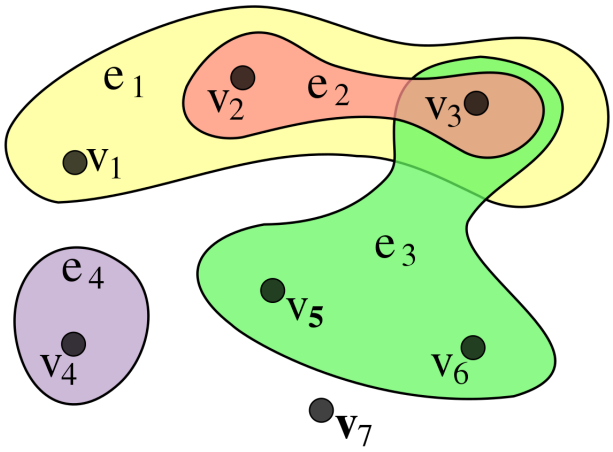
Upper Bounds for Deletion Correcting Codes

Bachelor Thesis

J.D.H. Ladd (5090350)



Thesis Committee: dr.ir. J.H. Weber (supervisor)
dr. J.A.M. de Groot

TU Delft Delft University of Technology

Department of Applied Mathematics
Delft University of Technology
Delft, The Netherlands
July 2, 2023

Abstract

The world can't keep up with its own data and therefore needs new ways to store it. Research is being done into new types of experimental data storage like DNA sampling or racetrack memory, which could completely revolutionize the way storage works. However, such storage media are prone to certain type of errors like deletion errors where information is lost. Consequently, there has been a surge in interest in developing deletion correcting codes to address this challenge. We are interested in the maximal size of a code correcting up to a certain number of deletions. Nevertheless, constructing these codes can be quite intricate and therefore we are often limited to working with bounds.

In this thesis, we will study and evaluate upper bounds for these codes. We will compare bounds that were derived analytically to bounds that have been computed with the use of linear programs. We will compute upper bounds for 1-, 2- and 3-deletion correcting codes. Moreover, we will show how to compute these bounds with a linear program, by posing the problem of finding a largest deletion correcting code as finding the transversal number on a hypergraph. It will become clear that the bounds computed through linear programming are much stronger than any bound derived analytically.

Terms and Definitions

Term	Meaning
Communication channel	Particular type of media through which a message is sent and received.
Symbol	Element in $\mathcal{A}_q$.
Alphabet	Finite set of symbols, i.e., $\mathcal{A}_q$.
String	A sequence of symbols.
Deletion	Removal of a symbol in a string
Substring	The sequence obtained after deletion(s) from a string.
Run	Maximal contiguous sequence of a string with identical symbols.
Code	Subset of $\mathcal{A}_q^n$.
Codeword	Element of a code.
Deletion set	Set of substrings obtained by s deletions from a string in $\mathcal{A}_q^n$.
Deletion correcting code	Subset of $\mathcal{A}_q^n$ with the property that the deletion sets are pairwise disjoint.
Optimal	Largest size
Graph	A pair of sets (V, E) where V is non-empty and E is a subset of the set $\{\{u, v\} : u, v \in V, u \neq v\}$.
Hypergraph	Generalization of a graph where an edge is any subset of the vertices.
Hyperedge	Edge of a hypergraph, i.e. a subset of vertices.
Matching	Set of pairwise disjoint hyperedges.
Transversal	Subset of the vertices such that every hyperedge is intersected.
Non-adjacent	If $uv \notin E$ for two vertices u, v in a graph $G = (V, E)$.
Independent set	Set of pairwise non-adjacent vertices in a graph.

Notation	Meaning
$\mathbb{N} := \{1, 2, 3, \ldots\}$	The positive integers.
$q \in \mathbb{N}_{\geq 2}$	The alphabet size.
$s \in \mathbb{N}$	The number of deletions.
$n \in \mathbb{N}$	The length of a string.
$\mathcal{A}_q$	The finite q-ary alphabet $\{0, 1, \ldots, q-1\}$.
$\mathcal{A}_q^n$	Set of all q-ary strings of length n.
x	String in $\mathcal{A}_q^n$.
$D_s(x)$	The s-deletion set of a string x.
$\mathcal{C}_{q,s,n}$	A q-ary code with strings of length n capable of correcting s deletions.
$\mathcal{C}^*_{q,s,n}$	A s-deletion correcting code of maximal size with q-ary alphabet and string length n.
$\lvert\mathcal{C}^*_{q,s,n}\rvert$	The cardinality of $\mathcal{C}^*_{q,s,n}$.
$VT_a(n)$	The Varshamov-Tenengolts code
$\mathcal{R}_q^n$	Repetition code, i.e, $\{\underbrace{00\cdots0}_{\text{n}}, \underbrace{11\cdots1}_{\text{n}}, \cdots, \underbrace{(q-1)(q-1)\cdots(q-1)}_{\text{n}}\}$.
$\mathcal{H} := (V, E)$	Hypergraph with vertices V and hyperedges E.
$\nu(\mathcal{H})$	Cardinality of a matching of maximum size among all matchings.
$\tau(\mathcal{H})$	Minimum cardinality of a transversal.
$\alpha(G)$	Largest size of an independent set.
ILP	Integer linear program.
LP	Linear program.
NP-hard	Complexity class such that the problems in this class cannot be solved in polynomial time.

Contents

Chapter 1

Introduction

The digital era offers us many things: technology like artificial intelligence, blockchain, 5G and the cloud. One of the perks that the digital era brings is that it allows for a globalized world. Never before has the world been so connected and there are over 8000 data centres spread around the globe [11]. Sending, storing and retrieving data has never been easier. However, communication channels and digital data storage aren't without errors [28]. These errors can happen while sending information or while retrieving stored information. The result is then a partial loss of information or in some cases, the information is altered. Take for example the Qantas flight 72 in 2008. An alteration of the Angle of Attack[1] information caused a near plane crash, injuring over a hundred people [3]. Another example of data being altered is the so called *bit flip* in a Belgium election in 2003. Here, a cosmic ray caused a 0 in a string to turn to a 1, attributing over 4000 votes to a certain candidate [27]. These types of errors are called *substitution* errors and are well researched.

In this thesis, we will focus only on a specific type of error called a *deletion* error. This is a type of error where a deletion takes place, i.e., a symbol gets deleted from a string. This can look like:

$$110100 \to 11000 \quad \text{or} \quad 110100 \to 010,$$

where a single deletion and three deletions occur respectively. Deletion errors have been far less studied and this is due to two reasons. The first one is that in communication channels they occur far less often than substitution errors and the second reason is that a *synchronization method* was used, if they did occur. For this method, there often is a built in 'check'-sequence in a codeword, like 0101. An example is that a sender sends a codeword with 0101 as the synchronization pattern. This is a useful way of checking if an error occurred when the receiver doesn't know the length of the message. If a deletion occurs, i.e.,

$$1100|0010|\underbrace{0101}_{\text{sync}}|0110\cdots \to 1100|0100|\underbrace{1010}_{\text{error}}|110\cdots,$$

we can use the synchronization pattern to check if a deletion has occurred. The symbol in red gets deleted and because of that all the other symbols move one spot to the left. Now the synchronization pattern doesn't read 0101 but 1010 and therefore the receiver of the information knows an error has taken place and can ask the sender to resend the information.

Recently however, experimental data storage have caused a surge in popularity for deletion correcting codes due to the breakthroughs in this field. One of these new data storages is the use of DNA sampling [12],[30] and stores information as a DNA sequence. This can be done by encoding data, synthesizing it into a DNA sequence and then storing it in a cool and regulated environment and can be decoded at any given time to retrieve the stored data. What makes DNA data storage attractive is that it has a very high storage density: it can store 215 million GB in a single gram of DNA.

