursts of size at most b. Thus, in the remainder of the paper, whenever we refer to bursts of deletions, all the results hold equivalently for bursts of insertions as well.

Throughout this paper, we let b be a fixed integer which divides n. For a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, we define the following $b \times \frac{n}{b}$ array:

$$A_b(\mathbf{x}) = \begin{bmatrix} x_1 & x_{b+1} & \cdots & x_{n-b+1} \\ x_2 & x_{b+2} & \cdots & x_{n-b+2} \\ \vdots & \vdots & \ddots & \vdots \\ x_b & x_{2b} & \cdots & x_n \end{bmatrix},$$

and for $1 \le i \le b$ we denote by $A_b(\mathbf{x})_i$ the ith row of $A_b(\mathbf{x})$.

For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$, the *Levenshtein distance* $d_L(\mathbf{x}, \mathbf{y})$ is the minimum number of insertions and deletions that is necessary to change $\mathbf{x}$ into $\mathbf{y}$. Unless stated otherwise, all logarithms in this paper are taken according to base 2.

B. Previous Work

In this subsection, we recall known results on codes that correct deletions and insertions. These results will be used later as a comparison reference for our constructions.

1) Single-deletion-correcting codes: The Varshamov-Tenengolts codes [14] are a family of single-deletion-correcting codes (see also Sloane's survey in [12]).

Definition 1 *For $0 \le a \le n$, the Varshamov-Tenengolts (VT) code $VT_a(n)$ is defined to be the following set:*

$$VT_a(n) \triangleq \left\{ \boldsymbol{x} = (x_1, \ldots, x_n) \,:\, \sum_{i=1}^{n} i x_i \equiv a \mod (n+1) \right\}.$$

The redundancy of the $VT_0(n)$ code is at most $\log(n+1)$ (see [12, Eq. (10)]). For all n, the union of all VT-codes forms a partition of the space $\mathbb{F}_2^n$, that is $\cup_{a=0}^{n} VT_a(n) = \mathbb{F}_2^n$.

2) b-burst-deletion-correcting codes: Construction 1 from [3] is the set of all codewords $\mathbf{c}$ such that each row of $A_b(\mathbf{c})$ is a codeword of the code $VT_0(\frac{n}{b})$. A deletion burst of size b deletes exactly one symbol in each row of $A_b(\mathbf{c})$ which can then be corrected by the VT-code. The redundancy of this construction is $b\left(\log\left(\frac{n}{b}+1\right)\right)$. The other two constructions from [3] have redundancy at least $\frac{n}{b}$.

3) Correcting a deletion burst of size at most b: To the best of our knowledge, the only known construction to correct a burst of size at most b is the one from [1] which has redundancy at least $\frac{n}{b}$.

4) Correcting b deletions (not a burst): The construction from [2] can correct b deletions at arbitrary positions (not in a burst) in a vector of length n and has redundancy $c \cdot b^2 \log(b) \log(n)$, for some constant c.

C. An Upper Bound on the Code Size

For large n, Levenshtein [10] derived an asymptotic upper bound on the maximal cardinality of a binary b-burst-deletion-correcting code $\mathcal{C}$ of length n. This bound states that for n large enough, an upper bound on the cardinality of the code $\mathcal{C}$ is approximately $\frac{2^{n-b+1}}{n}$, and hence its redundancy is at least roughly $\log(n) + b - 1$.

Based on the method by Kulkarni and Kiyavash in [8], we can derive the following explicit non-asymptotic upper bound.

Theorem 2 *Any binary b-burst-deletion correcting code $\mathcal{C}$ of length n satisfies*

$$|\mathcal{C}| \le \frac{2^{n-b+1} - 2^b}{n - 2b + 1}.$$

Notice that for $b = 1$ our upper bound in Theorem 2 equals to the upper bound in [8, Theorem 3.1] for single-deletion-correcting codes. Furthermore, for n large enough our upper bound coincides with the asymptotic bound from [10]. We conclude that the redundancy of a b-burst-deletion-correcting code is lower bounded by the following value

$$\log(n - 2b + 1) - \log(2^{-b+1} - 2^{b-n}) \approx \log(n) + b - 1.$$

III. b-Burst-Deletion-Correcting Codes

The main goal of this section is to provide a construction of b-burst-deletion-correcting codes, whose redundancy is better than the state of the art results we reviewed in Section II-B. We will first explain the main ideas of the construction and will then provide the specific details of all components in the construction.

