An Exploration of Worst-Case Deletion Correcting Codes

Bora Uyumazturk*[1] and Vasco Portilheiro†[1]

[1]Department of Computer Science, Stanford University

August 17, 2019

Abstract

In this survey, we cover several results concerning codes capable of correcting adversarial deletions. We begin by introducing the problem, and exploring some of the related combinatorial structure of strings. We then prove an asymptotic (in terms of block-length) bound on the size of a binary code correcting 1-deletion due to Levenshtein, and subsequently show that Varmshamov-Tenengoltz codes meet this bound. We then turn to the problem of correcting more than one deletion. Here, we explore Kulkarni et. al.'s work on how casting the problem in the language of hypergraphs gives an explicit upper bound on the size of code correcting a constant number of deletions. Shifting to the problem of correcting a fraction of deletions, we present a recent construction by Bukh, Guruswami, and Håstad of positive rate codes correcting a $\sqrt{2}-1 \approx 0.4142$ fraction of deletions, which significantly closes in on the best known upper bound of $1/2$.

1 Introduction

In recent years, there has been flurry of research regarding codes capable of correcting adversarial deletions. First studied in the 1960's by Levenshtein, these codes are still relatively poorly understood. For example, the maximum fraction of deletion for which there exist positive rate codes is still unknown (though we present recent progress towards answering this question in the last section of this survey). Even for a constant number of deletions, optimal codes remain elusive: high rate construction are typically lack generalizability due to the complex combinatorial structure of string subsequences.

Recent theoretical interest has also been driven by new applications, such as DNA based coding and unreliable packet-based communication. While it is also reasonable in such settings to model the deletion channel probabilistically, e.g. with a fixed probability of

*Equal contribution. Contact: yuyumaz@stanford.edu.
†Equal contribution. Contact: vascop@stanford.edu.

deletion for each symbol in the input string, in this paper we are concerned with absolute guarantees on whether the input can be recovered. (For more on deletion channels in general, including probabilistic models, see Mitzenmacher's extensive survey [8].)

As with general error correcting codes, the goal is to create a *codebook* $\mathcal{C} \subseteq \Sigma^n$, where Σ is a finite alphabet, which should allow us to correct some number $d \leq n$ deletions. That is, given a possible corrupted message $\tilde{x}$ we must be able to recover the intended message x, for any $x \in \mathcal{C}$. It is easy to see that this is only possible if each *codeword* (element of the code) deletes to a distinct set of strings. This motivates the following definitions.

Definition 1. The *d-deletion ball* $D_d(x)$ of a string $x \in \Sigma^n$ consist of all strings reachable from x by d deletions, i.e. all length $n-d$ substrings of x:

$$D_d(x) = \{y \in \Sigma^{n-d} : y \prec x\},$$

Note that by $y \prec x$ we mean that y is a substring (or subsequence) of x, i.e. the characters in y occur in the same order in x, but not necessarily consecutively. When referring to substring of consecutive symbols, we will instead use the term *subword*.

Definition 2. A *d-deletion correcting code of block-length n* (or, simply, a d-deletion code) $\mathcal{C} \subseteq \Sigma^n$ is a set of length n strings whose d-deletion balls are all distinct. That is, for any $x, y \in \mathcal{C}$ with $x \neq y$,

$$D_d(x) \cap D_d(y) = \emptyset.$$

We may also refer to a d-deletion code of length n over some alphabet Σ as an (n, d, Σ) deletion code. While in general we want to maximize d while minimizing n, best tradeoff is as of yet unknown. We will begin by exploring the case where $d = 1$. We derive an asymptotic bound on n in this case, originally due to Levenshtein [5], and then describe Varshamov-Tenengoltz codes, which meet this bound. We will then describe several approaches to the $d \geq 2$ case.

2 The Combinatorial Structure of 1-deletion and 1-insertion

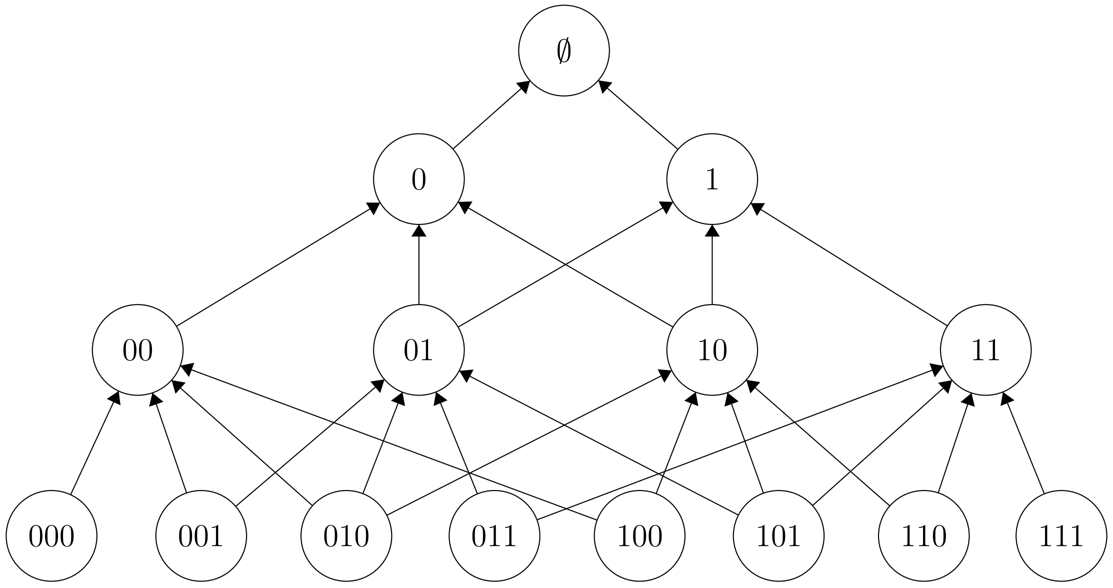
We begin by analyzing the size and distribution of d-deletion balls. Here, we will restrict ourselves to the binary alphabet $\Sigma = \mathbb{F}_2$, and single deletions, i.e. $d = 1$. (We will treat some of the work for $d > 1$ in later sections.) We begin by asking two fundamental questions.

Question 1. *Consider any binary string $x \in \mathbb{F}_2^n$. How many distinct strings can we get to by deleting a single symbol, or* bit*, from x? That is, what is the size of the 1-deletion ball of x, $|D_1(x)|$?*

Question 2. *Consider any binary string $y \in \mathbb{F}_2^{n-1}$. How many length n strings could y be arrived at by deleting a single bit? That is, how many elements have y in there 1-deletion balls, i.e. how big is $\{x \in \mathbb{F}_2^n : y \in D_1(x)\}$?*

Surprisingly, the answer to the second question above has a much simpler answer than the first one. However, it will point us towards the answer to the first — the size of a 1-deletion ball — which will prove much more useful for results to follow.

