A CONSTRUCTIVE LOWER BOUND FOR CARDINALITY OF CODEBOOKS CAPABLE OF CORRECTING MULTIPLE DELETION AND INSERTIONS

BY

FARZANEH KHAJOUEI

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Industrial Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Adviser:

Assistant Professor Negar Kiyavash

ABSTRACT

The construction of the largest codebook capable of correcting multiple number of deletions and insertions is an open problem in coding theory. The efforts in the design of these codes mostly concentrate on finding the largest codebook size for a fixed number of deletions and a codeword length. In fact, most of these codebooks are designed for a specific number of deletions as few as one or two. We are interested in finding the largest codebook that can correct multiple deletion and insertion errors. Previous research focused on block codes in dealing with deletion and insertion errors.

The problem of constructing the largest codebook can be converted into an independent set problem in some specific graphs. The exact solution for the maximal independent set in these graphs is equivalent to finding the largest possible codebooks capable of correcting specific number of deletions and insertions. We propose a greedy algorithm which can find a maximal solution in polynomial time in the number of vertices of the graph. Results are presented for block codes of length n and the lower bounds are proved from analyzing the greedy algorithm on these graphs. A general construction for binary block codes, capable of correcting up to s number of deletion and insertion errors, is proposed. The construction is based on the concatenation of codes with shorter blocks. The algorithm will construct an s deletion and insertion correcting code based on a given $\lceil \frac{s}{2} \rceil$-deletion insertion correcting code. The size of the codebook grows exponentially and is comparable to asymptotic lower bound of Levenshtein. The greedy algorithm combined with the concatenation method can give codebooks of larger sizes.

To my parents, for their love and support.

ACKNOWLEDGMENTS

During my two years of study at UIUC, as a Master's student, I have had the privilege of interacting with so many wonderful colleagues and friends who have been greatly influential in my graduate life. These few sentences are an attempt to express my deepest gratitude to all those who made such an exciting experience possible. My foremost gratitude goes to my adviser Professor Negar Kiyavash for numerous reasons. Being not only a great adviser and an amazingly brilliant researcher but also a great friend. Negar is undoubtedly one of the most influential people in my life. Secondly, my deep appreciation and heartfelt gratitude goes to Dr. Ankur Kulkarni for his support, patience and guidance. He also reviewed my thesis report very carefully for even the delicate specifics for which I am very thankful to him. I would like to extend my warmest gratitude to Professor Ramavarapu Sreenivas for his guidance and support. My special thanks goes to my friends Mavis Rodrigues and Siva Kumar for their help and useful comments in writing this thesis. In addition to those mentioned above, I am grateful to so many amazing friends who made my study at Illinois an unforgettable stage of my life and full of memorable moments: Behnaz Arzani, Daniel Cullina, Quan Geng, Xun Gong, Parisa Hosseinzadeh, Sachin Kadloor, Sreeram Kannan, Christopher Quinn, Samaneh Mesbahi, Amin Sadeghi, Rezvan Shahoei, and Maryam Sharifzadeh.

TABLE OF CONTENTS

LIST OF TABLES . . . vii

LIST OF FIGURES . . . viii

CHAPTER 1 INTRODUCTION . . . 1

CHAPTER 2 OVERVIEW OF DELETION/INSERTION CORRECTING CODES . . . 3
- 2.1 Synchronization channel . . . 3
- 2.2 Background and related work . . . 5
- 2.3 Definitions . . . 7
- 2.4 Optimization problems in graphs . . . 10

CHAPTER 3 CONSTRUCTION OF CODES BASED ON GREEDY ALGORITHM FOR MAXIMAL INDEPENDENT SET PROBLEM 17
- 3.1 Introduction . . . 17
- 3.2 Independent sets in graphs . . . 18
- 3.3 Deletion correcting codes and independent sets . . . 19
- 3.4 Experiments . . . 21

CHAPTER 4 CONSTRUCTION OF CODES BASED ON CONCATENATION . . . 24
- 4.1 Introduction . . . 24
- 4.2 Concatenation and larger codebooks . . . 24
- 4.3 Partitioning and vertex coloring of the graph . . . 33
- 4.4 Bounds on the size of the code . . . 33
- 4.5 Conclusion . . . 39

CHAPTER 5 CONCLUSION . . . 40

APPENDIX A COMPLEXITY OF THE GREEDY ALGORITHM . 41
- A.1 Time complexity and performance of the greedy algorithm . . 41

APPENDIX B DEGREE OF ONE DELETION GRAPHS . . . 42
- B.1 Definitions and background . . . 42
- B.2 Degree for 1-deletion graph . . . 43

APPENDIX C NUMERICAL RESULTS 50
C.1 Some numerical results 50

REFERENCES 53

LIST OF TABLES

2.1 Construction of a two deletion correcting code of length 10. . . 8
2.2 Duality relations between optimization problems in graphs. . . 16

3.1 Maximum cardinalities for $s = 1, 3, 4$ deletions. 21
3.2 Comparison of size of double deletion correcting codes from various methods 23

LIST OF FIGURES

3.1 Example of deletion and insertion code generation graph for $n = 4$ and $s = 2$. Vertices are placed around the unit circle. 19
3.2 Comparison of size of the codebook for double deletion. 22

B.1 Sequence alignment 46

CHAPTER 1

INTRODUCTION

Deletion channel is a channel in which the output is a subsequence of the inputs while preserving the order of the transmitted symbols. Deletion errors occur when symbols are randomly dropped, and a subsequence of the transmitted symbols is received. Similarly, in an insertion channel some symbols may be inserted into the transmitted sequence at random positions. Generally speaking, deletion and insertion channels are examples of channels in which some errors can easily occur and it may result in the loss of synchronization. These channels arise in packet based communication of information and in biological applications [1, 2]. Despite recent advances in developing good error correction codes, the problem of finding good deletion and insertion correcting codes remains open and the results are few and far in between.

Deletions, insertions and reversal are some common errors that might happen to coded symbols. The effort on the design of deletion correcting codes has mostly concentrated on finding the largest codebook size for a fixed number of deletions and a codeword length [3–8]. In fact, most of these codebooks are designed for a specific number of deletions- as few as one or two. Our goal is to find maximum number of codewords such that the received sequence can be decoded uniquely.