A deletion burst of size b in $\mathbf{x}$ deletes exactly one bit from each row of the array $A_b(\mathbf{x})$. That is, if a codeword $\mathbf{x}$ is transmitted, then the $b \times (\frac{n}{b} - 1)$ array representation of the received vector $\mathbf{y}$ has the following structure

$$A_b(\mathbf{y}) = \begin{bmatrix} y_1 & y_{b+1} & \cdots & y_{n-2b+1} \\ y_2 & y_{b+2} & \cdots & y_{n-2b+2} \\ \vdots & \vdots & \ddots & \vdots \\ y_b & y_{2b} & \cdots & y_{n-b} \end{bmatrix}.$$

Each row is received by a single deletion of the corresponding row in $A_b(\mathbf{x})$ [3], i.e., $A_b(\mathbf{y})_i \in D_1(A_b(\mathbf{x})_i)$, $\forall 1 \le i \le b$.

Since the channel deletes a burst of b bits, the deletions can span at most two columns of the codeword array. Therefore, information about the position of a deletion in a single row provides information about the positions of the deletions in the remaining rows. However, note that deletion-correcting codes are not always able to determine the exact position of the deleted bit. For example, assume the all-zero codeword was transmitted and a single deletion of one of the bits has occurred. Even if the decoder can successfully decode the received vector, it is not possible to know the position of the deleted bit since it could be any of the bits.

In order to take advantage of the correlation between the positions of the deleted bits in different rows and overcome the difficulty that deletion-correcting codes cannot always provide the location of the deleted bits, we construct a single-deletion-correcting code with the following special property.

The receiver of this code can correct the single deletion and determine its location within a certain predetermined range of consecutive positions. This code will be used to encode the first row of the codeword array and will provide partial information on the position of the deletions for the remaining $b-1$ rows. In these rows, we use a different code that will take advantage of this positional information.

The following is a high-level outline of the proposed codeword array construction:

- The first row in the array is encoded as a VT-code in which we restrict the longest run of 0's or 1's to be at most $\log(2n)$.
- Each of the remaining $(b-1)$ rows is encoded by a modified version of the VT-code, called a *shifted VT* (*SVT*)*-code*. This code is able to correct a single deletion in each row once the position where the deletion occurred is known to within $\log(2n)+1$ consecutive positions.

A. Run-length Limited (RLL) VT-Codes

We want to limit the length of the longest run in the first row of the codeword array. A length-n binary vector is said to satisfy the (d,k) *Run Length Limited (RLL)* constraint, denoted by $RLL_n(d,k)$, if between any two consecutive 1's there are at least d 0's and at most k 0's [6]. Since we are concerned with runs of 0's or 1's, we will state our constraints on the longest runs of 0's and 1's. Note that the maximum rate of codes which satisfy the (d,k) RLL constraint for fixed d and k is less than 1. To achieve codes with asymptotic rate 1, the restriction on the longest run is a function of the length n.

Definition 2 *A length-n binary vector $\boldsymbol{x}$ is said to satisfy the **f(n)-RLL(n)** constraint, and is called an **f(n)-RLL(n)** vector, if the length of each run of 0's or 1's in $\boldsymbol{x}$ is at most $f(n)$.*

A set of $f(n)$-RLL(n) vectors is called an *$f(n)$-RLL(n) code*, and the set of all $f(n)$-RLL(n) vectors is denoted by $S_n(f(n))$. The *capacity* of the $f(n)$-RLL(n) constraint is

$$C(f(n)) = \lim_{n\to\infty} \frac{\log(|S_n(f(n))|)}{n},$$

and for the case in which the capacity is 1, we define also the *redundancy* of the $f(n)$-RLL(n) constraint to be

$$r(f(n)) = n - \log(|S_n(f(n))|).$$

Lemma 1 *The redundancy of the $\log(2n)$-RLL(n) constraint is upper bounded by 1 for all n.*

Recall that our goal was to have the vector stored in the first row be a codeword in a VT-code so it can correct a single deletion and also limit its longest run. Hence we define a family of codes which satisfy these two requirements, by intersecting all VT-codes with the set $S_n(f(n))$.