Some intuition towards answering these question can be gained by examining the diagram below. In particular, we have a tree whose n-th level contains the binary length n strings. Edges in the tree go from length n strings to the length $n-1$ strings reachable by a single deletion.



Looking at this tree, one might hypothesize that every node at the n-th level connects "down" to $n+2$ nodes in the $(n+1)$-th level. This would answer our second question above: every length $n-1$ string is a 1-deletion away from $n+1$ distinct strings of length n. It turns out that this is in fact true. Furthermore, the proof contains an idea that will help us answer our other, more useful question.

The crucial insight will be to look at *runs* of consecutive bits in our string. For example, consider the string 000111100. This string consists of three total runs: a run of 0's of length 3, a run of 1's of length 4, and a run of 0's of length 2. For the sake of convenience, we define the *number of runs* function.

Definition 3. For any alphabet Σ, the *number of runs* function $r : \Sigma^* \to \mathbb{N}$ gives for any string the number of distinct runs of consecutive symbols in that string.

Lemma 2.1. *For $n \geq 1$, for every $y \in \mathbb{F}_2^n$, we have*

$$|\{x \in \mathbb{F}_2^{n+1} : y \in D_1(x)\}| = n + 2.$$

Proof. Consider any $y \in \mathbb{F}_2^n$. Let r_i be the length of the i-th run, such that

$$\sum_{i=1}^{r(y)} r_i = n.$$

We want to count the number of strings that y is a 1-deletion away from. Equivalently, we want to count how many distinct strings we can reach from y by inserting a single bit. We observe that by inserting a bit, we can never decrease the number of runs. We will thus consider two cases: the length $n+1$ strings we can reach that also contain $r(y)$ runs, and those that contain more than $r(y)$ runs.

It is clear that when inserting a bit, the number of runs can only remain the same if the bit is being inserted into a run of that same bit. That is, if the bit being inserted is a 1 (respectively 0), the number of runs remains the same if and only if it is being inserted into a run of 1's (respectively 0's). Since y contains $r(y)$ runs, there are $r(y)$ strings of length $n+1$ we can reach by a single insertion that also contain $r(y)$ runs.

There are two ways to increase the number of runs by inserting a bit. One is prepending a bit to the beginning of the string which differs from the first run, or append a bit at the end that differs from the last run, thus increasing the number of runs by one. This gives two more strings reachable from y by 1-insertion. The other way to increase the number of runs is by inserting a bit into a run of the opposite bit, e.g. inserting a 1 into a run of 0's. Note here that we restrict to inserting between two bits of the existing run, and not at the end (or beginning) of the existing run, as this would simply extend the next (or prior) run. For each run i, of length r_i, there are $r_i - 1$ positions "between" the bits in the run, and thus $r_i - 1$ strings reachable in this way by 1-insertion from y.

Summing up the contribution from each of these cases, we get a total number of strings reachable by 1-insertion

$$r(y) + 2 + \sum_{i=1}^{r(y)} r_i - 1 = r(y) + 2 + n - r(y) = n + 2.$$

□

This proof motivates looking at strings from the perspective of runs of bits. This lens directly helps answer our first question above.

Lemma 2.2. *For any alphabet Σ and any $x \in \Sigma^n$ ($n \geq 1$), the size of the 1-deletion ball is the number of runs $r(x)$. That is $|D_1(x)| = r(x)$.*

Proof. The proof of this fact comes simply by observing that within a single run, it doesn't matter which particular symbol is deleted: the resulting length $n-1$ string is the same. On the other hand, every single deletion must happen withing *some* run. Therefore, there are exactly the same number of distinct length $n-1$ substrings of x as there are runs, and $|D_1(x)| = r(x)$. □

Having established the size of single deletion balls, we ask a natural follow-up question, the answer to which again proves useful in the subsequent sections. In particular, for a given length n, how many strings are there of m runs? In answering this question, we again restrict ourselves to the binary case, obtaining the following result, initially due to Levenshtein [5].

Lemma 2.3. *In $\mathbb{F}_2^n$ the number of strings with m runs is*

$$|\{x \in \mathbb{F}_2^n : r(x) = m\}| = 2\binom{n-1}{m-1}.$$

Proof. Consider all strings of m runs beginning with a run of 0's. The number of such strings is the number of ways to pick the $m-1$ boundaries between the runs. The possible positions for these boundaries are the $n-1$ positions between n total symbols. There are therefore $\binom{n-1}{m-1}$ strings of length n that begin with 0 and contain m runs. Repeating this logic to count the strings beginning with 1 completes the proof. $\square$

The result above is useful in proving a bound on the size of a binary 1-deletion code asymptotic in the block-length, which gives us a way of evaluating whether a given family of binary deletion codes is "asymptotically optimal." In fact, this bound will allow us to demonstrate that one particular family of codes is optimal in this sense. For the rest of this survey, let $C(n, d, q)$ be an optimal (i.e. of largest cardinality) d-deletion code of length n and alphabet size q. The following bound for the case $q = 2$ and $d = 1$ is due to Levenshtein [5].

Theorem 2.4.

$$|C(n, 1, 2)| \sim \frac{2^n}{n}, \tag{1}$$

as $n \to \infty$.

The proof of this theorem will be done in two parts. We will first prove that $\frac{2^n}{n}$ is an asymptotic upper bound on $|C(n, 1, 2)|$. It will then remain to prove the existence of codes achieving this upper bound. One such family of codes is Varshamov-Tenengoltz codes, which are the subject of the next section. We thus defer the proof of Theorem 2.4 until the next section, and here only prove the upper bound statement.

Lemma 2.5.

$$|C(n, 1, 2)| \lesssim \frac{2^n}{n}, \tag{2}$$

as $n \to \infty$.

Proof. Fix some n and let $C = C(n)$. We now partition the codewords in C by the sizes of their 1-deletion balls, as follows. First, we let C_1 by the set of codewords with approximately $n/2$ runs. In particular,

$$C_1 = \{x \in \mathbb{F}_2^n : \frac{n}{2} - \sqrt{n \log n} \leq r(x) \leq \frac{n}{2} + \sqrt{n \log n}\}.$$

Let $C_2 = C \setminus C_1$ be the remaining codewords. We will now show that for large n, most codewords are in C_1, which in turn is bounded above by $\frac{2^n}{n}$.