Another example is racetrack memory storage [7] and stores data as a magnetic pattern and promises to be a very inexpensive and efficient way of storing data as it is estimated that racetrack memory only needs 10 nanoseconds to read or write a bit, compared to 10,000,000 nanoseconds needed for a traditional hard drive.

The problem with these types of data storage, is that they are susceptible to errors like deletion errors. With the DNA-storage channel, the data is sampled as a set of sequences but the retrieved set is often corrupted due to sequence loss and symbol deletion. In racetrack memory, researchers often can't fully control the magnetic patterns and this in turn effects the reliability of reading and writing data. The high possibility of deletion

[1] The angle between the reference line on an airplane and the oncoming air.

errors is the reason why deletion corrected codes have attracted so much attention.

In practice, it is often quite difficult to construct deletion correcting codes and therefore we are often limited to working with bounds. Constructing lower and upper bounds for deletion correcting codes gives us information on the sizes of these codes for certain parameters and therefore brings us a step closer to the construction of such codes. So studying deletion correcting codes will bring us also a step closer to implementing these experimental types of data storage into our society and thereby transform the digital world.

Since the the publishing of *A Mathematical Theory of Communication* in 1948, by Claude Shannon [28] and therefore the creation of Information Theory, there have been multiple constructions of s-deletion correcting codes and resulting problems.

Let us first start with some published work on binary codes correcting up to a certain number of deletions. In [29], Sloane gives a detailed insight on the binary single-deletion channel and specifically focuses on the Varshamov-Tenengolts codes. Sloane also gives a conjecture that the $VT_a(n)$ codes are optimal for all n. These $VT_a(n)$ codes were proven to be able to correct a single deletion by Levenshtein in 1965 [22].

Besides the $VT_a(n)$ codes, there have been many papers on how to construct other codes capable of correction deletions. One of these constructions was by Helberg and Ferreira in [13] with the so called *Helberg code* and has a number theoretic construction. This was done by looking at the weight spectra and Hamming distance properties of single-deletion correcting codes and then extending these to codes that can correct multiple deletions. In [2], Abdel-Ghaffer *et al.* proved that this code construction indeed corrects multiple deletions.

For q-ary alphabets, bounds for s-deletion correcting codes were derived analytically by Levenshtein in [21] and [20] with the help of combinatorial arguments.

More recently, in [24] algebraic theory is used by Liu and Xing to construct deletion correcting codes and also construct bounds for these codes.

A more recent discipline of constructing q-ary s-deletion correcting codes and finding bounds is via graph theory and linear programming. In [6], Butenko *et al.* construct binary s-deletion correcting codes by translating this problem to a maximum independent set problem $\alpha(G)$ and then using a branch and bound algorithm.

In [10], Cullina *et al.* find upper bounds by translating this problem to a colouring of a graph and also give a code construction for the 2-deletion case.

In [17] Kulkarni and Kiyavash translate the problem of finding the largest s-deletion correcting code to finding a matching number on a hypergraph and then use duality theorems to find upper bounds (this is also the method we will use in this thesis).

In [8], Cullina and Kiyavash use packing arguments and represent a communication channel as a bipartite graph, where channel input corresponds to left vertices and channel output corresponds to right vertices. An edge connects an input to an output that can be produced from it, i.e., a string will have edges to all elements in its deletion set. With this they provide a bound that is better than Levenshtein's asymptotic bound from [19] whenever the number of deletions to be corrected is greater than the alphabet size q.

In [9], Cullina and Kiyavash use a bipartite graph presentation and then convert their sphere-packing arguments to linear programs.

One of the resulting problems when studying deletion correcting codes is the size of the deletion set $D_s(\cdot)$. The sizes of these sets are important for constructing bounds for deletion correcting codes and Mercier *et al.* present some nice theorems about the sizes of these sets [26]. A few years later, in [23] Liron and Langberg give improved bounds on the sizes of the deletion sets for arbitrary numbers of deletions. Stronger bounds for deletion sets can contribute to new bounds for deletion correcting codes, as can be seen in [20].

In this thesis, we will look at the problem of computing upper bounds for s-deletion correcting codes via the use of linear programming. In the remainder of the introduction, we will mathematically define a deletion correcting code and introduce some definitions for our methods. Finally we will present the goal of this thesis and explain the layout.

Before we go any further, we must acquaint ourselves with some useful definitions that will be relied heavily upon in this thesis.

1.1 Definitions and Introducing Methodology

Suppose Adam and Eve are in a hypothetical communication channel that incurs deletions, meaning Adam and Eve send each other messages but the received messages are subject to partial loss of information due to any number of deletions. Suppose they write their messages from the binary alphabet of length 2, i.e., $\{00, 01, 10, 11\}$ with the following meanings:

$$\begin{aligned} 00 &\rightarrow \text{No!} \\ 01 &\rightarrow \text{Don't eat it!} \\ 10 &\rightarrow \text{Eat it!} \\ 11 &\rightarrow \text{Yes!} \end{aligned}$$

Adam sees Eve picking up an apple and sends Eve the message 01 to tell her not to eat it. Since this communication channel allows for deletions to take place, Eve unfortunately receives the message 1. Since 1 is not of length 2, Eve knows an error took place and tries to figure out which message was sent by adding a 0 or a 1 to the left or right side of the received message. Eve figures out that the received message could have come from the following three codewords: $01, 10$ and 11. There is now a high probability that Eve will choose the wrong codeword, i.e., 10, telling her to eat the apple or 11, which can also be interpreted as an encouraging sign to eat the apple. With miscommunications like these, a banishment from Paradise does not seem unlikely...

Suppose they had used only words in the binary repetition code of length 2, i.e., $\mathcal{R}_2^2 = \{00, 11\}$. Now it doesn't matter if Eve receives a 0 or a 1, as these each descended only from one possibility. We say that $\mathcal{R}_2^2$ can correct a single deletion, but this is no coincidence and will become clear after we introduce two definitions.

First let us denote $\mathcal{A}_q = \{0, 1, \ldots, q-1\}$ as the *q-ary alphabet* and let $\mathcal{A}_q^n$ be the set of all q-ary strings of length n. A *substring* of x is formed by taking a subset of the symbols of x without altering their order. A run of $x = x_1 \ldots x_n \in \mathcal{A}_q^n$ is a maximal contiguous substring with identical symbols.

Definition 1 *The s-deletion set $D_s(x)$ of x is the set of substrings of x obtained by deletion of s symbols from x.*

Example 1: if $x = 0112 \in \mathcal{A}_3^4$, then $D_1(0112) = \{112, 012, 011\}$.
Example 2: if $x = 11111 \in \mathcal{A}_2^5$, then $D_3(11111) = \{11\}$.

Definition 2 *A s-deletion correcting code for $\mathcal{A}_q^n$ is a set $\mathcal{C} \subseteq \mathcal{A}_q^n$ with the property that the s-deletion sets of the codewords are pairwise disjoint. Such a code of maximal size is denoted as $\mathcal{C}_{q,s,n}^*$. The cardinality of a largest s-deletion correcting code is denoted as $|\mathcal{C}_{q,s,n}^*|$.*

By looking at the codewords in $\mathcal{R}_2^2$, we see that the deletion sets for $s = 1$ are disjoint ($D_1(00) = 0, D_1(11) = 1$ and $0 \cap 1 = \emptyset$) and therefore $\mathcal{R}_2^2$ is a single-deletion correcting code.