It is known that the problem of finding deletion and insertion codes can be converted to an extremal graph problem, by finding the maximum independent set in an appropriately defined graph [9]. Let us assume that we are interested in binary codes of length n, capable of correcting s deletions. Let $G = (V, E)$ be an undirected graph where each vertex $v \in V$ is a binary sequence of length n. There are 2^n vertices in this graph. An edge is drawn between two vertices u and v if any of the subsequences produced after s or fewer deletions are the same. Clearly, finding an independent set in such a graph is equivalent to finding a codebook that can correct up to

s deletions and the maximum independent set corresponds to the codebook of the maximum size. However, finding the maximum independent set in a general graph is NP-hard [10]. Specially, for the coding problem of interest, the number of vertices grow exponentially with respect to the code length n and problem quickly becomes computationally intractable.

In this work, we will use a heuristic algorithm to find the largest known two deletion correcting codes for $n \leq 25$ [11]. Moreover, we provide a framework to design binary deletion correcting codes for a fixed length which combines a polynomial time heuristic algorithm to find the maximal independent set in a graph and the concatenation principle in coding theory. Once the graph G is constructed, our heuristic algorithm is quite efficient. However, as the codeword length n grows, the storage complexity of constructing the graph quickly becomes prohibitive. Hence, we use the heuristic to obtain codewords of smaller length and then concatenate them to the desired length n. Even through the work is focused on deletion errors, as proved by Levenshtein the codebook can also correct insertion errors [3].

The running time of the maximum independent set algorithm in general graphs is exponential; we should come up with either an exact algorithm for these specific graphs that can run in polynomial time or combine the solution with another method to overcome this restriction. In order to get larger cardinalities for codebooks, we will introduce a concatenation method to construct codes of any size based on available codes of smaller sizes. The cardinality of the codebook obtained by concatenation can be proved to grow exponentially in the code length and is comparable to the lower bound of Levenshtein [3]. Concatenation method combined with the greedy algorithm can give new results for the cardinality of codebooks.

This thesis is organized as follows: Chapter 2 reviews the literature and introduces the definitions and backgrounds needed for the rest of the thesis. After describing the background, in chapter 3 a polynomial time independent set algorithm is presented and the complexity of the algorithm is analyzed. In chapter 4 we describe how larger codebooks are constructed using concatenation. We present specific constructions for single and double deletion correcting codes. We will show that our combined approach outperforms all currently known constructions for any $s \geq 2$. Finally, we conclude our work in Chapter 5.

CHAPTER 2

OVERVIEW OF DELETION/INSERTION CORRECTING CODES

Deletion and insertion correcting codes are usually used to correct errors in channels with synchronization errors. In most communication networks, the objective is to find an encoding and decoding scheme that makes it possible to transfer information in rates near capacity. For channels with insertion and deletion errors, the capacity is unknown, although some upper and lower bounds on the capacity have been proved [8]. Previous research was focused on block codes in dealing with synchronization errors. We are interested in designing of codes that are capable of correcting multiple number of deletion and insertion errors.

In this chapter we will review important issues in construction of deletion and insertion correcting codes. To get an opinion about where these codes can be used first we introduce synchronization channel. We will give a brief review of previous efforts in literature in section 2.2. Basic definitions that are going to be used in the other chapters are given in section 2.3.

2.1 Synchronization channel

In most communication and storage channels, substitution errors are the most common type of errors. Substitution is referred to as the error in which a transmitted symbol is received as another symbol. A variety of coding techniques are developed to combat such errors.

However, channels may also suffer from synchronization errors. The concept of synchronization is defined as the following [12]: In a communication system if the events at the sender correspond to the events at the receiver, we call those systems synchronized. We can name three different kind of synchronization: carrier synchronization, bit synchronization and frame synchronization. Carrier synchronization deals with estimation of phase and frequency of the carrier wave. Variation in clock speed can cause bit synchronization error. The start and end of a frame, lost due to the insertions and deletions during the communication, causes frame synchronization errors.

Synchronization errors can be grouped into two types: deletion errors and insertion errors. Substitution is equivalent to one deletion and one insertion at the same place. Hence, it is a special case of deletion and insertion errors. Deletion errors occur when we are not receiving a transmitted symbol and insertion errors occur when we receive a spurious symbol that was not transmitted. Deletion and insertion errors can have a negative effect on the reliability of the communication channel even if powerful codes are used to correct substitution errors. Therefore, there is a compelling reason to consider codes that not only correct substitution errors, but can also recover from deletion and insertion errors.

In order to see how deletion and insertion errors can affect the reliability of communication, we can consider a case in a covert communication channel. In some applications, such as network flow watermarking, symbols are embedded into inter-packet delays. In this setup, packets are transported over a communication network via a set of links and nodes connecting the source to the destination. A failure in any part of the communication route may cause a packet to be lost, causing a deletion error. Repacketization, which is a common event in routers, will cause insertion errors. Using codes that only correct substitutions cannot guarantee the reliability of the communication.

Constructing good coding schemes and an efficient decoding algorithm is hard since we lack an understanding of the behavior of channels with synchronization errors. A wide range of techniques, ideas and tools are used to find codes that can correct synchronization errors. Some of these techniques were listed in [13]. The best known code construction, dealing with deletion and insertion errors, is the seminal work of Levenshtein on algebraic block codes. *Non-binary perfect codes* and *bursts correcting codes* are two other groups of codes designed to deal with these types of errors. *Synchronizable codes*, *marker codes*, *codes for weak synchronization errors*, *convolutional codes*, *spectral-null codes*, *expurgated codes* and *codes over random synchronization channels* are some other codes listed in [13] which can deal with synchronization errors.

2.2 Background and related work

To put our contribution into perspective, we will give a short review on previous related work. Levenshtein's seminal work introduced deletion and insertion correcting codes and derived upper and lower bounds on the size of s deletion and insertion correcting codes for any s [3]. Although these bounds on the size of the codebook are the best known general bound for any number of deletions or insertions, they are not constructive. Ullman studied the problem in a combinatorial context and derived both upper and lower bounds on the redundancy needed to correct various classes of synchronization errors [5].