To find the size of C_1, note first that by assumption the 1-deletion balls of the codewords in C are distinct, since C is a 1-deletion code. Thus, the size of C_1 cannot be larger than the maximum number of deletion balls which can be "packed" into $\mathbb{F}_2^{n-1}$:

$$|C_1| \leq \frac{2^{n-1}}{\min_{x \in C_1} |D_1(x)|} \leq \frac{2^{n-1}}{\frac{n}{2} - \sqrt{n \log n}} \lesssim \frac{2^n}{n}.$$

The remaining codewords in C are those in C_2. The number of these codewords is given by counting how many codewords there are for each of the remaining number of runs

$$\begin{aligned}|C_2| &\leq \sum_{m=1}^{\frac{n}{2}-\sqrt{n\log n}} 2\binom{n-1}{m-1} + \sum_{m=\frac{n}{2}+\sqrt{n\log n}}^{n} 2\binom{n-1}{m-1} \\ &\leq 2\sum_{m=1}^{\frac{n}{2}-\sqrt{n\log n}} 2\binom{n-1}{m-1} \ll \frac{2^n}{n}.\end{aligned}$$

□

3 Varshamov-Tenengoltz Codes

Varshamov-Tenengoltz codes form a family of 1-deletion codes. For a given block length n, there are parametrized by a *checksum* value $a \in \{0, \ldots, n\}$. They are defined as follows.

Definition 4. For a given $n \in \mathbb{N}$ we define the *Varshamov-Tenengoltz* code of checksum a to be

$$VT_a(n) = \{c \in \mathbb{F}_2^n : \sum_{i=1}^{n} ic_i \equiv a \mod n+1\}.$$

We first note the following fact, that shows that VT codes indeed asymptotically achieve the bound in Lemma 2.5.

Lemma 3.1. *As $n \to \infty$, the largest VT code of block length n is asymptotically of size $\frac{2^n}{n}$*

Proof. Since the $n+1$ different VT codes of size n partition $\mathbb{F}_2^n$, at least one of them must have size at least $\frac{2^n}{n+1}$, by the pidgeonhole principle. (For a fully rigorous treatment of the size of VT codes, see Sloane's indispensable survey on 1-deletion codes [11]. It turns out that the maximum is achieved by $a = 0$.) □

It thus remains to show that VT codes can in fact correct 1-deletions. We will first present the algorithm (Algorithm 1), and then prove that this algorithm is correct.

Algorithm 1: VT 1-deletion correction

```
Data: $n \in \mathbb{N}, a \in \{0, \ldots, n\}, \tilde{c} \in \mathbb{F}_2^{n-1}$ such that $\tilde{c} \in D_1(c)$ for some $c \in VT_a(n)$
Result: $c \in VT_a(n)$ such that $\tilde{c} \in D_1(c)$
begin
    $s \longleftarrow \sum_{i=0}^{n-1} i\tilde{c} \mod n+1$
    $\Delta \longleftarrow a - s \mod n+1$
    if $\Delta \leq wt(\tilde{c})$ then
        $c_1, c_2 \longleftarrow CountFromRight(\tilde{c}, 1, \Delta)$
        $b \longleftarrow 0$
    else
        $c_1, c_2 \longleftarrow CountFromRight(\tilde{c}, 0, n - \Delta)$
        $b \longleftarrow 1$
    $c \longleftarrow c_1 b c_2$
```

Note that we use c_1bc_2 to denote the concatenation of the three strings, and $wt(\cdot)$ to denote the function counting the number of 1's in a string (i.e. the *weight*). By $CountFromRight(\tilde{c}, e, \Delta)$ we mean a subroutine returning the given string $\tilde{c}$ split into two parts c_1 and c_2 such that c_2 contains only Δ occurrences of the bit e. (Such a subroutine would reasonably run in time $O(n)$.)

The proof of correctness of Algorithm 1 is originally due to Levenshtein [5].

Theorem 3.2. *Given $\tilde{c} \in \mathbb{F}_2^{n-1}$ such that $\tilde{c} \in D_1(c)$ for some $c \in VT_a(n)$, Algorithm 1 correctly identifies c.*

Proof. Consider any $c \in VT_a(n)$. Let $\tilde{c} \in \mathbb{F}_2^{n-1}$ be the string obtained by deleting the single bit b from position p in c. That is, if we write c as

$$c = c_1, c_2, \ldots, c_{p-1}, c_p, c_{p+1}, \ldots, c_{n-1}, c_n$$

we can correspondingly write $\tilde{c}$ as

$$\tilde{c} = c_1, c_2, \ldots, c_{p-1}, c_{p+1}, \ldots, c_{n-1}, c_n.$$

Let $R_1 = wt(c_{p+1}, \ldots, c_n)$ be the number of 1's to the right of the deletion, and similarly let $R_0 = n - p - R_1$ be the number of 0's to the right of the deletion. As in the algorithm, let $s \equiv \sum_{i=1}^{n-1} i\tilde{c} \mod n+1$ be the checksum of $\tilde{c}$, and let $\Delta \equiv a - s \mod n+1$. Note that $\Delta = pb + R_1$, as follows (for simplicity, we drop the $\mod n+1$ here):

$$\begin{aligned}
\Delta &\equiv a - \sum_{i=1}^{n-1} i\tilde{c}_i \\
&\equiv a - \left(\sum_{i=1}^{p-1} i\tilde{c}_i + \sum_{i=p}^{n-1} i\tilde{c}_i \right) \\
&\equiv a - \left(\sum_{i=1}^{p-1} ic_i + \sum_{i=p+1}^{n} (i-1)c_i \right) \\
&\equiv a - \left(\sum_{i=1}^{p-1} ic_i + \sum_{i=p+1}^{n} ic_i - \sum_{i=p+1}^{n} c_i \right) \\
&\equiv a - \left(\sum_{i=1}^{p-1} ic_i + \sum_{i=p+1}^{n} ic_i - R_1 \right) \\
&\equiv a + R_1 - \left(\sum_{i=1}^{n} ic_1 - pb \right) \\
&\equiv pb + R_1.
\end{aligned}$$

We now proceed by cases on the value of b.
Case $b = 0$. Note that in this case $\Delta = R_1 \leq wt(\tilde{c})$. Therefore, the algorithm will be in the correct branch of the if-statement, and correctly count R_1 1's from the end of $\tilde{c}$ before inserting a 0.

Case $b = 1$*.* Note that in this case $\Delta = p + R_1 > wt(\tilde{c})$, since the number of 1's before index p in c must be strictly less than p. This again shows that the algorithm will be in the correct branch of the if statement. Now note that

$$\Delta = p + R_1 = p + (n - p - R_0) = n - R_0,$$

and so $R_0 = n - \Delta$. So the algorithm will correctly count R_0 0's from the end of $\tilde{c}$ before inserting a 1. □

Thus, VT codes can correct 1-deletions (in linear time). This completes the proof of Theorem 2.4 by showing that VT codes are asymptotically optimal 1-deletion codes.