Furthermore, in this thesis we will make use of the following definitions taken from [4] and [5] for our methodology. As we will see in Chapter 3, we will pose the problem of finding an upper bound on $|\mathcal{C}_{q,s,n}^*|$ as a minimum transversal. But we need knowledge of hypergraphs and matchings in order to do so.

Definition 3 *A hypergraph $\mathcal{H}$ denoted by $\mathcal{H} = (V; E = (e_i)_{i \in I})$ on a finite set V is a family $(e_i)_{i \in I}$ of subsets of V called hyperedges. $I = \{1, \ldots, m\}$, where m is the number of hyperedges.*

Definition 4 *A matching is a set of pairwise disjoint hyperedges of $\mathcal{H}$. A maximum matching is a matching of maximum size among all matchings. The matching number $\nu(\mathcal{H})$ of $\mathcal{H}$ is the cardinality of a maximum matching.*

Definition 5 *Let $V(e)$ be the set of vertices covered by a hyperedge e. A set $\mathcal{B} \subseteq V$ is a transversal if it meets every hyperedge, i.e., $\forall e \in E : \mathcal{B} \cap V(e) \neq \emptyset$. The minimum cardinality of a transversal is the transversal number, denoted by $\tau(\mathcal{H})$.*

1.2 Goal of the Thesis

In this thesis, we aim to study and evaluate upper bounds for s-deletion correcting codes. We will look at existing bounds and also compute bounds via linear programming. By the end of the thesis, we will have a comprehensive understanding of different bounds for $|\mathcal{C}_{q,s,n}^*|$, how to compute these values and know what the advantages and disadvantages are for certain bounds.

1.3 Organisation of the Thesis

This thesis is organized as follows. In Chapter 2 we will highlight some upper bounds for $|\mathcal{C}^*_{q,s,n}|$ and talk about how these were derived. In Chapter 3 we will first look at different numerical values for one of the bounds from the previous chapter and then work out how we can use linear programming to compute bounds for $|\mathcal{C}^*_{q,s,n}|$. In Chapter 4 we will show the results of the linear programs and compare these to the bounds that we have seen in Chapter 2. Finally, in Chapter 5 we will make our conclusions and discuss possible directions for future work.

Chapter 2

Classification of Bounds

Now that we have a better understanding of the research that has been done, we want to highlight a few bounds from some of the published papers and use these to compare them with the results from the linear programs in the next chapter. We can split these bounds into two groups: bounds that have been derived analytically and bounds that have been derived with the help of linear programs. In general, the analytic bounds are older and the so called *LP-bounds* are more recent.

But first, we will show a few code constructions that we will use as a lower bound for the cardinalities of $|\mathcal{C}^*_{q,s,n}|$.

2.1 Some Code Constructions

For the binary single-deletion case, the $VT_a(n)$ codes are the largest known codes up to date. These codes have the following property: if $x = (x_1 \dots x_n)$ is a codeword, then it must satisfy

$$\sum_{i=1}^{n} i x_i \equiv a \mod n+1.$$

The sizes of the codebooks depend on the value of a, as can be seen in Table 2.1 below.

Table 2.1: *The cardinalities of $VT_a(n)$ for different values of a and n.*

n	$a=0$	$a=1$	$a=2$	$a=3$	$a=4$	$a=5$	$a=6$	$a=7$	$a=8$	$a=9$	$a=10$
1	1	1									
2	2	1	1								
3	2	2	2	2							
4	4	3	3	3	3						
5	6	5	5	6	5	5					
6	10	9	9	9	9	9	9				
7	16	16	16	16	16	16	16	16			
8	30	28	28	29	28	28	29	28	28		
9	52	51	51	51	51	52	51	51	51	51	
10	94	93	93	93	93	93	93	93	93	93	93

As we can see, the codes with $a = 0$ have the largest cardinality. To see why this is true, one can have a look at Theorem 2.2 from [29]. This states that

$$|VT_a(n)| = \frac{1}{2(n+1)} \sum_{d \mid n+1, d \text{ odd}} \phi(d) \frac{\mu(\frac{d}{(d,a)})}{\phi(\frac{d}{(d,a)})} 2^{(n+1)/d},$$

where $\phi(\cdot)$ is Euler's totient function, $\mu(\cdot)$ is the Möbius function and (d, a) is $\gcd(d, a)$. If $a = 0$, we get

$$\mu(\frac{d}{(d,a)}) \leq 1 = \mu(1) = \mu(\frac{d}{(d,0)}),$$
$$\phi(\frac{d}{(d,a)}) \geq 1 = \phi(1) = \phi(\frac{d}{(d,0)}).$$

So

$$|VT_a(n)| \leq \frac{1}{2(n+1)} \sum_{d|n+1, d \text{ odd}} \phi(d) 2^{(n+1)/d} = |VT_0(n)|.$$

As mentioned before, in 1965, Levenshtein gave a clever decoding algorithm to prove that the Varshamov-Tenengolts codes $VT_a(n)$ [31] are single-deletion correcting codes [22]. We can now use these codes as a lower bound on the size of a largest binary single-deletion correcting code. This is because we know what the cardinality is of this code, so there must be at least as many codewords in $|\mathcal{C}^*_{2,1,n}|$.

For 2 deletions in the binary alphabet, we will use code sizes from [16] to compare with the other bounds. These codes have been found by posing the problem of finding a largest 2-deletion correcting code as a maximum independent set problem.

For larger alphabets in the 2 deletion case we will use the code sizes from [18] as a comparison. These were found by generalizing Helberg's construction of binary 2-deletion correcting codes to non-binary alphabets.

2.2 Analytical Bounds

We will now give two nonasymptotic upper bounds provided by Levenshtein. The first nonasymptotic upper bound is derived from a combinatorial argument

$$|\mathcal{C}^*_{q,1,n}| \leq \frac{q^{n-1} + (n-2)q^{n-2} + q}{n} \tag{2.1}$$

and was found in 1992 [21]. This bound was found by constructing a subset of $\mathcal{C}_{q,s,n}$ where the codewords in this subset have certain properties and applying combinatorics to bound the cardinality of this subset.

The second nonasymptotic upper bound is

$$|\mathcal{C}^*_{q,s,n}| \leq \frac{q^{n-s}}{\sum_{i=0}^{s} \binom{r-s+1}{i}} + q \sum_{i=0}^{r-1} \binom{n-1}{i} (q-1)^i \tag{2.2}$$

and was provided in 2002 [20]. Here r is an integer satisfying $1 \leq s \leq r+1 \leq n$ where r is chosen such that the right hand side of (2.2) is minimized. The proof of (2.2) uses the same method as in [19]. If x is a codeword in $\mathcal{C}^*_{q,s,n}$ and $||x||$ is the number of runs of x, then Levenshtein splits $\mathcal{C}^*_{q,s,n}$ into

$$\{x \in \mathcal{C}^*_{q,s,n} : ||x|| \geq r+1\} + \{x \in \mathcal{C}^*_{q,s,n} : ||x|| \leq r\}.$$

Levensthein then bounds both of these sets as in [19] but takes a new and improved lower bound on the cardinality of a deletion set of x into account that was proven by Hirschberg in [14] to obtain the result.

2.3 LP-Bounds

Numerical Bounds are a relatively new discipline of deletion correcting codes and often make use of graph theory and optimization. Even though in this thesis we are mainly interested in the numerical bounds, Kulkarni and Kiyavash have also derived a closed form expression as an upper bound [Theorem 3.1] in [17] by giving an analytical solution to a fractional transversal. The upper bound is

$$|\mathcal{C}^*_{q,1,n}| \leq \frac{q^n - q}{(q-1)(n-1)}. \tag{2.3}$$

This expression was found by giving a solution to the fractional transversal and then calculating the weight of this solution. In the next chapter we will compare (2.3) to the other bounds.