Sloane presented a block synchronization correcting code which could correct a single deletion error per block. He used the Varshamov-Tenengolts code for this purpose [6]. Varshamov-Tenengolts code is a single asymmetric error correcting code and is referred as Leveneshtein's code [14]. The encoding and decoding algorithm for Levenshtein's code are very efficient, but the code cannot correct beyond a single deletion. The Varshamov-Tenengolts code is defined as the set of all n-tuple binary vectors satisfying the following equation:

$$VT_a(n) = \{x| \sum_{i=1}^{n} i.x_i \equiv a \bmod n+1, x = (x_1, \ldots, x_n) \in F_2^n\} \tag{2.1}$$

Helberg and Ferreira analyzed the weight spectra and the Hamming distance properties of single insertion and deletion correcting codes [15]. They proposed the first generalized number-theoretic code construction to correct multiple random insertion and deletion errors by using these relationships. They used the idea of Varshamov-Tenengolts code and generalized to correct any number of deletions. Their method can be used for constructing codebooks capable of correcting any number of deletions or insertions, but the size of the codebooks is far from optimum. Their codebook construction is as the following:

$$\sum_{i=1}^{n} v_i x_i \equiv a \bmod u \tag{2.2}$$

$$v_i = 1 + \sum_{j=1}^{s} v_{i-j} \tag{2.3}$$

$$u = 1 + \sum_{j=0}^{s-1} v_{n-j} \tag{2.4}$$

In the case of $s = 2$, we would have a sequence that recurse on only the last two previous steps. The coefficients equal to $v_i = 1 + v_{i-1} + v_{i-2}$ for each $1 \leq i \leq n$ and the mod is $u = 1 + v_n + v_{n-1}$. We can see that the recursive formula gives the Fibonacci numbers plus one. The first n terms of the sequence are weights and the last term is the mod.

Swart and Ferriera [7] used a run-length representation of sequences to determine sub and super sequences after two insertions or deletions. By searching through these sequences, they found a double insertion/deletion codebook of a larger size than what was previously known. However, their method is computationally expensive and cannot provide optimal solutions.

Butenko *et al.* [16] found the largest codebook by considering the equivalent maximum independent set problem. They suggested a heuristic to lower bound the size of the maximum independent set. Based on this heuristic, they proposed an exact algorithm for the maximal independent set problem in a general graph. Clearly such an exact algorithm finds the largest codebook size. However, because of prohibitive complexity, the algorithm cannot go beyond code length $n = 11$.

Constructing algebraic codes capable of correcting multiple deletions and insertions was not successful. In fact, the best known codes have all been found through search algorithms. In coding theory, there are good algebraic constructions for block codes capable of correcting specific number of errors using the Hamming metric, but there are few results for the Levenshtein metric; see [3, 6]. Butenko et al. found binary codes for correcting two deletions for codes of maximum length of 11.

2.3 Definitions

For two given sequences x and y of the same length, the *Hamming distance*, denoted by $d_H(x, y)$, is defined as the number of places where the corresponding symbols are different. For two sequences x and y, possibly of different lengths, the *Levenshtein distance*, $d_L(x, y)$ is defined as the minimum number of deletions and insertions needed to convert one sequence to the other. Note that substitution operation is not considered in the definition. If we include the substitution operation, we call the distance, the *edit distance* of two sequences. The deletion, insertion and substitution operations are usually referred to as edit operations. It is easy to show that the Hamming distance and the Levenshtein distance satisfy non-negativity, identity of indiscernibles, symmetry and triangular inequality. As a result they are metrics.

A binary block code of length n, denoted by $C \subseteq F_2^n$, is a set of n-tuples taken from $F_2 = \{0, 1\}$. In this thesis, we are using the Levenshtein metric in the costruction of our codes. In the following we will define some terminology and notations that we are going to use in the rest of the thesis.

Definition 1 Let x be a binary vector of length n. Define $D_s(x)$ as the set of all binary vectors of length $n - s$ obtained from x by deleting s bits in arbitrary positions. Similarly, $I_s(X)$ is defined as the set of all binary vectors of length $n + s$ obtained from x by inserting s bits in arbitrary positions.

Definition 2 A code $C_{s,n} \subseteq F_2^n$ is said to be s-deletion/insertion correcting code, if and only if for all c_i and c_j in $C_{s,n}$, $c_i \neq c_j$: $D_s(c_i) \cap D_s(c_j) = \emptyset$. For example $C_{1,3} = \{000, 101\}$ and $C_{2,6} = \{000000, 000111, 111000, 111111\}$.

Definition 3 Define $\mathcal{C}_{s,n}$ as a family of sets, each of which is an s-deletion and insertion correcting codes of the largest size. In other words, if $C \in \mathcal{C}_{s,n}$, then C is an s-deletion/insertion correcting code and there are no codewords in F_2^n that can be added to C to increase its cardinality.

Definition 4 Let $G(V, E)$ be an undirected graph where each vertex $v \in V$ is labeled with a binary sequence of length n, i.e. $V = F_2^n$. Define $L_{s,n} =$

$G(V,E)$, a graph in which an edge connects two vertices u and v if and only if $d_L(u,v) \leq 2s$.

Definition 5 Let $G(V,E)$ be an undirected graph with vertex set V and edge set E. Suppose S is a subset of V and let $G(S)$ denotes the subgraph induced by vertices in S. A set IS $\subseteq V$ is an independent set if the edge-set of $G(\text{IS})$ is the empty set. An independent set is maximal if it is not a subset of any larger independent set. An independent set with maximum cardinality is called maximum independent set.

Definition 6 The concatenation of an n_1-bit code C_1 of size $|C_1|$ and an n_2-bit code C_2 of size $|C_2|$, denoted by $C_1 \times C_2$, is an (n_1+n_2)-bit code of size $|C_1| \cdot |C_2|$, where each codeword in $C_1 \times C_2$ is obtained by the Cartesian product of codewords from C_1 to the codewords from C_2.

Example Suppose $C_1 \in \mathcal{C}_{2,3}$ and $C_1 \in \mathcal{C}_{2,7}$ with the codewords $C_1 = \{000, 111\}$ and $C_2 = \{0000000, 1111111, 0000111, 1111000, 0101010\}$. Table 2.1 shows how we can make codebook of length 10 by concatenating two double deletion correcting codes of length 3 and 7.