4 Upper Bounds on d Deletion codes for $d \geq 1$

4.1 From deletion, to hypergraphs, to linear programs

Levenshtein's asymptotic upper bound above relies critically on the simple expression of the size of deletion balls in terms of the number of runs in a given string. Moreover, since we are only considering a constant number of deletions, one might prefer to have a non-asymptotic result in order to evaluate construction for small n, say. In [4], Kulkarni et. al. establish a non-asymptotic upper bound for multiple deletions. While the result is valuable in and of itself, the methodology, which introduces the use of hypergraphs for analysis of deletion codes, is also of independent interest.

The strategy here will be to translate the problem of finding the maximum size of deletion codes into a statement about hypergraphs. We will then formulate an equivalent linear program, and by relaxing the linear program and looking at the resulting dual, we will arrive at a good upper bound.

Definition 5. A *hypergraph* $\mathcal{H}$ is a tuple $(X, \mathcal{E})$, where X is a finite set of vertices and $\mathcal{E}$ is a collection of nonempty subsets of X, such that $\cup_{E \in \mathcal{E}} E = X$. Elements of $\mathcal{E}$ are commonly referred to as hyperedges.

A hypergraph generalizes the notion of a graph. While edges in a graph just connect pairs of vertices, an edge in a hypergraph (a hyperedge) is simply a subset of the set of vertices, connecting any number of vertices. If a vertex is in an edge, we say that it is *covered* by that edge. Also, we call two hyperedges *disjoint* if their intersection is empty.

We now show how deletion codes can be described using a hypergraph. Consider the hypergraph $\mathcal{H}_{n,d,\Sigma} = (\Sigma^{n-d}, \mathcal{D}_d(\Sigma^n))$, where $\mathcal{D}_d(\Sigma^n) = \{D_d(x) : x \in \Sigma^n\}$ is the collection of deletion balls around every string in Σ^n. This graph completely captures the relationships between codes and their subsequences. In this context, the problem of finding a deletion code amounts to finding a collection of hyperedges in $\mathcal{H}_{n,d,\Sigma}$ which are pairwise disjoint. It turns out such collections are well studied in the theory of hypergraphs, and are called matchings.

Definition 6. A *matching* of a hypergraph $\mathcal{H} = (X, \mathcal{E})$ is a collection of pairwise disjoint hyperedges. The matching number $\nu(\mathcal{H})$, is the size of the largest matching of $\mathcal{H}$.

Thus the matching number of $\mathcal{H}_{n,d,\Sigma}$ is exactly the size of the largest d-deletion code in Σ^n. The following notion is "dual" to matchings, in a way which we will soon make precise.

Definition 7. A *transversal* of $\mathcal{H} = (X, \mathcal{E})$ is a subset of vertices T such that every edge covers at least one vertex in T. The transversal number $\tau(\mathcal{H})$ is the size of the smallest such set.

Finally, we define the incidence matrix of a hypergraph.

Definition 8. Let $\mathcal{H} = (X, \mathcal{E})$ be a hypergraph with n vertices $x_1, ..., x_n$ and m edges $E_1, ..., E_m$. Then the indicidence matrix A of $\mathcal{H}$ is the n-by-m matrix with $A_{ij} = 1$ if $x_i \in E_j$, and 0 otherwise.

Note that the incidence matrix depends on the ordering of the vertices and the edges, so from now on we will assume that these are fixed. Using the incidence matrix, we see that $\nu(\mathcal{H})$ and $\tau(\mathcal{H})$ are in fact solutions of the following binary programs.

Lemma 4.1. *Let $\mathcal{H} = (X, \mathcal{E})$ be a hypergraph with n vertices and m edges, and let A be the incidence matrix of $\mathcal{H}$. Then:*

$$\nu(\mathcal{H}) = \max\{\mathbf{1}^T z : Az \leq \mathbf{1}, z \in \mathbb{F}_2^m\}$$
$$\tau(\mathcal{H}) = \min\{\mathbf{1}^T w : A^T w \geq \mathbf{1}, w \in \mathbb{F}_2^n\}.$$

Proof. We first show that any $z \in \mathbb{F}_2^m$ such that $Az \leq \mathbf{1}$ corresponds to a valid matching on $\mathcal{H}$. Let $S(z)$ be the set of hyperedges of $\mathcal{H}$ corresponding to the indices of z which are equal to 1. Then by the definition of A, we see that the j-th element of Az is the number of hyperedges in $S(z)$ which cover vertex x_j. If this is less than or equal to 1 for each j, it immediately implies that no vertex is covered by 2 or more hyperedges in $S(z)$, so $S(z)$ is a valid matching. It is easy to see that any matching of $\mathcal{H}$ corresponds to a vector z satisfying all the constraints, so there is a one-to-one correspondence between matchings and feasible vectors. Taking the maximum weight of all these vectors will give us the matching number $\nu(\mathcal{H})$.

Similarly, any feasible w corresponds to a valid transversal, and vice versa. Let $T(w)$ be the set of vertices corresponding to the indices where w has a 1. Then the i-th coordinate of $A^T w$ is exactly the number of edges which cover at least one vertex in $T(w)$. If this is greater than or equal to 1 for all i, then $T(w)$ is a transversal. Clearly any transversal has a corresponding feasible w, so the correspondence is one-to-one. Therefore taking the minimum weight over all feasible w will give us $\tau(\mathcal{H})$. $\square$

Moreover we can relax this so that $z \in \mathbb{N}^m$ and $w \in \mathbb{N}^n$, since it would clearly never help to have a coordinate be greater than 1. Therefore we have the integer programs:

$$\nu(\mathcal{H}) = \max\{\mathbf{1}^T z : Az \leq \mathbf{1}, z \in \mathbb{N}^m\}$$
$$\tau(\mathcal{H}) = \min\{\mathbf{1}^T w : A^T w \geq \mathbf{1}, w \in \mathbb{N}^n\}.$$

We have demonstrated that $\nu(\mathcal{H})$ and $\tau(\mathcal{H})$ can be described as solution to integer programs, and moreover, we can see that these integer programs are duals of each other!