Chapter 3

Methodology

In this chapter, we will discuss the methods used in this thesis and analyse the results. As could be seen in the previous section, there exist multiple bounds for the sizes of deletion correcting codes. Also, we have seen that there are multiple approaches to arriving at said bounds. As mentioned before, the goal of this thesis is to understand and numerically study existing bounds.

3.1 Numerical Analysis of (2.2)

Let's have a closer look at the analytical bound found by Levenshtein in 2002 [20]. Namely,

$$|\mathcal{C}^*_{q,s,n}| \leq \frac{q^{n-s}}{\sum_{i=0}^{s}\binom{r-s+1}{i}} + q\sum_{i=0}^{r-1}\binom{n-1}{i}(q-1)^i, \tag{3.1}$$

where $1 \leq s \leq r+1 \leq n$. Since this is not a closed form expression, it is not clear for which r the upper bound is the strongest. In [17], it is heuristically argued that $r \approx \frac{n}{2}$ is asymptotically optimal for the binary single deletion case ($q = 2$, $s = 1$). Below is a table with the numerical values of the bound when plugging in different values of r and n.

Table 3.1: *The rounded down values U of the upper bound given by (3.1), for different values of n and r.*

$n=4$		$n=5$		$n=6$		$n=7$		$n=8$		$n=9$		$n=10$		$n=11$		$n=12$		$n=13$	
r	U	r	U	r	U	r	U	r	U	r	U	r	U	r	U	r	U	r	U
1	6	1	10	1	18	1	34	1	66	1	130	1	258	1	514	1	1026	1	2050
2	10	2	15	2	22	2	35	2	58	2	103	2	190	2	363	2	706	2	1391
3	16	3	26	3	40	3	60	3	90	3	138	3	220	3	368	3	646	3	1182
		4	33	4	58	4	96	4	153	4	237	4	362	4	556	4	873	4	1417
				5	67	5	124	5	219	5	368	5	597	5	942	5	1465	5	2270
						6	135	6	258	6	474	6	837	6	1422	6	2340	6	3757
								7	270	7	526	7	996	7	1824	7	3228	7	5532

When looking at the table, we notice that for $4 \leq n \leq 7$, $r = 1$ gives the tightest upper bound. Only for $n \geq 8$, the value of r that gives the tightest upper bound becomes higher, although it only moves up one position to $r = 2$ and to $r = 3$ for $n \geq 12$. For small n these results make sense since the second term on the right hand side of (3.1) is relatively contributing heavier to a higher (and therefore weaker) value for the upper bound. Only when plugging in large values for n, is the value of r that gives the smallest upper bound getting close to $r \approx \frac{n}{2}$.

In Table 3.2, the values of r are given that minimize the upper bound given in equation (3.1). We see that as n increases, that the value of r slowly grows to $r \approx \frac{n}{2}$. We have only computed this bound for n up to 1000

Table 3.2: *The values of r such that equation (3.1) is minimized for the binary single-deletion case, for large n.*

n	250	500	1000
r	96	205	432

since this approaches the limit of a value that Matlab will return before returning the value `Inf` (This happens roughly when $n = 1024$, since in Matlab, 2^{1024} yields `Inf`).

3.2 A linear programming approach

3.2.1 Introduction

For a communication channel where for example we start with an element from one set $\mathcal{A}_q^n$ and end with an element from another set $\mathcal{A}_q^{n-s}$ by only removing s symbols, we talk of a deletion channel. Unfortunately, with deletion channels, we often are unable to produce neat sphere-packing arguments to find bounds, like we can do with substitution errors and Hamming codes. Instead, we must get our hands dirty and make use of linear programs to produce bounds.

We will translate the problem of finding a largest s-deletion correcting codes to finding a matching number and transversal number and these lie naturally in linear programming. The actual solution to these problems are NP-hard [15]. So in order to compute bounds, we use relaxations for our linear programs: here, the feasible regions can be computed in polynomial time. Even though the feasible regions can be computed in polynomial time, deletion errors still act on an exponentially large input space. This exponential input space will become clear later on.

3.2.2 Linear Programming Formulation

We will make use of hypergraphs ($\mathcal{H} = (V, E)$) to formulate the problem of finding a largest s-deletion correcting code. In Chapter 1 we have seen the mathematical definition of a hypergraph, but in other words this is just a is a generalization of a graph in which an edge is an arbitrary subset of the vertices and thus called a hyperedge.

We will work with the following hypergraph, just like in [17]:

$$\mathcal{H}_{q,s,n}^D = (\mathcal{A}_q^{n-s}, \{D_s(x) | x \in \mathcal{A}_q^n\}). \tag{3.2}$$

Here, the vertices correspond to strings in $\mathcal{A}_q^{n-s}$ and the hyperedges correspond to all the substrings one can create by deleting s symbols from a string in $\mathcal{A}_q^n$. As we have seen before, for a s-deletion correcting code, we have that the deletion sets $D_s(x)$ must be pairwise disjoint for $x \in \mathcal{A}_q^n$. Thus a s-deletion correcting code has disjoint hyperedges in $\mathcal{H}_{q,s,n}^D$, hence it corresponds to a matching in $\mathcal{H}_{q,s,n}^D$. Clearly, the size of a largest s-deletion correcting code, $|\mathcal{C}_{q,s,n}^*|$, corresponds to a maximum matching $\nu(\mathcal{H}_{q,s,n}^D)$ in our hypergraph.

It is generally known that the dual of a matching $\nu(\mathcal{H}_{q,s,n}^D)$ is a transversal $\tau(\mathcal{H}_{q,s,n}^D)$, but for a quick proof one can have a look at [17]. Because of this, the matching and transversal problems now satisfy weak duality, which states that the value of the maximization problem cannot be greater than the value of the minimization problem [1], i.e.,

$$\nu(\mathcal{H}_{q,s,n}^D) \leq \tau(\mathcal{H}_{q,s,n}^D). \tag{3.3}$$

This formulation is great for finding upper bounds, since any solution to the transversal number acts as an upper bound on the matching number and thus is an upper bound for $|\mathcal{C}_{q,s,n}^*|$.

Now that we have established this relationship, we will for now focus on the finding the transversal number $\tau(\mathcal{H}_{q,s,n}^D)$. We can formulate the matching number as an integer linear program [25], hence we can also formulate the transversal as an integer linear program [1] and get the following:

$$\tau(\mathcal{H}_{q,s,n}^D) = \min \sum_{x \in \mathcal{A}_q^{n-s}} w(x) \tag{3.4}$$

$$\text{s.t.} \sum_{x \in D_s(y)} w(x) \geq 1 \qquad \forall y \in \mathcal{A}_q^n \tag{3.5}$$

$$w(x) \in \{0, 1\} \quad \forall x \in \mathcal{A}_q^{n-s}. \tag{3.6}$$

The variables are $w(x), x \in \mathcal{A}_q^{n-s}$, where $w(x)$ is a binary variable such that

$$w(x) = \begin{cases} 1 & \text{if x is included in the transversal} \\ 0 & \text{otherwise.} \end{cases}$$

Also, for a transversal number we are looking for the smallest subset $\mathcal{B} \subset \mathcal{A}_q^{n-s}$ that intersects every hyperedge in $\{D_s(x)|x \in \mathcal{A}_q^n\}$. Hence we minimize the objective function. For a hyperedge $e_i \in \{D_s(x)|x \in \mathcal{A}_q^n\}$, the sum $w(x)$ over all the vertices x that are covered by e_i must be at least 1 which explains the constraint.