Codewords	0000000	1111111	0000111	1111000	0101010
000	0000000000	0001111111	0000000111	0001111000	0000101010
111	1110000000	1111111111	1110000111	1111111000	1110101010

Table 2.1: Construction of a two deletion correcting code of length 10.

Definition 7 Let $A, B \subseteq F_2^n$ be two sets, $d_L(A)$ is defined as the minimum Levenshtein distance between any two sequences in A and $d_L(A,B)$ is defined as the minimum Levenshtein distance between two sets A and B such that:

$$d_L(A,B) = \min_{x \in A, y \in B} d_L(x,y)$$

.

Definition 8 The longest common subsequence of two sequences x and y, denoted by $LCS(x,y)$, is defined as the common sub-sequence obtained from x and from y by minimum number of deletions.

The length of the longest common subsequence can be obtained from the Levenshtein distance between x and y, where we are allowing only deletion and insertion and not substitution.

Definition 9 A vertex coloring of $G = (V, E)$ is an assignment of k colors to vertices in G such that none of the adjacent vertices can have the same color. Vertices with the same color form an independent set. The minimum number of colors is called the chromatic number of G and is usually denoted by $\chi(G)$. The graph coloring problem minimizes the number of disjoint independent sets of G that form a partition of V.

There are some bounds on the chromatic number of a graph. For instance, an upper bound is given by Brooks [17]. Brooks' theorem states a relationship between the maximum degree of a graph and its chromatic number. Consider a graph which is not a complete graph or a cycle graph of odd length. Suppose Δ is the maximum degree of the graph, then the the vertices can be colored with maximum Δ colors where $\chi(G) \leq \Delta(G)$.

For a binary sequence X of length n, let $D_s(X)$ and $I_s(X)$ denote the set of all binary sub-sequences and super-sequences obtained from X by s number of deletions and insertions respectively. Let τ be the number of runs in the sequence X, where $\tau \leq n$. The cardinality of these sets can be obtained from the following ([3], [18]):

$$\binom{\tau - s + 1}{s} \leq |D_s(X)| \leq \binom{\tau + s - 1}{s} \tag{2.5}$$

$$|I_s(X)| = \sum_{i=0}^{s} \binom{n+s}{i} \tag{2.6}$$

Construction of the largest deletion and insertion correcting code relates to the problem of finding maximal independent sets in appropriately defined graphs. The maximum independent set problem and its related optimization problems are well studied in literature. In the next section, we will introduce these optimization problems formally and we will see how these problems are related.

2.4 Optimization problems in graphs

In this section, we survey different mathematical formulation for some optimization problems in graphs. We will give precise definition and mathematical formulation for these problems is provided: the maximum independent set problem, maximum clique, minimum vertex covering, maximum matching problem and different versions of coloring problem.

In algorithms, we usually study two cases for each problem, a decision version and an optimization version. The decision version usually determines whether there is a solution of given size or not. The optimization version provides a solution with the optimum size. In many cases, it is easier to solve the decision version of the problem. The following definitions hold through the report.

Let $G(V, E)$ be an undirected graph where $V = \{1, 2, \ldots, n\}$ denotes the set of vertices and E denotes the set of edges. Let $|V| = n$ and $|E| = m$, define A_G as the adjacency matrix of the graph G. The graph $\overline{G} = (V, \overline{E})$ is the complement graph of $G(V, E)$ where $\overline{E}$ is the complement of E. In all of the formulations x_i is a non-negative variable assigned to node $i \in V$. For $V' \subseteq V$, $G(V')$ is the subgraph induced by V' on G. The neighbor set of vertex i is denoted by $N(i)$ and the degree of that vertex is denoted by $d_i = |N(i)|$. A non-zero and non-negative weight vector $w \in R^n$ is considered as the weight vector for the vertex set where each w_i is associated with vertex i.

2.4.1 Maximum Independent Set Problem

In a graph G an independent set I is a subset of vertices, none of which are adjacent. Equivalently, I is an independent set if and only if the edge set of $G(I)$ is empty. An independent set is maximal if it is not a subset of any larger independent set. An independent set with maximum cardinality in a graph is called maximum independent set. The size of maximum independent set in a graph G is denoted by $\alpha(G)$ and is usually called the independence number, stability number or vertex packing number. There are many equivalent formulations for maximum independent set problem [19]. Some existing approaches will be reviewed in the following.

The maximum weight independent set problem can be formulated as an integer programming problem as the following:

$$\begin{aligned} \max \quad & \textstyle\sum_{i=1}^{n} w_i x_i \\ \text{subject to} \quad & x_i + x_j \leq 1 \quad \forall (i,j) \in E \\ & x_i \in \{0,1\} \quad i = 1, \ldots, n \end{aligned} \tag{2.7}$$

The problem can be converted to another formulation which is quadratically constrained [20].

$$\begin{aligned} \max \quad & \textstyle\sum_{i=1}^{n} w_i x_i \\ \text{subject to} \quad & x_i x_j = 0 \quad \forall (i,j) \in E \\ & x_i^2 - x_i = 0 \quad i = 1, \ldots, n \end{aligned} \tag{2.8}$$

Suppose $A = A_G - J$ where A_G is the adjacency matrix and J is n by n identity matrix. The following formula is a formulation in the global quadratic zero-one problem considered in [21].

$$\begin{aligned} \max f(x) = \quad & \max x' A x \\ \text{subject to} \quad & x_i \in \{0,1\} \quad i = 1, \ldots, n \end{aligned} \tag{2.9}$$

Abello et al. provide two polynomial formulations where $\alpha(G)$ can be characterized as an optimization problem [22]. It is proved that for problems which are linear with respect to each variables, the optimal solution always has a zero-one values for each x_i.

$$\begin{aligned} \alpha(G) = \max_{0 \leq x_i \leq 1, i=1,\ldots,n} & \sum_{i=1}^{n} (1 - x_i) \Pi_{(i,j)\in E} x_j \\ & x \in [0,1]^n \end{aligned} \tag{2.10}$$

A quadratic polynomial formulation was provided in [22] as the following:

$$\alpha(G) = \max_{0 \le x_i \le 1, i=1,\ldots,n} \sum_{i=1}^{n} x_i - \sum_{(i,j)\in E} x_i x_j$$
$$x \in [0,1]^n \tag{2.11}$$

2.4.2 Maximum Clique Problem

The maximum clique problem can be stated in two different ways [19].