It is a fundamental theorem that *weak duality* holds for integer programs. That is, any solution of the dual problem is an upper bound on solutions of the primal problem. In our case, this implies that

$$\nu(\mathcal{H}) \leq \tau(\mathcal{H}). \tag{3}$$

We have made some progress, but the solutions to these integer programs are still fairly opaque. To get explicit bounds, Kulkarni et al. ([4]) consider the real-valued relaxations of these integer programs. Letting $\nu^*(\mathcal{H})$ and $\tau^*(\mathcal{H})$ denote the values of these relaxations, we have the linear programs:

$$\nu^*(\mathcal{H}) = \max\{\mathbf{1}^T z : Az \leq \mathbf{1}, z \geq 0\} \tag{4}$$

$$\tau^*(\mathcal{H}) = \min\{\mathbf{1}^T w : A^T w \geq \mathbf{1}, w \geq 0\}. \tag{5}$$

We call $\nu^*(\mathcal{H})$ and $\tau^*(\mathcal{H})$ the *fractional matching number* and *fractional transversal number*, respectively. We will also refer to any element of $\{z : Az \leq \mathbf{1}, z \geq 0\}$ as a *fractional matching* and any element of $\{w : A^T w \geq \mathbf{1}, w \geq 0\}$ as a *fractional transversal.* Because (4) and (5) are now dual linear programs, their solutions obey *strong duality*, allowing us to conclude that

$$\nu^*(\mathcal{H}) = \tau^*(\mathcal{H}). \tag{6}$$

Now, since the feasible regions for $\nu(\mathcal{H})$ and $\tau(\mathcal{H})$ are contained in the feasible regions for their fractional counterparts, we see that

$$\nu(\mathcal{H}) \leq \nu^*(\mathcal{H}) \text{ and } \tau(\mathcal{H}) \geq \tau^*(\mathcal{H}). \tag{7}$$

Combining these with with (6), we have that for any fractional transversal w,

$$\nu(\mathcal{H}) \leq \nu^*(\mathcal{H}) = \tau^*(\mathcal{H}) \leq \mathbf{1}^T w. \tag{8}$$

This key observation is what will allow us to upper bound $\nu(\mathcal{H}_{n,d,\Sigma})$, which immediately implies a bound on the size of any (n, d, Σ) deletion code.

4.2 Bigger deletion balls ($d > 1$)

In order to prove our upper bound, we will need to develop more understanding of deletion and insertion sets of strings. In particular, we will make use of the following lemma, due to Hirshbirg and Regnier [9].

Lemma 4.2. *Let $s \in \mathbb{N}$, $s \leq n$, and let $x \in \Sigma^n$. Then if $x \in D_s(y)$, then*

$$|D_s(x)| \leq |D_s(y)|. \tag{9}$$

The proof of this is a result of the following lemma, which we will prove to further illustrate the unique flavor of analysis that deletion coding demands.

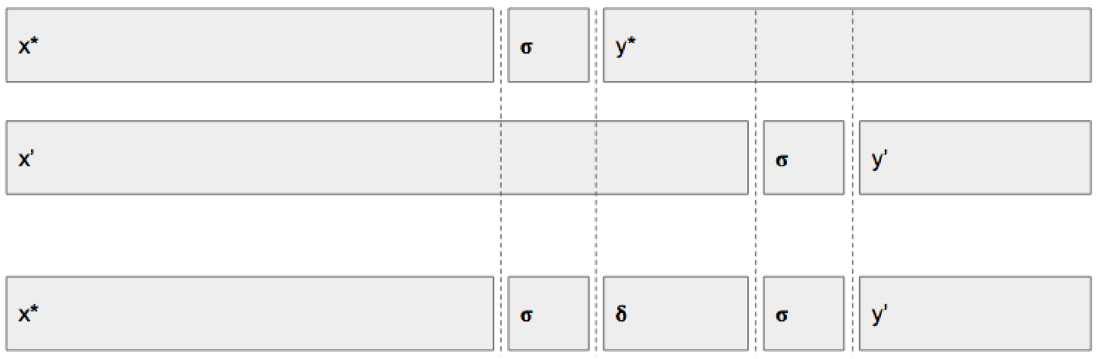


Figure 1: Decomposition of l induced by decompositions $x^*\sigma y^*$ and $x'\sigma y'$.

Lemma 4.3. *Let $x \in \Sigma^n$, $y \in \Sigma^m$, and let $\sigma \in \Sigma$, and let $s \in \mathbb{N}$. Then*

$$|D_s(xy)| \leq |D_s(x\sigma y)|, \tag{10}$$

where xy and $x\sigma y$ denote the concatenations of those strings.

Proof. We begin by looking at elements of $D_s(x\sigma y)$ which can still be expressed as $x'\sigma y'$ for sufficiently long subsequences x' and y' of x and y. Formally let

$$D_s^\sigma(x\sigma y) = \{l \in D_s(x\sigma y) : \exists s' \leq s, x' \in D_s'(x), y' \in D_{s-s'}(y) \text{ s.t. } l = x'\sigma y'\}.$$

Note that $D_s^\sigma(x\sigma y) \subseteq D_s(x\sigma y)$. We will show that $D_s^\sigma(x\sigma y)$ maps surjectively onto $D_s(xy)$. We define $\Gamma : D_s^\sigma(x\sigma y) \to D_s(xy)$ as follows. Write each $l \in D_s^\sigma(x\sigma y)$ as the concatenation $l = x'\sigma y'$, for some $x' \in D_{s'}(x)$ and $y' \in D_{s-s'}(y)$, where x' is chosen to have maximal length. It is clear that there is a unique way to do this. Then define $\Gamma(l) = x'y'$. Therefore each l maps to a unique string in $D_s(xy)$, making Γ well-defined.

To see that it is surjective, consider any $t \in D_s(xy)$. Write $t = x^*y^*$ (again, $x^* \in D_{s'}(x), y^* \in D_{s-s'}(y), s' \leq s$) where x^* has maximal length. Now consider $l = x^*\sigma y^*$. If this x^* is maximal in this decomposition of l, then $\Gamma(l) = x^*y^* = t$. For the sake of contradiction, assume that $l = x'\sigma y'$ for some x', y' where x' is strictly longer than x^*. In particular, this implies that $l = x^*\sigma\delta\sigma y'$ (see Figure 1), where $x'\sigma\tilde{y} = x^*$, and $\tilde{y}\sigma y^* = y'$.

First assume δ is non-empty. Then $x^*\sigma\delta \prec x' \prec x$, so $x^*\delta \prec x$. Similarly, $\sigma y' \prec y^* \prec y$. But this means that $x^*\delta\sigma y' = t$, a contradiction, since $x^*\delta$ is longer than x^*, which was assumed to be maximal.

Now assume δ is empty, so $x^*\sigma\sigma y' = l$, that is, $x' = x^*\sigma$, and $y^* = \sigma y'$. Therefore $x'y' = x^*\sigma y' = x^*y^* = t$, which again contradiction the fact that x^* was maximal.

Therefore x^* was in fact maximal, so $\Gamma(l) = t$, so Γ surjective. This implies that

$$|D_s(xy)| \leq |D_s^\sigma(x\sigma y)| \leq |D_s(x\sigma y)|,$$

which is what we wanted to prove. □

Observe that Lemma 4.2 can now be proven by inductively applying Lemma 4.3.