To see why a transversal number acts as an upper bound on the cardinality of a deletion correcting code, we can take another look at the linear program formulated in (3.4)-(3.6). The transversal number seeks to minimize the number of vertices that are needed to cover every hyperedge. In other words, it seeks to minimize the number of x in $\mathcal{A}_q^{n-s}$ such that every deletion set created by a string in $\mathcal{A}_q^n$ has has at least one of these x in it. If this number is τ, then this means that there can be at most τ disjoint deletion sets, i.e., at most τ codewords a s-deletion correcting code.

But if one has any knowledge of linear programming, the thought will come to mind that there is a way to find a better bound on $\nu(\mathcal{H}_{q,s,n}^D)$, namely, working with LP-relaxations. For an ILP, the feasible points are also feasible points of the LP-relaxation. To see this, suppose that F is the feasible region of the LP-relaxation. Then the feasible region of the ILP is precisely $\mathbb{Z}^n \cap F$. Because of this, the transversal number is no less than the fractional transversal number, denoted as $\tau^*(\mathcal{H}_{q,s,n}^D)$. We get the following inequality:

$$\tau^*(\mathcal{H}_{q,s,n}^D) \leq \tau^(\mathcal{H}_{q,s,n}^D). \tag{3.7}$$

and since $\tau^*(\mathcal{H}_{q,s,n}^D)$ is an LP-relaxation, it satisfies strong duality and therefore is equal to the LP-relaxation of the maximum matching [1]. Combining this knowledge with (3.3) and (3.7), we get

$$\nu(\mathcal{H}_{q,s,n}^D) \leq \nu^*(\mathcal{H}_{q,s,n}^D) = \tau^*(\mathcal{H}_{q,s,n}^D) \leq \tau^(\mathcal{H}_{q,s,n}^D). \tag{3.8}$$

Another benefit of the linear program relaxation is that it can be computed in polynomial time and therefore we can compute the fractional transversal number to attain an upper bound for $|\mathcal{C}_{q,s,n}^*|$. Even though in this thesis we will use the fractional transversal number, in (3.8) we see that we also could have computed a fractional matching number to obtain an upper bound on the cardinality of a largest s-deletion correcting code.

As was mentioned above, we will focus only on the fractional transversal problem to avoid dealing with a NP-hard problem. The main reason for this is that a matching uses the insertion sets of a string x [17], i.e., all the strings that can be created by *inserting* s symbols in x and we have not defined this in our thesis. This has one big benefit: transversals have q^{n-s} variables in the objective function, whereas a matching will have q^n variables in the objective function and this allows for much compacter programs. We can write a fractional transversal as follows.

$$\tau^*(\mathcal{H}_{q,s,n}^D) = \min \sum_{x \in \mathcal{A}_q^{n-s}} w(x) \tag{3.9}$$

$$\text{s.t.} \sum_{x \in D_s(y)} w(x) \geq 1 \qquad \forall y \in \mathcal{A}_q^n \tag{3.10}$$

$$w(x) \geq 0 \qquad \forall x \in \mathcal{A}_q^{n-s}. \tag{3.11}$$

Notice that no minimizing $w(x)$ will be greater than 1, so the fractional transversal problem is equivalent to the integer transversal problem. The only difference is that the fractional transversal problem allows the variable $w(x)$ to take any value between 0 and 1, since we got rid of the constraint that $w(x)$ must be either 0 or 1.

3.2.3 A Worked Out Example

In order to better understand how $\tau^*(\mathcal{H}_{q,s,n}^D)$ is computed, we have worked out an example below. To avoid any intricateness, we pick $q = 2$, $s = 1$ and $n = 3$. So we will be working in $\mathcal{A}_2^3$, i.e. the binary codes of length 3 for the single-deletion case. Let's first write down all possible strings.

$$\mathcal{A}_2^3 = \{000,\ 101,\ 001,\ 110,\ 010,\ 011,\ 100,\ 111\},$$

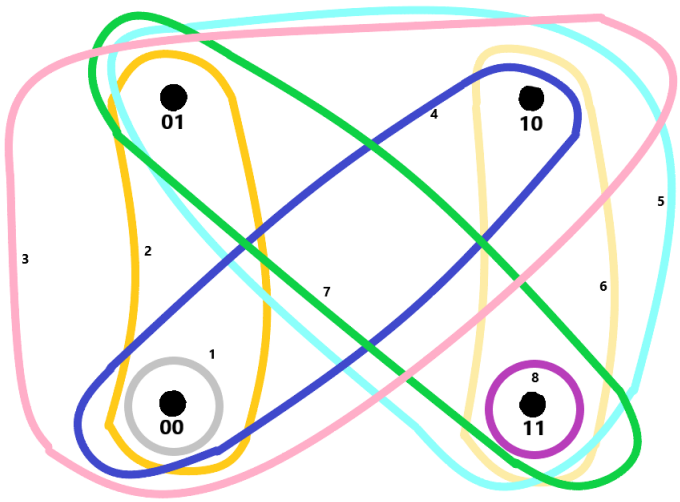
$$\mathcal{A}_2^{3-1} = \{00,\ 01,\ 01,\ 11\}.$$

The next step is to compute the deletion sets $D_1(x)$, for $x \in \mathcal{A}_2^3$.

$$
\begin{aligned}
1 &: D_1(000) = \{00\},\\
2 &: D_1(001) = \{00,\ 01\},\\
3 &: D_1(010) = \{00,\ 01,\ 10\},\\
4 &: D_1(100) = \{00,\ 10\},\\
5 &: D_1(101) = \{01,\ 10,\ 11\},\\
6 &: D_1(110) = \{10,\ 11\},\\
7 &: D_1(011) = \{01,\ 11\},\\
8 &: D_1(111) = \{11\}.
\end{aligned}
$$

Recall that for the hypergraph $\mathcal{H}_{2,1,3}^D = (\mathcal{A}_2^{3-1}, \{D_1(x) | x \in \mathcal{A}_2^3\})$, the vertices correspond to the strings in $\mathcal{A}_2^{3-1}$ and we conclude that there must be 4 vertices. The hyperedges correspond to strings in $\mathcal{A}_2^3$ so we see that there are 8 hyperedges. A quick visualisation of $\mathcal{H}_{2,1,3}^D$ yields:

Figure 3.1: *A visualisation of the graph $\mathcal{H}_{2,1,3}^D$. The edges are numbered and match the corresponding deletion sets above.*



We can now write out the objective function and the constraints using the 8 deletion sets. We get:

$$
\begin{aligned}
\text{minimize z} = & w(00) + w(01) + w(10) + w(11)\\
\text{s.t.}\quad & w(00) \geq 1\\
& w(00) + w(01) \geq 1\\
& w(00) + w(01) + w(10) \geq 1\\
& w(01) + w(10) + w(11) \geq 1\\
& w(10) + w(11) \geq 1\\
& w(01) + w(11) \geq 1\\
& w(11) \geq 1\\
& w(00), w(01), w(10), w(11) \geq 0.
\end{aligned}
$$

Since this linear program is small enough, we can now use an ordinary LP-solver and fill in the constraints manually. This will then return an objective function value:

$$z = 2, \quad w(00) = 1,\ w(01) = 0,\ w(10) = 0,\ w(11) = 1.$$

We now know that a binary single-deletion correcting code with words of length $n = 3$ has a cardinality of at most 2. This knowledge can then be used to construct such a codebook, perhaps with the knowledge of a known lower bound (note that the optimal codebook for this case has indeed cardinality 2 [29]). On a side note, the

correctness of the objective function value can be easily checked by looking at Figure 3.1 since we want to find the minimum number of vertices that intersect every hyperedge. After checking, we see that the vertices **00** and **11** satisfy this.