Decision version: Given a graph G and an integer k, are there k vertices in the graph which are all adjacent to each other?

Optimization version: Find a clique with maximum cardinality. Consider $w(G)$ as the clique number, then the maximum clique problem asks for the following:

$$w(G) = \max\{|S| : S \text{ is a clique in } G\}$$

The following formula provides an integer programming formulation, in which the value of the optimal solution is equal to $w(G)$, the clique number.

$$\begin{array}{rll} \max & \sum_{i=1}^{n} w_i x_i & \\ \text{subject to} & \sum_{i\in S} x_i \le 1 & \forall S \in C = \{\text{Maximal Cliques of G}\} \\ & x_i \in \{0,1\} & i = 1,\ldots,n \end{array} \tag{2.12}$$

Maximum clique problem is equivalent of the maximum independent set problem in the complement graph $\overline{G}$. Therefore, any formulation for the independent set problem can be used for the maximum clique problem by replacing E with $\overline{E}$ [22]. The clique formulation in (2.12) is preferred to the formulation in (2.8) since its optimal solution has a smaller gap when we use the linear relaxation. We should point out that the efficiency of the second formulation is less since it has exponential number of constraints.

Motzkin and Straus [23] provided a continuous formulation for the quadratic form. The global optimal solution equals to $\frac{1}{2}(1 - \frac{1}{w(G)})$

$$\max f(x) = \max \tfrac{1}{2} x' A x$$

$$\text{subject to} \quad e'x = 0$$
$$x \geq 0 \tag{2.13}$$

Edge formulation:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} x_i \\ & x_i + x_j \leq 1, \forall (i,j) \in \overline{E} \\ & x_i \in \{0,1\}, i = 1, 2, \ldots, n \end{aligned} \tag{2.14}$$

2.4.3 Graph Coloring Problem

1. Vertex Coloring

   A vertex coloring of $G = (V, E)$ is an assignment of k colors to vertices in G such that none of the adjacent vertices can have the same color. Vertices with the same color form an independent set and we should have at least the size of maximum clique number of different colors.

   The minimum number of colors is called the chromatic number of G and usually denoted by $\chi(G)$. The graph coloring problem precisely minimizes the number of disjoint independent sets of G that form a partition of V . Apparently, minimum clique partition problem is a dual problem of graph coloring. Let $\bar{\chi}(G)$ denotes the minimum number for the clique partition of G, the following is immediate.

   $$\alpha(G) \leq \bar{\chi}(G) = \chi(\bar{G}).$$

   From the definition, the vertex coloring can be translated into the following constraints: for each vertex v_i and color k, we can consider a variable

   $$x_{ik} \in \{0,1\}$$

   where x_{ik} is one if we color v_i with color k.

Each vertex should be colored with only one color:

$$\sum_k x_{ik} = 1 \ \forall i$$

No adjacent vertices can have the same colors $\forall \ (i, j) \in E$, $\forall k$:

$$x_{ik} + x_{jk} \leq 1$$

Thus formulation has a variable for each independent set in the graph. Although this formulation has enormous number of variables, effective column generation techniques were developed for this problem.

The problem of determining whether K colors are enough to color the graph can be formulated as the following. Let x_{ik} be a binary variable for $i \in V, 1 \leq k \leq K$. The variable x_{ik} is equal to one if vertex i has color k and zero otherwise.

$$\text{Minimize } \sum_s x_s$$

$$x_{ik} + x_{jk} \leq 1 \forall (i, j) \in E, \forall k$$

$$\sum_k x_{ik} = 1 \forall i$$

$$x_{ik} \in \{0, 1\}.$$

2. Edge Coloring

   An edge coloring of G asks for the smallest set of colors needed to color the edges E, such that no two edges with the same color have common endpoint. For each edge $(i, j) \in E$ assign a color k, $x_{ijk} = 1$ if edge (i, j) is colored with color k.

$$x_{ijk} \in \{0, 1\}.$$

   Each edge can be colored with only one color.

$$\sum_k x_{ijk} = 1, \ \forall (i, j) \in E$$

Edges with the same endpoints cannot have the same color:

$$\sum_{j} x_{ijk} \leq 1, \forall i \in V,\ \forall k$$

$$\sum_{i} x_{ijk} \leq 1, \forall j \in V,\ \forall k$$

2.4.4 Minimum Vertex Cover Problem

This problem asks for the minimum number of vertices such that every edge has at least one endpoint in it.

A subset $S \subseteq V$ is a clique cover in G if and only if S is and independent set in $\bar{G}$ if and only if $V - S$ is a vertex cover in $\bar{G}$.

2.4.5 Minimum Dominating Set Problem

A subset of vertices that can cover all other vertices is called dominating set. In other words, S is a dominating set if each vertex in G is either in s or is connected to at least one vertex in G. The minimum dominating set problem asks for such set with minimum cardinality.

2.4.6 Clique Covering Problem

The clique cover problem asks to find the minimum number of cliques that include every vertex in the graph.

There are two different problems in this area: k-clique covering of the graph and clique covering number of the graph.

The problem of k-clique covering is defined on a graph G with a given clique size k. The problem is to use the least number of cliques such that each edge is contained in at least one clique and all vertices are also covered (isolated vertices). There might be some edges in cliques which are not in the graph. The problem of finding the clique covering number of graph usually arises in the case of perfect graphs. It asks for a partitioning of edges into minimum number of cliques. All edges of the graph is used in this problem.

It is easy to see that an instance problem of the k-colorability in graph G can be transformed into k-clique covering problem in its complement graph $\bar{G}$. A partition of $\bar{G}$ into k cliques then corresponds to finding a partition of the vertices of G into k independent sets; each of these sets can then be assigned one color to yield a k-coloring.

Table 2.2: Duality relations between optimization problems in graphs.