Definition 3 *Let a,n be two positive integers where $0 \le a \le n$. The $VT_{a,f(n)}(n)$ code is defined to be the intersection between $VT_a(n)$ and $S_n(f(n))$. That is,*

$$VT_{a,f(n)}(n) = \{\boldsymbol{x} \ : \ \boldsymbol{x} \in VT_a(n), \boldsymbol{x} \in S_n(f(n))\}.$$

Corollary 1 follows from the pigeonhole argument.

Corollary 1 *For all n, there exists $0 \le a \le n$ such that the redundancy of $VT_{a,\log(2n)}(n)$ is at most $\log(n+1)+1$.*

B. Shifted VT-Codes

Let us now focus on the remaining $(b-1)$ rows of our codeword array. Decoding the first row in the received array allows the decoder to determine the locations of the deletions of the remaining rows up to a set of consecutive positions. We define a new class of codes with this positional knowledge of deletions in mind.

Definition 4 *A **P-bounded single-deletion-correcting code** is a code in which the decoder can correct a single deletion given knowledge of the location of the deleted bit to within P consecutive positions.*

We create a new code, called a *shifted VT* (*SVT*)*-code*, which is a variant of the VT-code and is able to take advantage of the positional information as defined in Definition 4.

Construction 1 *For $0 \le c < P$ and $d \in \{0,1\}$, let the shifted Varshamov-Tenengolts code $SVT_{c,d}(n,P)$ be:*

$$SVT_{c,d}(n,P) \triangleq \left\{\boldsymbol{x} : \sum_{i=1}^{n} ix_i \equiv c (\bmod P), \sum_{i=1}^{n} x_i \equiv d (\bmod 2)\right\}.$$

Other modifications of the VT-code have previously been proposed [4]. The next lemma proves the correctness of this construction and provides a lower bound on the cardinality of these codes.

Lemma 2 *For all $0 \le c < P$ and $d \in \{0,1\}$, the $SVT_{c,d}(n,P)$-code (as defined in Construction 1) is a P-bounded single-deletion-correcting code, and there exist $0 \le c < P$ and $d \in \{0,1\}$ such that the redundancy of the $SVT_{c,d}(n,P)$-code is at most $\log(P)+1$ bits.*

Proof: In order to prove that the $SVT_{c,d}(n,P)$-code is a P-bounded single-deletion-correcting code, it is sufficient to show that there are no two codewords $\mathbf{x},\mathbf{y} \in SVT_{c,d}(n,P)$ that have a common subvector of length $n-1$ where the locations of the deletions are within P positions.

Assume in the contrary that there exist two different codewords $\mathbf{x},\mathbf{y} \in SVT_{c,d}(n,P)$, where there exist $1 \le k,\ell \le n$, where $|\ell - k| < P$, such that $\mathbf{z} = \mathbf{x}_{[n]\setminus\{k\}} = \mathbf{y}_{[n]\setminus\{\ell\}}$, and assume that $k < \ell$. Since $\mathbf{x},\mathbf{y} \in SVT_{c,d}(n,P)$, we can summarize these assumptions in the following three properties:

1) $\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} y_i \equiv 0(\bmod 2)$.
2) $\sum_{i=1}^{n} ix_i - \sum_{i=1}^{n} iy_i \equiv 0(\bmod P)$.
3) $\ell - k < P$.

According to these assumptions and since $\mathbf{x}_{[n]\setminus\{k\}} = \mathbf{y}_{[n]\setminus\{\ell\}}$, it is evident that k is the smallest index for which $x_k \neq y_k$, and ℓ is the largest index for which $x_\ell \neq y_\ell$. Additionally, from the first property $\mathbf{x}$ and $\mathbf{y}$ have the same parity and thus $x_k = y_\ell$. Outside of the indices k and ℓ, $\mathbf{x}$ and $\mathbf{y}$ are identical, while inside they are shifted by one position:

$$\begin{aligned} x_i &= y_i \quad \text{for } i < k \text{ and } i > \ell, \\ x_i &= y_{i-1} \quad \text{for } k < i \le \ell. \end{aligned}$$