Another way to interpret why this transversal number acts as an upper bound on $|\mathcal{C}^*_{2,1,3}|$ is the following: the linear program has now minimized the number of vertices such that every hyperedge has one of these vertices in it. In other words, it has given the minimum number of substrings in $\mathcal{A}_2^{3-1}$ needed to cover every deletion set made from the strings in $\mathcal{A}_2^3$. Since this number is 2, intuitively there are at most 2 disjoint deletion sets, meaning there are at most two strings of length $n = 3$ in a single-deletion correcting code.

Chapter 4

Results

In the previous example, we worked with parameters $q = 2$ and $n = 3$. This lead to $2^3 = 8$ constraints and $2^{3-1} = 4$ variables in the objective function. For the single-deletion case if we look at the corresponding hypergraph $\mathcal{H}^D_{q,s,n} = (\mathcal{A}_q^{n-s}, \{D_s(x)|x \in \mathcal{A}_q^n\})$, we see that we must have q^{n-1} vertices and q^n hyperedges in general. This leads to q^{n-1} variables in our linear program and q^n constraints. When using software to solve this linear program, the input is a q^n x q^{n-1} matrix and it becomes clear that we have an exponentially large input space. In general we get an input of q^n x q^{n-s} for s deletions.

4.1 Solutions to the Fractional Transversal Numbers

On the next page we show Tables 4.1a, 4.1b and 4.1c with the values for $\tau^*(\mathcal{H}^D_{q,1,n})$, $\tau^*(\mathcal{H}^D_{q,2,n})$ and $\tau^*(\mathcal{H}^D_{q,3,n})$ respectively. As one can see, the tables look a little empty. This is due to the 'limited' computing capacity of the laptop used in this thesis.[1] On a side note, this will come back in the discussion. Still we we can show some new results, namely the columns for $q = 6$ and $q = 7$. The Matlab code used to find these results can be found in the Appendix, and this can be adapted to any $q-$ary alphabet and any number of deletions.

When we compare the Tables 4.1a, 4.1b and 4.1c, we expect to see the values of the fractional transversals decrease as the number of deletions increase. This is because there will be less codewords that we can put in a 2-deletion codebook since there will be less strings that will be able to satisfy the property of disjoint deletion sets and even less strings for three deletions. Another explanation for this is that the sizes of the deletion sets are now at most the binomial $\binom{n}{2}$ for the 2-deletion case and at most $\binom{n}{3}$ for the 3-deletion case. This is because there are $\binom{n}{2}$ ways to delete two symbols from a string of length n and $\binom{n}{3}$ ways to delete three symbols from a string of length n. Since the deletion sets are larger, we can automatically expect less disjoint deletion sets for the same set $\mathcal{A}_q^n$.

The next thing to comment on is the horizontal trend in each table: when comparing the upper bounds for different q, it makes sense that the values of the fractional transversal are larger for higher q and the same n. This is because the alphabets are larger and therefore there are more words to 'choose' from.

Also, we have left out the values of the upper bounds for $n \leq 2$ for Tables 4.1a and 4.1b and $n \leq 3$ in Table 4.1c, as we consider these trivial. If $n = 1$, then a deletion correcting code can only allow one codeword to be in it, since we retrieve an empty set after any number of deletions. If $n = 2$, then for the 2-deletion case we again retrieve an empty set and therefore can only have one codeword by the same reasoning as before.

In general, if $n = s$, $\mathcal{C}^*_{q,s,n}$ can only have one codeword in it.

If $n = 2$ for the binary 1-deletion case, note that there are only two strings in $\mathcal{A}_2^{2-1}$ so there are at most two disjoint deletion sets (0 and 1), hence there can be at most two words in a single-deletion code with codewords of length 2.

If $n = 3$ for the 2-deletion case, there are only q strings in $\mathcal{A}_q^{3-2}$ and by the same reasoning as above there can be at most q codewords in a 2-deletion correcting code.

If $n = 4$ for the 3-deletion case, we again get at most q different codewords in a 3-deletion correcting code.

In general we get the rule: if $s = n - 1$, then $|\mathcal{C}^*_{q,s,n}| = q$.

[1] HP ZBook 15 G5 i7 (8th gen).

Table 4.1: *The fractional transversal numbers* $\tau^*(\mathcal{H}^D_{q,s,n})$ *for different values of* q *and* n *and* $s = 1$, $s = 2$ *and* $s = 3$ *respectively.*

(a) *single-deletion*

n	$q=2$	$q=3$	$q=4$	$q=5$	$q=6$	$q=7$
3	2	5	8	11	16	21
4	4	12	25	45	73	112
5	6	24	69	158	314	
6	10	62	231			
7	17	153				
8	30					
9	53					
10	96					
11	175					
12	321					

(b) *two deletions*

n	$q=2$	$q=3$	$q=4$	$q=5$	$q=6$	$q=7$
3	2	3	4	5	6	7
4	2	3	6	8	10	14
5	2	6	13	22	34	
6	4	13	33			
7	5	26				
8	7					
9	12					
10	20					

(c) *three deletions*

n	$q=2$	$q=3$	$q=4$	$q=5$	$q=6$	$q=7$
4	2	3	4	5	6	7
5	2	3	4	7	9	
6	2	5	9			
7	2	8				
8	4					
9	5					

4.2 Comparing the Bounds

Now that we have computed some upper bounds via the linear program of a transversal number, we can see how these compare to the analytical bounds in Chapter 2. For the single-deletion case, Table 4.2, We can think of the Varshamov-Tenengolts codes as a lower bound on the maximum cardinality of a single-deletion correcting code. This is because we know how many codewords are in a $VT_0(n)$ code and we know that this is a single-deletion code so we know there must be at least as many codewords in a largest single-deletion code.

In Table 4.2 we see that when n increases, all values of the different upper bounds also increase. This is due to the longer strings so there are more combinations of symbols which leads to more strings in $\mathcal{A}_q^n$ when n increases. We also see that for the LP column, the bound is tight for some values of n and always presents the strongest upper bound. The KK column, which is the bound in closed form expression found by Kulkarni and Kyiavash in [17] and equation (2.3) in this thesis, was derived from a linear program and has reasonably strong upper bounds, even as n increases. The two analytical bounds found by Levenshtein are clearly the weakest and they both grow much faster as n becomes larger. So even the strongest bounds in Table 3.1 are weaker than both bounds related to linear programs. There is a horizontal trend that suggests the following: the older the bound, the weaker the bound.

For the single-deletion case, we have chosen to only compare the upper bounds for binary codes. This motivation is twofold. Firstly, there are more bounds for the binary single-deletion case so we can compare the results of the linear program to more values. Secondly, there are many other comparisons in literature for q-ary alphabets when $s = 1$, like in [17]. In Table 4.3, we *will* look at larger alphabets.

Table 4.2: *Upper bounds for the maximum cardinality of a single-deletion correcting code. From left to right:* $VT_0(n)$*, the sizes of the largest known single-deletion correcting codes. These values can be interpreted as a lower bound on* $|\mathcal{C}^*_{2,1,n}|$*. LP shows the values of the fractional transversal number* $\tau^*(\mathcal{H}^D_{2,1,n})$*. KK shows the values of the closed form expression found by Kulkarni and Kiyavash, eq. (2.3). LEV-2 corresponds with the lowest values of each n for eq. (2.2), which can be found in Table 3.1 and LEV-1 corresponds with the values from eq. (2.1). All values have been rounded down.*

$q = 2$

n	$VT_0(n)$	LP	KK	LEV-2	LEV-1
3	2	2	3	4	2
4	4	4	4	6	4
5	6	6	7	10	8
6	10	10	12	18	16
7	16	17	21	34	32
8	30	30	36	58	64
9	52	53	63	103	128
10	94	96	113	190	256
11	172	175	204	363	512
12	316	321	372	646	1024

Also, to the best of the author's knowledge, there are no good codes correcting three deletions so there are no good comparisons to be made.