Covering-Packing Dualities	
Covering problems	Packing problems
Minimum Set Cover	Maximum Set Packing
Minimum Vertex Cover	Maximum Matching
Minimum Edge Cover	Maximum Independent Set

In the next chapter, we will discuss how the problem of finding largest deletion and insertion correcting codes relates to the problem of finding maximal independent sets in appropriately defined graphs.

CHAPTER 3

CONSTRUCTION OF CODES BASED ON GREEDY ALGORITHM FOR MAXIMAL INDEPENDENT SET PROBLEM

3.1 Introduction

It is known that the problem of finding deletion and insertion codes can be converted to a graph problem, namely finding the maximum independent set in an appropriately defined graph [9]. However, finding $C_{s,n}$ with maximum cardinality for large values of n through finding the maximum independent set is not practical as the later problem in a general graph is **NP**-hard [10]. Given the special structure of the graph, we show that a certain heuristic algorithm can be used to find large maximal independent sets.

Inspired by Sloane's approach of converting the problem of finding the largest codebook to that of finding the maximal independent set in an appropriately defined graph, we design a polynomial time heuristic algorithm for finding the maximal independent set of a given graph. This heuristic gives us a codebook, the cardinality of which is a lower bound on the size of the maximum feasible codebook. However, the polynomial complexity excludes the graph construction which becomes interactable as the size of the graph grows, i.e, the code length grows. To overcome this limitation, we optimally concatenate the codes obtained from our heuristic to construct codes of larger length.

In this chapter, we present the relation between independent set problem and code construction in Section 3.2. In Section 3.3 we will study the greedy minimum degree algorithm for the independent set problem in our specific graphs. Section 3.4 provides some simulations data from running the greedy algorithm on the graphs.

3.2 Independent sets in graphs

Lemma 3.1. *An independent set IS in $L_{s,n}$ is equivalent to an s-deletion and insertion correcting code.*

Proof. Suppose $u, v \in F_2^n$ and $u, v \in IS$, then $d_L(u, v) > 2s$. From the Definition 4, in an s-deletion/insertion correcting code C, for all $u, v \in C, u \neq v$: $D_s(u) \cap D_s(v) = \emptyset$. Therefore the statement in the Lemma 3.1 can be translated to the following statement: $d_L(u, v) > 2s \Longleftrightarrow D_s(u) \cap D_s(v) = \emptyset$.

First we prove $d_L(u, v) > 2s \Rightarrow D_s(u) \cap D_s(v) = \emptyset$. If $D_s(u) \cap D_s(v) \neq \emptyset, \exists z$ s.t $z \in D_s(u) \cap D_s(v)$. Since u, v has the same length, the number of deletions should be equal to number of insertions. Then there exists a sequence of s deletion and s insertion operations such that $u \to z \to v$ and $d_L(u, v) \leq 2s$.

Conversely, suppose $d_L(u, v) \leq 2s$, since the order of deletions and insertions is not important, assume we do the deletion first. Then there exist a z such that $u \to z \to v$ and $z \in D_{s'}(u) \cap D_{s'}(v)$, where $s' \leq s$. Therefore, $D_{s'}(u) \cap D_{s'}(v) \neq \emptyset \Rightarrow D_s(u) \cap D_s(v) \neq \emptyset$. □

Lemma 3.2. *A code that can correct s deletion errors can correct a total of s deletion and insertion errors, and vice versa.*

Proof. For a code C that can correct s-deletions $D_s(u) \cap D_s(v) = \emptyset$. On the other hand, for an s-deletions and insertions correcting code, any two codewords should be at a minimum distance af $2s + 1$, therefore for any u, v within the code $d_L(u, v) > 2s$. Moreover, from the proof of Lemma 3.1, $\forall u, v \in C : D_s(u) \cap D_s(v) = \emptyset \Longleftrightarrow d_L(u, v) > 2s$. Hence the deletion and insertion correcting code is also a deletion correcting code, and vice versa. □

Note that Levenshtein also proved the statement in Lemma 3.2 using a combinatorial argument [3]. As a result of Lemma 3.2 we construct our code only for deletion errors but it is a deletion and insertion correcting code.

Clearly, finding an independent set in a graph from Definition 4 is equivalent to finding a codebook that can correct up to s deletion. The maximum independent set in such graphs corresponds to the codebook of maximum size. In general graphs, maximum independent set and closely related problems including maximum clique, chromatic number, and minimum vertex cover are known to be **NP**-hard [10]. Consequently, all known exact algorithms rapidly become computationally infeasible as input word length

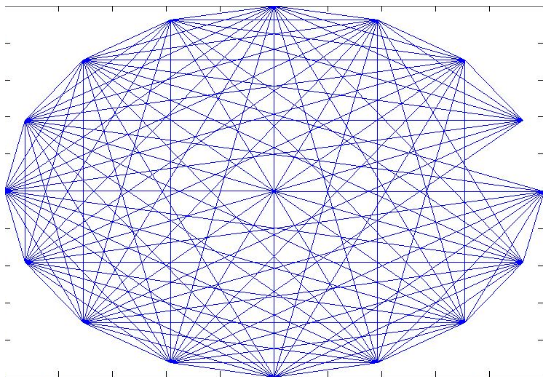

Figure 3.1: Example of deletion and insertion code generation graph for $n = 4$ and $s = 2$. Vertices are placed around the unit circle.

increases. Indeed, even approximating the size of the maximum independent set is difficult in general graphs. Specially, for $L_{s,n}$, the number of vertices grows exponentially in the code length and the problem quickly becomes computationally intractable.

This problems deals with a special class of graphs. Actually, studying this special class of graphs is an interesting problem by itself. The complexity of maximum independent set algorithm is polynomial in some families of graphs. If our graphs belongs to those families there might be an exact algorithm for the maximal independent set problem that runs in polynomial time. In this work, we used those results from graph theory that works in general graphs.

3.3 Deletion correcting codes and independent sets

As mentioned in the introduction and noted in [9], the problem of finding the maximal independent set in an appropriately defined graph is equivalent to our problem of finding s-deletion/insertion correcting codebooks of length n.