4.3 Putting it all together

Armed with Lemma 4.2 we can now proceed towards upper bounding $|C(n, d, q)|$.

Theorem 4.4. *Let Σ be a finite alphabet of size q, and let $d, n \in \mathbb{N}$ with $d < n$. Then*

$$|C(n, d, q)| \leq \sum_{x \in \Sigma^{n-d}} \frac{1}{|D_s(x)|}. \tag{11}$$

Proof. We do this by constructing a fractional transversal of $\mathcal{H}_{n,d,\Sigma}$ and applying (8). Recall that the vertices $\mathcal{H}_{n,d,\Sigma}$ correspond to elements of Σ^{n-d}, so to construct a fractional transversal, we must specify $w(x)$ for each $x \in \Sigma^{n-d}$. For each such x, we let

$$w(x) = \frac{1}{|D_s(x)|}. \tag{12}$$

Now we verify that this is a valid fractional transversal. Clearly $w(x) \geq 0$ for all x. To check that it satisfies the constraint $A^T w \geq \mathbf{1}$, we need that for each $y \in \Sigma^n$ (recall that hyperedges correspond to codes in the original space),

$$\sum_{x \in D_s(y)} w(x) \geq 1. \tag{13}$$

Expanding this, we see that

$$\begin{aligned} \sum_{x \in D_s(y)} w(x) &= \sum_{x \in D_s(y)} \frac{1}{|D_s(x)|} && (14) \\ &\geq \sum_{x \in D_s(y)} \frac{1}{|D_s(y)|} && (15) \\ &= 1, && (16) \end{aligned}$$

where (15) is a consequence of Lemma 4.2. Therefore w is a valid fractional transversal, and so applying (8), we see that

$$\begin{aligned} |C(n, d, q)| &= \nu(\mathcal{H}_{n,d,\Sigma}) && (17) \\ &\leq \mathbf{1}^T w && (18) \\ &= \sum_{x \in \Sigma^{n-d}} \frac{1}{|D_s(x)|}, && (19) \end{aligned}$$

proving the claim. □

To gain some understanding of how this bound behaves, we can use our combinatorial results from Section 1 to find the closed form solution for $d = 1$.

Corollary 1.

$$|C(n, 1, 2)| \leq \frac{2^n - 2}{n - 1}. \tag{20}$$

Proof. From Theorem 4.4, we know that:

$$|C(n,1,2)| \le \sum_{x\in\mathbb{F}_2^{n-1}} \frac{1}{|D_1(x)|}. \tag{21}$$

Recall that in the case of 1-deletion, the expression for the deletion ball is very simple: $D_1(x) = r(x)$, where $r(x)$ is the number of runs in x. Therefore

$$\begin{aligned}
\sum_{x\in\mathbb{F}_2^{n-1}} \frac{1}{|D_1(x)|} &= \sum_{x\in\mathbb{F}_2^{n-1}} \frac{1}{r(x)} &(22)\\
&= \sum_{m=1}^{n-1} |\{x \in \mathbb{F}_2^{n-1} : r(x) = m\}| \frac{1}{m} &(23)\\
&= \sum_{m=1}^{n-1} 2\binom{n-2}{m-1}\frac{1}{m} &(24)\\
&= \frac{2}{n-1}\sum_{m=1}^{n-1}\binom{n-1}{m} &(25)\\
&= \frac{2(2^{n-1}-1)}{n-1} &(26)\\
&= \frac{2^n-2}{n-1}. &(27)
\end{aligned}$$

□

We see that asymptotically (20) coincides with Levenshtein's original bound of $\frac{2^n}{n}$. However, note that this bound also holds for small n as well.

Unfortunately, the size of deletion balls for $d > 1$ are more complicated, and do not admit a simple explicit form. However, Kulkarni et. al. use Theorem 4.4 in combination with various bounds on deletion balls to prove the following non-asymptotic result.

Theorem 4.5. *Let $|\Sigma| = q$, and let $d, n \in \mathbb{N}$, $d < n$. Then then*

$$|C(n,d,q)| \le \frac{d!q^n}{(q-1)^d n^d}. \tag{28}$$

It is difficult to assess the value of these bounds, since there are relatively few code constructions to compare them against. However, there have been recent results for binary 2-deletion codes which begin to put these bounds in context. To match the phrasing of these results, we translate Theorem 4.5 into a statement about the number of redundancy bits required for deletion correction. We see that number of redundnancy bits for an $(n, 2, \mathbb{F}_2)$ deletion code is bounded by the following:

$$\begin{aligned}
n - \log_2|\mathcal{C}| &\ge n - \log_2\left(\frac{2^{n+1}}{n^2}\right) &(29)\\
&= n - (n + 1 - 2\log_2(n)) &(30)\\
&= 2\log_2(n) - 1. &(31)
\end{aligned}$$

In 2015, Brakensiek et. al. [1] published a buffer-based construction for an arbitrary constant number of deletions d which had $c_d \log_2(n)$ bits of redundancy, where $c_d = O(d^2 \log d)$. For $d = 2$ this gave codes with rate approaching 1, however the authors of [3] show that the constant is $c_2 = 128$, which is far from the lower bound of $2\log_2(n) - 1$. They propose a new construction which only uses $8\log_2(n) + O(\log_2 \log_2 n)$ redundancy bits. Even more recently, Sima et. al. [10] introduce a code which only uses $7\log_2(n) + o(\log_2(n))$ bits of redundancy, which we can see is very close to optimal. Unfortunately the latter two constructions pertain to the case $d = 2$. Redundancy upper bounds for larger d remain an open question.

5 Coding for a fraction of deletions

We conclude with a discussion of a different parameter regime, where instead of deleting a constant number of symbols, the adversary can now delete a constant fraction p of the bits sent. More precisely, if n is the block length of a code $\mathcal{C}$, the adversary can delete pn bits, which means that $(1-p)n$ bits are received on the other end. Recent research has focused on determining $p^*(q)$, defined as the maximum fraction of deletions such that there exists a family of pn-deletion correcting codes over an alphabet of size q with rate bounded away from zero. We will focus on the particular case where $q = 2$, i.e. binary alphabets.

A first obvious upper bound is $p^*(2) \leq \frac{1}{2}$. This is because an adversary who can delete half of the bits can just delete every occurrence of the minority bit, so that all that remains is all 1s or all 0s. (This generalizes to length a k alphabet, with $p^*(q) \leq 1 - \frac{1}{q}$). Surprisingly, the question of whether $p^*(2)$ is strictly less than $\frac{1}{2}$ is still open.

A priori one might think that that $p^*(2)$ should be much less than $\frac{1}{2}$, since that upper bound would also applies to erasure codes, which don't have the same synchronization challenges. However, in 2017, Bukh, Guruswami, and Håstad [2] presented a code which achieves a positive rate for up to a $p = \sqrt{2} - 1 \approx 0.414$ fraction of deletions. This marks a considerable improvement over the previous lower bound of 0.17, which was achieved using a random coding argument [7].