We consider two scenarios: $x_k = y_\ell = 0$ or $x_k = y_\ell = 1$. First assume that $x_k = y_\ell = 0$, and in this case we get that

$$\sum_{i=1}^{n} ix_i - \sum_{i=1}^{n} iy_i = \sum_{i=k}^{\ell} ix_i - \sum_{i=k}^{\ell} iy_i = \sum_{i=k+1}^{\ell} ix_i - \sum_{i=k}^{\ell-1} iy_i$$
$$= \sum_{i=k+1}^{\ell} iy_{i-1} - \sum_{i=k}^{\ell-1} iy_i = \sum_{i=k}^{\ell-1} (i+1)y_i - \sum_{i=k}^{\ell-1} iy_i = \sum_{i=k}^{\ell-1} y_i.$$

The sum $\sum_{i=k}^{\ell-1} y_i$ cannot be equal to zero or else we will get that $\mathbf{x} = \mathbf{y}$, and hence

$$0 < \sum_{i=1}^{n} ix_i - \sum_{i=1}^{n} iy_i = \sum_{i=k}^{\ell-1} y_i \leq \ell - k < P,$$

in contradiction to the second property.

A similar contraction can be shown for $x_k = y_\ell = 1$. Thus, the three properties cannot all be true, and the $SVT_{c,d}(n, P)$-code is a P-bounded single-deletion-correcting code. ■

There are two major differences between the SVT-codes and the usual VT-codes. First, the SVT-codes restrict the overall parity of the codewords. This parity constraint costs an additional redundancy bit, but it allows us to determine whether the deleted bit was a 0 or a 1. Second, in the VT-code, the weights assigned to each element in the vector are $1, 2, \ldots, n$; on the other hand, in the SVT-code, these weights can be interpreted as repeatedly cycling through $1, 2, \ldots, P-1, 0$ (due to the $\bmod P$ operation). Because of these differences, a VT-code requires roughly $\log(n+1)$ redundancy bits while a SVT-code requires approximately only $\log(P) + 1$ redundancy bits.

C. Code Construction

We are now ready to construct b-burst-deletion-correcting codes by combining the ideas from the previous two subsections into a single code.

Construction 2 *Let $\mathcal{C}_1$ be a $VT_{a,\log(2n/b)}(n/b)$ code for some $0 \leq a \leq n/b$ and let $\mathcal{C}_2$ be a shifted VT-code $SVT_{c,d}(n/b, \log(n/b)+2)$ for $0 \leq c < n/b+2$ and $d \in \{0, 1\}$. The code $\mathcal{C}$ is constructed as follows*

$$\mathcal{C} \triangleq \{\boldsymbol{x} : A_b(\boldsymbol{x})_1 \in \mathcal{C}_1, A_b(\boldsymbol{x})_i \in \mathcal{C}_2, \text{ for } 2 \leq i \leq b\}.$$

Theorem 3 *The code $\mathcal{C}$ from Construction 2 is a b-burst-deletion-correcting code.*

Proof: Assume $\mathbf{x} \in \mathcal{C}$ is the transmitted vector and $\mathbf{y} \in D_b(\mathbf{x})$ is the received vector. In the $b \times (n/b - 1)$ array $A_b(\mathbf{y})$, every row is therefore received by a single deletion of the corresponding row in $A_b(\mathbf{x})$.

Since the first row of $A_b(\mathbf{x})_1$ belongs to a $VT_{a,\log(2n/b)}(n/b)$ code, the decoder of this code can successfully decode and insert the deleted bit in the first row of $A_b(\mathbf{y})_1$. Furthermore, since every run in $A_b(\mathbf{x})_1$ consists of at most $\log(2n/b)$ bits, the locations of the deleted bits in the remaining rows are known within $\log(n/b) + 2$ consecutive positions. Finally, the remaining $b - 1$ rows decode their deleted bit since they belong to a shifted VT-code $SVT_{c,d}(n/b, \log(n/b) + 2)$ (Lemma 2). ■

To conclude this discussion, the following Corollary summarizes the result presented in this section.