To construct the appropriate graph $G = (V, E)$, we consider each codeword $c_i \in C$ as a vertex $v_i \in V$. An edge $(v_i, v_j) \in E$ between pair of v_i and v_j is inserted if and only if $D_s(c_i) \cap D_s(c_j) \neq \emptyset$. Figure 3.1 depicts one such graph for $n = 4$ and $s = 2$.

Based on the definition of G, an independent set I in G is equivalent to a $C_{s,n}$ codebook, since it satisfies the conditions of Definition 2. A simple and yet powerful method for solving the maximum independent set problem is the greedy selection of vertices discussed in [11]. Even with the greedy algorithm that runs in polynomial time, the problem becomes computationally intractable since the size of the input grows exponentially. Thus we are looking for constructive way that can help us design these codes.

3.3.1 Min-Degree Selection Method

A simple and yet powerful method for finding a maximal independent set, is the greedy selection of vertices in V in the increasing order of vertex degree. This method runs through several iterations.

Algorithm 1. Heuristic maximal independent set algorithm

Input: Set $G(V, E)$, for each $v \in V$, $d(v)$: the vertex degree and $N(v)$: the set of neighbors.

```
1. Initialize $I$ with empty set;
2. while ($V \neq \emptyset$) {
3.       Select the vertex $v \in V$ with minimum $d(v)$;
4.       Add the selected vertex $v$ to $I$;
5.       Remove $v$ and all the vertices in $N(v)$ from $V$;
6.       Remove the edges incident to $v \cup N(v)$ from $E$;
   }
7. Output the maximal independent set $I$;
```

The algorithm starts with selection of minimum degree vertex in the graph, removing its neighbors and repeating the procedure for the remaining graph. In step 5, we remove the neighbors of v from V to prevent their selection in future iterations, because according to Definition 3, it is not possible to have neighbors in I. Note that removing the edges from E (step 6) will affect the vertex degree of the remaining vertices in V.

Since no two neighbors are placed in I, it is always an independent set, which corresponds to a deletion/insertion correcting codebook. Note that each vertex v is removed from V for further iterations, when we add that vertex or one of its neighbors to I. Therefore, when the algorithm finishes, I contains a maximal independent set but not necessarily the maximum one.

Table 3.1: Maximum cardinalities for $s = 1, 3, 4$ deletions.

n	$s = 1$		$s = 3$		$s = 4$	
	VT	Min-Deg	Helberg	Min-Deg	Helberg	Min-Deg
2	2	2	-	-	-	-
3	2	2	-	-	-	-
4	4	4	2	2	-	-
5	6	6	2	2	2	2
6	10	10	2	2	2	2
7	16	15	2	2	2	2
8	30	26	3	4	2	2
9	52	43	4	5	2	2
10	94	76	4	6	3	4
11	172	130	5	8	4	5
12	316	231	6	12	4	6
13	586	416	8	15	4	7
14	1096	759	8	20	5	9
15	2048	1367	9	28	6	12
16	3856	2520	11	40	8	14
17	7286	4641	15	58	8	19
18	13798	8550	16	83	8	25
19	26216	15843	18	123	9	34
20	49940	29436	22	186	11	45

3.4 Experiments

We briefly present experimental results for the construction of some deletion correcting codebooks. In Figure 3.2, we plot the codebook cardinalities with respect to the length of codewords for n up to 30. It can be seen that our construction improves upon Levenshtein lower bound for the 2-deletion correcting codes for this range of n.

In Table 3.1, we compared maximum cardinalities obtained from Helberg's method [15] and our method. Note that for $s = 1$, Helberg's code is the same as Levenshtein's code conjectured to be the optimal code for single deletion correction [6]. For single deletion, VT code outperforms our heuristic, but for larger number of deletions our codebook cardinalities are larger than any previously known results, namely Helberg's code. We have omited $s = 2$ as they apperead in Table 3.2 already.

In Table 3.2, our results together with the best known cardinalities for double deletion correcting codebooks are presented. The numbers under the

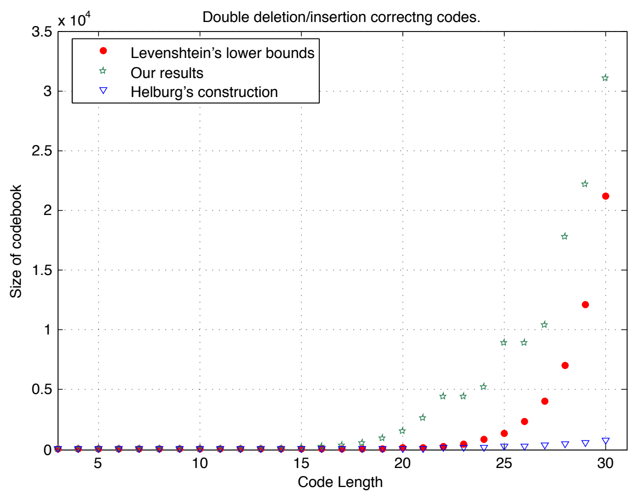

Figure 3.2: Comparison of size of the codebook for double deletion.

column Butenko are the optimum codebook sizes obtained from the exact solution of maximum independent set problem [16]. We can from the codebook sizes that our minimum degree selection algorithm recovered exceed the cardinality of codebooks obtained by Helberg and Swart's methods [7,15] presented in the same figure.

Even with our polynomial time algorithm, it is not practical to find codebook for very large n. The proposed concatenation method helps us to design codebooks with larger cardinality than the Helberg method which is the best known construction for multiple deletions. We show the results of the concatenation technique under the column "concat" we have improved upon the Levenshtein's lower bound on the cardinality of 2-deletion correcting codebooks for length up to 34. For very large n, we can construct a codebook by dividing the code into smaller blocks for which we have an optimal solution from the minimum degree selection algorithm.