We will now provide a sketch of the construction in [2]. Their approach makes use of concatenated codes, which typically use an explicit "outer code" over a larger alphabet, and then a good (usually random) "inner code", which encodes each of the symbols in the outer code. Curiously, in this construction, the outer code is found using a random coding argument, while the inner code is given explicitly.

Before we discuss this explicit construction, we will need a few definitions.

Definition 9. Given a subsequence w' of a string w, let the *span* of w' in w, denoted $span_{w'}(w)$, be the length of the shortest contiguous subword of w which contains the subsequence w'.

For example, the span of 101 in 010010 is 4.

Definition 10. If w' is a common subsequence of w_1 and w_2 ($w' \prec w_1$ and $w' \prec w_2$), we the define the *mutual span* of w' in w_1 and w_2 as

$$span_{w'}(w_1, w_2) = span_{w'}(w_1) + span_{w'}(w_2).$$

For sake of convenience we may sometimes say that span of a code $\mathcal{C}$ is large if the mutual span of any subsequence w' in any two codewords $w_1, w_2 \in \mathcal{C}$ scales up quickly (read, with a large linear factor) in the length of the subsequence.

Definition 11. Finally, let $LCS(w_1, w_2)$ be the length of the longest common subsequence of w_1 and w_2.

Note that $\mathcal{C}$ is a p-fraction deletion code if and only if for any two codewords c_1 and c_2, $LCS(c_1, c_2) < (1-p)n$. The goal, then, is to ensure that the LCS of any two codewords of the concatenated code is sufficiently small. We will sometimes abuse notation and refer to the LCS of an entire code $\mathcal{C}$ to mean the maximal LCS between any two codewords in $\mathcal{C}$.

The following lemma gives the key quantitative relationship between the *span* of common subsequences of two strings and their LCS, which the authors of [2] exploit to prove that the LCS of the final concatenated code is sufficiently small.

Lemma 5.1. *Let $w_1, w_2 \in \Sigma^n$. If $span_{w'}(w_1, w_2) \geq b \cdot len(w') - \delta$ for every common subsequence w' of w_1 and w_2 and some $\delta > 0$, then $LCS(w_1, w_2) \leq \frac{2n+\delta}{b}$.*

This is easily seen by observing that for $w_1, w_2 \in \Sigma^n$ it must be the case that

$$span_{w'}(w_1, w_2) \leq 2n.$$

One can draw an analogy between the expansion requirements of Lemma 5.1 and the connectivity properties of the expander graphs used in the construction of high-rate expander codes [12].

To ensure the concatenated code has a large span (and thus small LCS by Lemma 5.1), the authors proceed according to the following roadmap. First, they construct the inner code to be composed of "harmonic" codewords, which they show have large spans. They then establish a bound on the span of the concatenated code in terms of the LCS of the outer code, showing that outer codes with sufficiently small LCS give concatenated codes with large span. Finally, they use probabilistic methods to prove the existence of outer codes with sufficiently small LCS.

We now define "harmonic" codewords.

Definition 12. Given an alphabet $\Sigma = \{1, ..., q\}$ and a block length qL, the *harmonic codeword of amplitude* A is defined as

$$f_A := (1^A 2^A ... q^A)^{L/A},$$

where A divides L, and x^A denotes the string x repeated A times.

The key property of these codewords is that if $A \ll B$, then a long common subsequence of f_A and f_B must have relatively large span. This is easiest to see when $A = 1, B > A$ (Figure 2). To have a small span in f_B, the common subsequence must have long runs of the same letter. However, this will result in a long span in f_A, since every subsequent "match" will required you to skip $q+1$ indices. Therefore for $A \ll B$, one might expect that for for any common subsequence s, we have $span_s(f_A, f_B) \approx (q+1)len(s)$. The following lemma makes this precise (note that appearance of the $q+1$ factor).

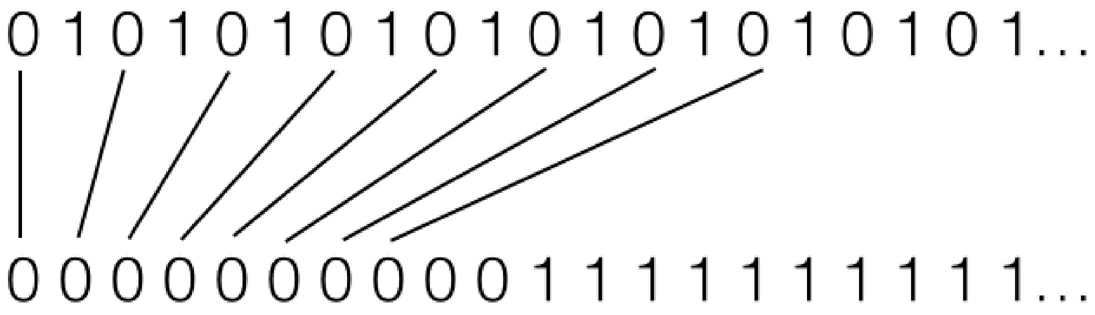


Figure 2: The top string is f_A, and the bottom string is f_B, for $A = 1, B = 10$. To have a short span in f_B, the common subsequence must have large span in f_A, and vice-versa.

Lemma 5.2. *Let $\Sigma = [q]$, and let f_A^∞ be the infinitely repeated word $(1^A 2^A ... q^A)^*$. Let A and B be natural numbers with $qA < B$, and suppose that s is a common subsequence of f_A^∞ and f_B^∞. Then*

$$span_s(f_A^\infty, f_B^\infty) \geq \left(q + 1 - \frac{qA}{B}\right) len(s) - 2(A + B). \tag{32}$$

Now we define our concatenation function. Assume our outer alphabet is $[Q]$, and we have an inner code of harmonic codewords with alphabet $[q]$, with $q < Q$. Let R (a parameter we can later determine) be such that R^{q-1} divides L, where kL is the length of each encoded symbol of the outer code. Then we let $\tau : [Q] \to [q]^{qL}$ be defined by

$$l \in [Q] \mapsto f_{R^{l-1}} \in [q]^{qL}. \tag{33}$$

We extend τ to entire words in $[Q]^n$ so that for $w \in [Q]^n$, $\tau(w) = (\tau(w_1), ..., \tau(w_2))$. Using this concatenation function, the authors establish the following critical bound, which relates the span of the concatenated code to the LCS of the outer code.