Corollary 2 *For sufficiently large n, there exists a b-burst-deletion-correcting code whose number of redundancy bits is at most*

$$\log(n) + (b-1)\log(\log(n)) + b - \log(b).$$

IV. Correcting a Burst of Length at most b (consecutively)

In this section, we consider the problem of correcting a burst of consecutive deletions of length at most b. As defined in Section II, a code capable of correcting a burst of at most b consecutive deletions needs to be able to correct any burst of size a for $a \leq b$. The case $b = 2$ was already solved by Levenshtein with a construction that corrects a single deletion or a deletion of two adjacent bits [10]. The redundancy of this code, denoted by $\mathcal{C}_L(n)$, is at most $1 + \log(n)$ bits. Hence this code asymptotically achieves the upper bound for correcting a burst of exactly 2 deletions. Our strategy for correcting a burst of length *at most* b is to construct a code from the intersection of the code $\mathcal{C}_L(n)$ with the codes that correct a burst of length *exactly* i, for $3 \leq i \leq b$. We refer to each i as a *level* and in each level we will have a set of codes that form a partition of the space. Thus, our overall code will be the largest intersection of the codes at each level.

We therefore build upon the codes from Section III and leverage them as codes in each level. However, since the codes from Section III do not provide a partition of the space we will have to make one additional modification in their construction so it will be possible to intersect the codes in each level and get a code which corrects a burst of size at most b. Recall that in our code from Construction 2 we needed the first row in our codeword array, $A_b(\mathbf{x})_1$, to be run-length limited so that the remaining rows could effectively use the SVT-code. Similarly, in order to correct at most b consecutive deletions we want the first row of each level's codeword array to be an $RLL(N_b)$-vector, where $N_b = \lceil \log(n \log(b)) \rceil + 1$. In other words, $A_i(\mathbf{x})_1$ will satisfy the N_b-RLL$(\frac{n}{i})$ constraint for $3 \leq i \leq b$. We add the term *universal* to signify that an RLL constraint on a vector refers to the RLL constraint on the first row of each level.

Definition 5 *A length-n binary vector $\boldsymbol{x}$ is said to satisfy the **f(n)-URLL(n,b)** constraint, and is called an **f(n)-URLL(n,b)** vector, if the length of each run of 0's or 1's in $A_i(\boldsymbol{x})_1$ for $3 \leq i \leq b$, is not greater than $f(n)$. Additionally, the set of all **f(n)-URLL(n,b)** vectors is denoted by $U_{n,b}(f(n))$.*

The *redundancy* of the $f(n)$-URLL(n, b) constraint is defined as

$$r_U(f(n)) = n - \log(|U_{n,b}(f(n))|).$$

It can be shown that $r_U(N_b) \leq \log(\log(b))$. That is, the following lemma holds.

Lemma 3 *The redundancy of the N_b-URLL(n,b) constraint is upper bounded by $\log(\log(b))$ bits:*

$$r_U(N_b) = \log(\log(b)).$$

In addition to limiting the longest run in the first row of every level, each vector $A_i(\mathbf{x})_1$ should be able to correct a single deletion. We define the following family of codes.

Construction 3 *Let n be a positive integer and $\boldsymbol{a} = a_3, \ldots, a_b$ a vector of non-negative integers such that $0 \leq a_i \leq n/i$ for $3 \leq i \leq b$. The code $\overline{VT}_{\boldsymbol{a},f(n)}(n)$ code is defined as follows:*

$$\overline{VT}_{\boldsymbol{a},f(n)}(n) \triangleq \left\{\boldsymbol{x} \;:\; A_i(\boldsymbol{x})_1 \in VT_{a_i}\left(\frac{n}{i}\right), 3 \leq i \leq b, \right.$$
$$\left. \boldsymbol{x} \in U_{n,b}(f(n))\right\}.$$

The next corollary provides an upper bound on the redundancy of the codes from Construction 3.

Corollary 3 *For all n, there exists a vector $\boldsymbol{a} = (a_3, \ldots, a_b)$ such that the redundancy of the code $\overline{VT}_{\boldsymbol{a},N_b}(n)$ is at most $(b-2)\log(n) + \log(\log(b))$ bits.*

With the universal RLL-constraint in place, we can use the SVT-codes defined in Section III for each of the remaining rows in each level.