In Table 4.3, we show the results of the bounds obtained via the linear program and compare these to a bound found by Levenshtein. Also, we have given the code sizes of the best known codes capable of correcting 2 deletions. For the binary case, we have taken the code constructed by Butenko *et al.* in [6], where it can be seen in [16] that this code construction is indeed the largest for $q = 2$. For the 3-ary and 4-ary alphabets, we have taken a generalized Helberg 2-deletion correcting code construction from [18] to compare with the LP bounds. To the best of the author's knowledge, there are no good 5-ary codes correcting two deletions.

The first thing that catches our eye when looking at Table 4.3, is that we have only have 2 columns next to n and the column with known code sizes. This is because there are very few bounds for 2-deletion codes (or s-deletion codes for $s \geq 2$ for that matter) because this seems to be far less studied than the single-deletion case. For example, the first Levenshtein bound (2.1) only holds for a single deletion and the same is true for the bound found by Kulkarni and Kiyavash (2.3).

In each sub-table of Table 4.3, the values of the LP column were taken from Table 4.1b, where the fractional transversal number was computed for a 2-deletion correcting code and LEV-2 gives us the strongest bounds obtained from (2.2). We see the same vertical and horizontal trends as in table 4.2 even though there are less values here. As n increases, the values of the upper bounds increase as well. We also see that for each sub-table, the Levenshtein bound is far weaker in relation to the the fractional transversal number for the two deletion case than for the single deletion case.

In Table 4.3a we can compare the *LP* bound with the largest known sizes of binary codes capable of correcting 2 deletions and we see that for $n \leq 8$, the LP-bound is tight. In Tables 4.3b and 4.3c, we can't make any conclusions about the tightness of the LP bound by looking at the GH column. However for Table 4.3b, we can construct a repetition code to see that the LP bound is tight for $n = 3$ and $n = 4$. Namely a 3-ary repetition code of length $n = 3$, $\mathcal{R}^3_3 = \{000, 111, 222\}$ is a 2-deletion correcting code since one can check that the deletion sets of the codewords in $\mathcal{R}^3_3$ are pairwise disjoint. If $n = 4$, then the repetition code $\mathcal{R}^4_3 = \{0000, 1111, 2222\}$ is also a 2-deletion correcting code by the same reasoning.

For Table 4.3c, $\mathcal{R}^3_4$ acts as a 2-deletion correcting code with $q = 4$ and $n = 3$, so the *LP* bound is also tight.

In Table 4.3d, we can again use a repetition code to see that the bound is tight for $n = 3$. This time we take $\mathcal{R}^3_5 = \{000, 111, 222, 333, 444\}$.

Table 4.3: *Upper bounds for the maximum cardinality of a 2-deletion correcting code. LP shows the values of the fractional transversal number* $\tau^*(\mathcal{H}^D_{q,2,n})$. *In Table 4.3a, Butenko gives the sizes of the largest known binary 2-deletion code taken from [16] and LEV-2 shows the lowest values values of eq. (2.2). In Tables 4.3b and 4.3c, GH gives the sizes of a generalized 2-deletion Helberg code as presented in [18]. Again, all values have been rounded down.*

(a) $q = 2$.

n	Butenko	LP	LEV-2
3	2	2	4
4	2	2	6
5	2	2	10
6	4	4	18
7	5	5	34
8	7	7	66
9	11	12	106
10	16	20	156

(b) $q = 3$.

n	GH	LP	LEV-2
3	2	3	6
4	2	3	12
5	3	6	30
6	4	13	84
7	4	26	246

(c) $q = 4$.

n	GH	LP	LEV-2
3	2	4	8
4	2	6	20
5	3	13	68
6	4	33	260

(d) $q = 5$.

n	LP	LEV-2
3	5	10
4	8	30
5	22	130

Chapter 5

Discussion

The goal of this thesis was to study and evaluate upper bounds for deletion correcting codes. We have looked at how to obtain upper bounds for q-ary alphabets for 1- , 2- and 3-deletion correcting codes using linear programming and have compared the fractional transversal solutions for the single and 2 deletion case to other upper bounds that were derived analytically. The main conclusion is presented below. Furthermore, we will discuss the shortcomings of the methods used and we will give possible avenues for future work.

We conclude the following about the results that we have presented in this thesis. We have seen that the fractional transversal numbers produce the strongest bounds and that the so called 'LP-bounds' as defined in Chapter 2 are better bounds than their analytical counterparts. But in every case, the bounds computed by the fractional transversal outperformed all the other bounds and were also tight up to certain values of n. Also, a fractional transversal approach seems to be more amenable than other approaches since it has a compact representation.

Even though the upper bounds contrived from the fractional transversal number were superior to the other bounds, this method still has its limitations. Take for example the produced tables in the previous chapter. We were only able to show values of bounds for $n \leq 12$ for the binary single-deletion case and even less for other parameters. This is due to the exponentially large input size of the linear programs and therefore requires a lot of computing time. If one wanted to know that the fractional transversal number was for a single-deletion correcting code with codewords of length 17, one would need a supercomputer to even come close to getting a result.

Also, since computing time plays a big role in the fractional transversal method, it may be worth looking into better solvers like JuMP or having a look at Neos, instead of leaning towards personal preference.

For future work, a trivial direction is that one could try compute upper bounds for a higher number of deletions. Also, one could try to gain access to a supercomputer and therefore compute even more values with the code for the linear program in the Appendix.

Another direction is to compute lower bounds on $|\mathcal{C}^*_{q,s,n}|$ via linear programming, where one could translate the problem of finding a lower bound for $|\mathcal{C}^*_{q,s,n}|$ as finding the maximum independent set number $\alpha(G)$ and using theorems derived from graph theory to obtain a lower bound on $\alpha(G)$.

Also, maybe with the help of the previous direction, one could write out a linear program of a fractional matching and use the values of the fractional decision variables to construct a deletion correcting code for different parameters.

Appendix A

Matlab Code for Computing a Fractional Transversal Number

```
%parameters
n = 4;
q = 3;
s = 2;
m = n-s;


clear floor;

% Generate F_q
indices = cell(1, n);
[indices{:}] = ndgrid(0:(q-1));
% Create q-ary codebook of length n
codebook = zeros(q^n, n);
for i = 1:n
    codebook(:, i) = indices{i}(:);
end


T = table();


% Generate substrings
substrings = [];
for row = 1:size(codebook, 1)
    current_row = codebook(row, :);
    combinations = nchoosek(1:n, s); % Generate all combinations of s
       symbols to delete

    % Delete s symbols and generate substrings
    for combo = 1:size(combinations, 1)
        modified_row = current_row;
        modified_row(combinations(combo, :)) = []; % Delete symbols

        % Generate substrings
        row_substrings = cellstr(num2str(modified_row', '%d')); % Convert
           row to cell array of strings
        row_substrings = strcat(row_substrings{:}); % Concatenate all
           strings in the cell array
        T(row, combo) = {row_substrings};
    end
```

```
end
T;

% generate F^{q-s}
indices = cell(1,m);
[indices{:}] = ndgrid(0:(q-1));
codebook2 = zeros(q^m, m);
for i = 1:m
    codebook2(:,i) = indices{i}(:);
end

% create objective function for LP
obj = "";
for j = 1:q^m
    delj = codebook2(j, [1:m]);
    o = sprintf('%d', delj);
    obj(j) = o;
end
obj;
% create matrix M with the rows as the hyperedges
M = zeros(q^n, q^m);
for i = 1:q^n
    for j = 1:q^m
        for k = 1:size(combinations, 1)
            if T{i,k} == obj(j)
                M(i,j) = 1;
            end
        end
    end
end

M;
% M in the form of Ax \geq b
A = -M;
f = ones(1, q^m);
b = ones(1, q^n);
b = -b;
[x, fval] = linprog(f,A,b);
upperbound = floor(fval);
Table = table(q, n,s,  upperbound)
```

Bibliography

[1] K. Aardal, L. van Iersel, and R. Janssen. *Optimization lecture notes*. Delft University of Technology. 2022.