Lemma 5.3. *Let w_1 and w_2 be two words over $[Q]^n$, and let s be a common subsequence of $\tau(w_1)$ and $\tau(w_2)$. Then*

$$\begin{aligned} span_s(\tau(w_1), \tau(w_2)) \geq &\left(q + 1 - \frac{q}{R} - \frac{8R^{Q-1}}{L}\right) len(s) \\ &- 2Lq(q+1)LCS(w_1, w_2) - 16R^{Q-1}. \end{aligned} \tag{34}$$

We note that the proof of this fact relies on Lemma 5.2.

Using Lemma 5.3, we can choose parameters R and L to prove the existence of positive rate and high span concatenated codes, given the existence of outer codes with low LCS. Applying Lemma 5.1, we immediately translate this to a upper bound on the achievable LCS and thus a lower bound on the correctable fraction of deletions.

Theorem 5.4. *Let $\mathcal{C}^{out} \in [Q]^n$ be a code with LCS less than γn, and let $\mathcal{C}$ be the concatenated code given by applying the mapping τ defined above to $\mathcal{C}^{out}$ for some choice of $q \geq 2$ (with* *the parameters R and L depending only on Q, q and γ). Let $N = qnL$ be the block length of $\mathcal{C}$. Then for any $c_1, c_2 \in \mathcal{C}$ and any common subsequence s,*

$$span_s(c_1, c_2) \geq (q+1)len(s) - 4\gamma qN, \tag{35}$$

Proof. Let

$$R = \left\lceil \frac{2q}{\gamma} \right\rceil \text{ and } L = 16R^{Q-1} \left\lceil \frac{1}{\gamma} \right\rceil. \tag{36}$$

Plugging R and L into Lemma 5.3, we have that

$$span_s(c_1, c_2) \geq (q+1-\gamma)\, len(s) - 2(q+1)\gamma N - \gamma L \tag{37}$$

$$\geq (q+1)len(s) - 4\gamma qN. \tag{38}$$

□

Note that for this concatenated construction to give codes with rate bounded away from 0, we rely on the existence $\mathcal{C}^{out}$ with positive rate. This is proved in [2] using a random coding argument.

Corollary 2. *Let $\mathcal{C}$ be the concatenated code given by Theorem 5.4. For $c_1, c_2 \in \mathcal{C}$,*

$$LCS(c_1, c_2) \leq \left(\frac{2+4\gamma q}{q+1}\right) N \leq \left(\frac{2}{q+1} + 4\gamma\right) N, \tag{39}$$

or equivalently, $\mathcal{C}$ can correct a fraction

$$1 - \left(\frac{2}{q+1} + 4\gamma\right) \tag{40}$$

of deletions.

Proof. The proof is direct by applying Lemma 5.1 to the result of Theorem 5.4. □

Note that for $q = 2$ this already gives us codes correcting up to a 1/3 fraction of deletions. We refer the reader to [2] to see how the authors further optimize this scheme to obtain codes correcting a $\sqrt{2} - 1$ fraction.

6 Conclusion

In this report, we presented approaches to the worst-case deletion coding problem. We began with the classic approach of Levenshtein [5, 6], which got us to the asymptotically optimal Varshmamov-Tenengoltz 1-deletion codes. We then explored modern approaches to how we might be able to correct more than just single deletions, reviewing both Kulkarni's novel hypergraph-based analysis [4] and an approach by Bukh, Guruswami, and Håstad based using concatenated codes which to our knowledge corrects the highest constant fraction of deletions in current literature [2].

A number of open problems remain in the field of worst-case deletion correction. A couple we will highlight here are fully characterizing the size of d deletion balls for $d > 1$, achieving the minimum number of redundancy bits for a constant number of deletions, and further closing the gap between the current best fraction of correctable deletion and the current best upper bound of 1/2.

Acknowledgements

We would like to thank Mary Wootters for introducing us to the topic of error correcting codes and for valuable feedback during our study of deletion codes. We would also like to thank Noah Shutty for advice on research directions.

References

[1] Joshua Brakensiek, Venkatesan Guruswami, and Samuel Zbarsky. “Efficient Low-Redundancy Codes for Correcting Multiple Deletions”. In: *CoRR* abs/1507.06175 (2015). arXiv: `1507.06175`. URL: `http://arxiv.org/abs/1507.06175`.

[2] Boris Bukh and Venkatesan Guruswami. “An improved bound on the fraction of correctable deletions”. In: *CoRR* abs/1507.01719 (2015). arXiv: `1507.01719`. URL: `http://arxiv.org/abs/1507.01719`.

[3] Ryan Gabrys and Frederic Sala. “Codes Correcting Two Deletions”. In: *CoRR* abs/1712.07222 (2017). arXiv: `1712.07222`. URL: `http://arxiv.org/abs/1712.07222`.

[4] Ankur A. Kulkarni and Negar Kiyavash. “Non-asymptotic Upper Bounds for Deletion Correcting Codes”. In: *CoRR* abs/1211.3128 (2012). arXiv: `1211.3128`. URL: `http://arxiv.org/abs/1211.3128`.

[5] V. I. Levenshtein. “Binary Codes Capable of Correcting Deletions, Insertions and Reversals”. In: *Soviet Physics Doklady* 10 (Feb. 1966), p. 707.

[6] V.I. Levenshtein. “On perfect codes in deletion and insertion metric”. In: *Discrete Math. Appl.* 2 (3), pp. 241–258. URL: `https://doi.org/10.1515/dma.1992.2.3.241`.

[7] George S. Lueker. “Improved Bounds on the Average Length of Longest Common Subsequences”. In: *J. ACM* 56.3 (May 2009), 17:1–17:38. ISSN: 0004-5411. DOI: `10.1145/1516512.1516519`. URL: `http://doi.acm.org/10.1145/1516512.1516519`.

[8] Michael Mitzenmacher. “A survey of results for deletion channels and related synchronization channels”. In: *Probab. Surveys* 6 (2009), pp. 1–33. DOI: `10.1214/08-PS141`. URL: `https://doi.org/10.1214/08-PS141`.

[9] Daniel S. Hirschberg and Mireille Regnier. “Tight Bounds on the Number of String Subsequences”. In: *Journal of Discrete Algorithms* 1 (June 2001).

[10] Jin Sima, Netanel Raviv, and Jehoshua Bruck. “Two Deletion Correcting Codes from Indicator Vectors”. In: *CoRR* abs/1806.09240 (2018). arXiv: `1806.09240`. URL: `http://arxiv.org/abs/1806.09240`.

[11] Neil J. A. Sloane. “On single-deletion-correcting codes”. In: *Ohio State University* 2 (2001), pp. 273–291.

[12] D. A. Spielman. “Linear-time encodable and decodable error-correcting codes”. In: *IEEE Transactions on Information Theory* 42.6 (Nov. 1996), pp. 1723–1731. ISSN: 0018-9448. DOI: `10.1109/18.556668`.