Construction 4 *Let $\mathcal{C}_L(n)$ be the code from [10], $\mathcal{C}_1$ be the code $\overline{VT}_{\boldsymbol{a},N_b}(n)$ for some vector $\boldsymbol{a}$, and for $3 \le i \le b$ let $\mathcal{C}_{2,i}$ be a shifted VT-code $SVT_{c_i,d_i}(n/i, N_b+1)$ for $0 \le c_i \le n/i$ and $d_i \in \{0,1\}$. The code $\mathcal{C}$ is constructed as follows*

$$\mathcal{C} \triangleq \{\boldsymbol{x} : \boldsymbol{x} \in \mathcal{C}_L(n), \boldsymbol{x} \in \mathcal{C}_1, \\ A_i(\boldsymbol{x})_j \in \mathcal{C}_{2,i}, 3 \le i \le b, 2 \le j \le i\}.$$

The correctness of the codes from Construction 4 as well as their redundancy analysis are stated in the next theorem.

Theorem 4 *For sufficiently large n, there exists a code which can correct a consecutive deletion burst of size at most b whose number of redundancy bits is at most*

$$(b-1)\log(n) + \left(\binom{b}{2} - 1\right)\log(\log(n)) + \binom{b}{2} + \log(\log(b)).$$

V. Correcting a Burst of Length at most b (non-consecutively)

In this section, we briefly summarize our construction for correcting a non-consecutive deletion burst of length at most b for $b \le 4$. Note that for $b = 1$, we can use a VT-code and for $b = 2$, we use Levenshtein's construction [10]. We focus here on the case $b = 3$.

The construction uses a code which can correct two adjacent deletions immediately followed by a single insertion. For shorthand, we refer to this type of error as a $(2,1)$-*burst*, such a code is called a $(2,1)$-*burst-correcting code*, and the set of all $(2,1)$-bursts of a vector $\mathbf{x}$ is denoted by $D_{2,1}(\mathbf{x})$. For instance, if the vector $\mathbf{x} = (0,1,0,0,1,0)$ is transmitted then we get

$$D_{2,1}(\mathbf{x}) = \{(0,0,0,1,0),(1,0,0,1,0),(0,1,0,1,0), \\ (0,1,1,1,0),(0,1,0,0,0),(0,1,0,0,1)\}.$$

Note that $D_1(\mathbf{x}) \subseteq D_{2,1}(\mathbf{x})$ and hence every $(2,1)$-burst-correcting code is a single-deletion-correcting code as well.

The following is a construction for $(2,1)$-burst-correcting codes.

Construction 5 *For three integers $n \ge 4$, $a \in \mathbb{Z}_{2n-1}$, and $c \in \mathbb{Z}_4$, the code $\mathcal{C}_{2,1}(n,a,c)$ is defined as follows:*

$$\mathcal{C}_{2,1}(n,a,c) \triangleq \left\{\boldsymbol{x} \in \mathbb{F}_2^n : \sum_{i=1}^{n} x_i \equiv c \bmod 4, \\ \sum_{i=1}^{n} i \cdot x_i \equiv a \bmod (2n-1)\right\}.$$

We are now ready to present our construction for $b = 3$. The idea is it to use an intersection of three codes. The first one is a single-deletion-correcting code, the second one is a 3-burst-correcting codes, and the third one corrects a burst of size two. Note that we cannot use the code from Levenshtein [10] as the third code since the burst of two deletions are not necessarily adjacent. When correcting these deletions in a $2 \times (n/2)$ array we will get one row with a single deletion and the other suffers two adjacent deletions, followed by a single insertion. These two types of deletions can be corrected by the $(2,1)$-burst-correcting codes.