[2] K. A. S. Abdel-Ghaffar et al. "On Helberg's Generalization of the Levenshtein Code for Multiple Deletion/Insertion Error Correction". In: *IEEE Transactions on Information Theory* 58.3 (2012), pp. 1804–1808. DOI: 10.1109/TIT.2011.2174961.

[3] R. Beresnevicius. "When automation fails: remembering Qantas flight 72". In: *Aerotime* (2020).

[4] A Bishnoi. *Graph Theory*. Lecture notes from the course AM3550, TU Delft. 2023.

[5] A. Bretto. *An Introduction to Hypergraph Theory*. 1st ed. Springer Cham, 2013.

[6] S. Butenko et al. "Finding Maximum Independent Sets in Graphs Arising from Coding Theory". In: *Proceedings of the 2002 ACM Symposium on Applied Computing*. SAC '02. Madrid, Spain: Association for Computing Machinery, 2002, pp. 542–546. ISBN: 1581134452. DOI: 10.1145/508791.508897. URL: https://doi-org.tudelft.idm.oclc.org/10.1145/508791.508897.

[7] Y. M. Chee et al. "Codes correcting position errors in racetrack memories". In: *2017 IEEE Information Theory Workshop (ITW)*. 2017, pp. 161–165. DOI: 10.1109/ITW.2017.8278045.

[8] D. Cullina and N. Kiyavash. "An Improvement to Levenshtein's Upper Bound on the Cardinality of Deletion Correcting Codes". In: *IEEE Transactions on Information Theory* 60.7 (2014), pp. 3862–3870. DOI: 10.1109/TIT.2014.2317698.

[9] D. Cullina and N. Kiyavash. "Generalized Sphere-Packing Bounds on the Size of Codes for Combinatorial Channels". In: *IEEE Transactions on Information Theory* 62.8 (2016), pp. 4454–4465. DOI: 10.1109/TIT.2016.2565578.

[10] D. Cullina, A. A. Kulkarni, and N. Kiyavash. "A coloring approach to constructing deletion correcting codes from constant weight subgraphs". In: *2012 IEEE International Symposium on Information Theory Proceedings*. 2012, pp. 513–517. DOI: 10.1109/ISIT.2012.6284242.

[11] B. Daigle. *Data Centers Around the World: A Quick Look*. United States International Trade Commission. 2021.

[12] R. Heckel, G. Mikutis, and R. N. Grass. *A Characterization of the DNA Data Storage Channel*. 2018. arXiv: 1803.03322 [cs.ET].

[13] A.S.J. Helberg and H.C. Ferreira. "On multiple insertion/deletion correcting codes". In: *IEEE Transactions on Information Theory* 48.1 (2002), pp. 305–308. DOI: 10.1109/18.971760.

[14] D. S. Hirschberg. "Bounds on the Number of String Subsequences". In: *Combinatorial Pattern Matching*. Ed. by Maxime Crochemore and Mike Paterson. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 115–122. ISBN: 978-3-540-48452-3.

[15] R. M. Karp. "Reducibility Among Combinatorial Problems". In: *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Ed. by Michael Jünger et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 219–241. ISBN: 978-3-540-68279-0. DOI: 10.1007/978-3-540-68279-0_8. URL: https://doi.org/10.1007/978-3-540-68279-0_8.

[16] F. Khajouei, M. Zolghadr, and N. Kiyavash. "An algorithmic approach for finding deletion correcting codes". In: *2011 IEEE Information Theory Workshop*. 2011, pp. 25–29. DOI: 10.1109/ITW.2011.6089432.

[17] A. A. Kulkarni and N. Kiyavash. "Nonasymptomatic Upper Bounds for Deletion Correcting Codes". In: *IEEE transactions on information theory, vol. 59, no. 8* (2013), pp. 5115–5130.

[18] T. A. Le and H. D. Nguyen. "New Multiple Insertion/Deletion Correcting Codes for Non-Binary Alphabets". In: *IEEE Transactions on Information Theory* 62.5 (2016), pp. 2682–2693. DOI: 10.1109/TIT.2016.2541139.

[19] V.I. Levenshtein. "Binary Codes Capable of Correcting Deletions, Insertions and Reversals". In: *Soviet Physics Doklady* 10 (1966).

[20] V.I. Levenshtein. "Bounds for Deletion/Insertion Correcting Codes". In: *Proceedings International Symposium on Information Theory*. IEEE. 2002.

[21] V.I. Levenshtein. "On Perfect Codes in Deletion and Insertion Metric". In: *Discrete Mathematics and Applications* 2.3 (1992), pp. 241–258.

[22] V.I. Levensthein. "Binary codes capable of correcting spurious insertions and deletions of ones (in Russian)". In: *Russian Problemy Peredachi Informatsii* 1 (1965).

[23] Y. Liron and M. Langberg. "A characterization of the number of subsequences obtained via the deletion channel". In: *2012 IEEE International Symposium on Information Theory Proceedings*. 2012, pp. 503–507. DOI: 10.1109/ISIT.2012.6284240.

[24] S. Liu and C. Xing. "Bounds and Constructions for Insertion and Deletion Codes". In: *IEEE Transactions on Information Theory* 69.2 (2023), pp. 928–940. DOI: 10.1109/TIT.2022.3199503.

[25] M. Mahajan. "Matchings in Graphs". Lecture notes of the Institute of Mathematical Sciences, Chennai. 2010.

[26] H. Mercier, M. Khabbazian, and V. K. Bhargava. "On the Number of Subsequences When Deleting Symbols From a String". In: *IEEE Transactions on Information Theory* 54.7 (2008), pp. 3279–3285. DOI: 10.1109/TIT.2008.924730.

[27] A. Nappa, C. Hobbs, and A. Lanzi. "Deja-Vu: A Glimpse on Radioactive Soft-Error Consequences on Classical and Quantum Computations". In: *arXiv e-prints*, arXiv:2105.05103 (May 2021), arXiv:2105.05103. DOI: 10.48550/arXiv.2105.05103. arXiv: 2105.05103 [cs.CR].

[28] C. E. Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[29] N.J.A. Sloane. "On Single-Deletion-Correcting Codes". In: *Codes and designs, Ohio State University* (2002).

[30] S. Taluja, J. Bhupal, and S. R. Krishnan. "A Survey Paper on DNA-Based Data Storage". In: *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*. 2020, pp. 1–4. DOI: 10.1109/ic-ETITE47903.2020.62.

[31] R.R. Varshamov and G.M. Tenengolts. "Codes which correct single asymmetric errors (in Russian)". In: *Avtomatika i Telemekhanika* 26.2 (1965), pp. 288–292.