Construction 6 *Let $\mathcal{C}_3$ denote the code from Construction 2 for $b = 3$. For four integers n, $a_1 \in \mathbb{Z}_n$, $a_2 \in \mathbb{Z}_{2n-1}$, $c \in \mathbb{Z}_4$, let $\mathcal{C}_{b\le3}(n,a_1,a_2,c)$ be the following code:*

$$\mathcal{C}_{b\le3}(n,a_1,a_2,c) \triangleq \Big\{\boldsymbol{x} \in \mathbb{F}_2^n : \boldsymbol{x} \in VT_{a_1}(n), \boldsymbol{x} \in \mathcal{C}_3 \\ A_2(\boldsymbol{x})_1, A_2(\boldsymbol{x})_1 \in \mathcal{C}_{2,1}(\frac{n}{2}, a_2, c)\Big\}.$$

Theorem 5 *There exists a code by Construction 6 which can correct a non-consecutive burst of size at most 3 with redundancy at most $4\log(n) + 2\log(\log(n)) + 6$.*

Similarly, we provide another construction for the case $b = 4$ whose redundancy is $7\log(n) + 2\log(\log(n)) + 4$. However, we cannot extend these ideas for $b > 4$ and it is left as an open problem to construct efficient codes for correcting a non-consecutive burst of deletions of size $b > 4$.

VI. Conclusion and Open Problems

In this paper we studied burst-deletion-correcting codes in three models. Our main contribution is the construction of b-burst-correcting codes with redundancy at most $\log(n) + (b-1)\log(\log(n)) + b - \log(b)$. We extended this construction also for codes which correct a consecutive burst of size at most b, and studied codes which correct a burst of size at most b (not necessarily consecutive) for the cases $b = 3, 4$. While the results in the paper provide a significant contribution in the area of codes for insertions and deletions, there are still several interesting problems which are left open. This includes the construction of better codes, especially for the case of a non-consecutive deletion burst of size at most b and the derivation of tighter upper bounds for (consecutive and non-consecutive) bursts of size at most b.

References

[1] P. A. Bours, "Codes for correcting insertions and deletion errors," PhD thesis, Eindhoven University of Technology, Jun. 1994.

[2] J. Brakensiek, V. Guruswami, and S. Zbarsky, "Efficient low-redundancy codes for correcting multiple deletions," *CoRR*, vol. abs/1507.06175, 2015. [Online]. Available: http://arxiv.org/abs/1507.06175

[3] L. Cheng, T. G. Swart, H. C. Ferreira, and K. A. S. Abdel-Ghaffar, "Codes for correcting three or more adjacent deletions or insertions," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2014, pp. 1246–1250.

[4] D. Cullina, A. A. Kulkarni, and N. Kiyavash, "A coloring approach to constructing deletion correcting codes from constant weight subgraphs," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2012, pp. 513–517.

[5] F. Dandashi, A. Griggs, J. Higginson, J. Hughes, W. Narvaez, M. Sabbouh, S. Semy, and B. Yost, "Tactical edge characterization framework," *MITRE Technical Report MTR070331*, 2007.

[6] K. Immink, *Coding techniques for digital recorders*. Prentice Hall, College Div., 1991.

[7] J. Jeong and C. T. Ee, "Forward error correction in sensor networks," *University of California at Berkeley*, 2003.

[8] A. A. Kulkarni and N. Kiyavash, "Nonasymptotic Upper Bounds for Deletion Correcting Codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 8, pp. 5115–5130, Aug. 2013.

[9] V. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals (in russian)," *Doklady Akademii Nauk SSR*, vol. 163, no. 4, pp. 845–848, 1965.

[10] ——, "Asymptotically optimum binary code with correction for losses of one or two adjacent bits," *Systems Theory Research (translated from Problemy Kibernetiki)*, vol. 19, pp. 293–298, 1967.

[11] C. Schoeny, A. Wachter-Zeh, R. Gabrys, and E. Yaakobi, "Codes for correcting a burst of deletions or insertions," *CoRR*, vol. abs/1602.06820, 2016. [Online]. Available: http://arxiv.org/abs/1602.06820

[12] N. J. A. Sloane, "On single-deletion-correcting codes," in *Proc. Codes and Designs*, 2001, pp. 273–291.

[13] G. Tenengolts, "Nonbinary codes, correcting single deletion or insertion (corresp.)," *Information Theory, IEEE Transactions on*, vol. 30, no. 5, pp. 766–769, 1984.

[14] R. R. Varshamov and G. M. Tenengolts, "Codes which correct single asymmetric errors (in russian)," *Automatika i Telemkhanika*, vol. 161, no. 3, pp. 288–292, 1965.