Table 3.2: Comparison of size of double deletion correcting codes from various methods

n	Leven. LB	Helberg	Swart	Butenko	Min Deg	concat	Leven. UB
3	0	1		2	2		2
4	0	2	2	2	2		2
5	0	2	2	2	2		2
6	0	3	4	4	4		4
7	0	4	5	5	5		5
8	1	5	7	7	7		8
9	1	6	10	11	10		12
10	1	8	14	16	15		20
11	2	9	20	24	21		34
12	3	11	29		32		57
13	4	15			49		97
14	6	18			78		167
15	10	22			126		291
16	16	30			201		512
17	25	35			331		907
18	39	43			546		1618
19	64	57			911		2904
20	104	69			1539		5,242
21	172	88			2599		9,510
22	286	114			4450		17,331
23	479	142			7650		31,714
24	809	177			13,209		58,254
25	1,374	227			22,953		107,374
26	2,349	286				22,953	198,546
27	4,040	366				26,418	368,224
28	6,987	459				45,906	684,784
29	12,145	587				45,906	1,276,744
30	21,209	740				52,836	2,386,092
31	37,205	946				91,812	4,469,268
32	65,536	1,188				114,765	8,388,608
33	115,892	1,519				160,671	15,775,821
34	205,695	1,914				229,530	29,722,957
35	366,351					344,295	56,097,532
36	654,620					482,013	106,048,575
37	1,173,337					734,496	200,787,368
38	2,109,237					1,170,603	380,717,322
39	3,802,168					1,813,287	722,887,329
40	6,871,947					2,869,125	1,374,389,534

CHAPTER 4

CONSTRUCTION OF CODES BASED ON CONCATENATION

4.1 Introduction

As mentioned in the previous chapter, the time complexity of the Min-Degree selection algorithm becomes prohibitive as n grows. In this chapter, we are going to use a concatenation method to generate larger codebooks. We will prove a constructive lower bound on the size of the codebook and compare our bound with the Levenshtein's lower bound. The construction is based on concatenation of codes with shorter blocks. It will construct an s-deletion and insertion correcting code based on a given $\lceil \frac{s}{2} \rceil$-deletion insertion correcting code.

4.2 Concatenation and larger codebooks

In this section we will propose a construction method to form an s-deletion correcting code from the concatenation of two sets of smaller codes. First we will introduce a simple concatenation method to construct larger codes from codes with smaller sizes. Then based on the concatenation idea, we will introduce a method of construction that uses a code for smaller number of deletions to construct a larger code that can correct more number of deletions.

4.2.1 A simple concatenation method for construction of codes with larger lengths

In this section we are going to use concatenation to generate larger codebooks.

Lemma 4.1. *Let $x, y \in F_2^n$ and $w = LCS(x, y)$. Length of w is less than $n - s$ if and only if $d_L(x, y) > 2s$.*

Proof. If length of w is less than $n - s$, then $w \in D_{s'}(x) \cap D_{s'}(y)$, where $s' > s$. Since w is $LCS(x, y)$ for all $s < s'$: $D_s(x) \cap D_s(y) = \emptyset$, then from the proof of Lemma 3.1, $d_L(x, y) > 2s$. Conversely, if $d_L(x, y) > 2s$, then $D_s(x) \cap D_s(y) = \emptyset$. Then more than s bits should be deleted from both sequences to get a common subsequence. Therefore, length of w is less than s. □

Theorem 4.2. *Let $C_1 \in \mathcal{C}_{s,n_1}$, $C_2 \in \mathcal{C}_{s,n_2}$ be two sets of s-deletion correcting code and let $n = n_1 + n_2$. Code $C = C_1 \times C_2$ is an s-deletion correcting code.*

Proof. In order to prove this theorem, we present a decoding algorithm for $C_1 \times C_2$ as follows.

- Suppose that $(n_1 + n_2)$-bit codeword $c = c_1c_2$ has been transmitted and $(n_1 + n_2 - s')$-bit codeword r has been received, where $c_1 \in C_1$, $c_2 \in C_2$ and $s' \leq s$. Assume that in r, i deletions took place in c_1 and $s' - i$ deletions took place in c_2.

- To decode r, we need to decode remnant of c_1 (denoted by r_1) which is the leftmost $n_1 - i$ bits of r and remnant of c_2 by r_2 which is the rightmost $n_2 - (s' - i)$ bits of r using C_2 code.

- Since C_1 can correct up to s deletions, we can remove another $s' - i$ bits from the right side of r_1 (which will result in the leftmost $n_1 - s'$ bits of r). This sequence of bits is what is left from c_1 after s' deletions, so C_1 decoder can construct c_1 from it.

- Likewise, since C_2 decoder can correct up to s deletions, we can remove another i bits from the left side of r_2 (which will result in the rightmost $n_2 - s'$ bits of c'). This sequence of bits is what is left from c_2 after s' deletions, so C_2 decoder can construct c_2 from it.

□

In the following, we will give an alternative proof for the theorem and we will use it in the code construction.

Proof. Let's think of the code C as a two segmented code where the first segment is formed from codewords in C_1 and the second segment is formed from codewords in C_2. Suppose x and y are two elements of C. Let $x = x_1x_2$

and $y = y_1 y_2$, where $x_1, y_1 \in C_1$ and $x_2, y_2 \in C_2$. Suppose $z = LCS(x, y)$. We will show that the length of z should be less that $n - s$. If so, as a result of Lemma 4.1, C is an s-deletion/insertion correcting code.

Suppose we have a common subsequence of x and y that has a length of $n - s$. Assume $s = s_1 + s_2 = s'_1 + s'_2$ where s_1 deletions occurred in x_1, s_2 in x_2, s'_1 in y_1 and s'_2 in y_2. Note that s_1, s_2, s'_1 and s'_2 are less than s. Let $n'_1 = n_1 - s'$ where $s' = \max\{s_1, s'_1\} \leq s$. Consider the first n'_1 bits of z as the substring $z[1 \ldots n'_1]$ of z. Then $z[1 \ldots n'_1]$ is a substring of a member of $D_{s'}(x_1)$ and $z[1 \ldots n'_1] \in D_{s'}(y_1)$ therefore $z[1 \ldots n'_1] \in D_{s'}(x_1) \cap D_{s'}(y_1)$. Thus for $s' \leq s$, $D_{s'}(x_1) \cap D_{s'}(y_1) \neq \emptyset$. This is a contradiction because $x_1, y_1 \in C_1$ and C_1 is an s-deletion/insertion correcting code. We could have used the same argument for the second segment of the sequence. As a result, for two sequences x and y in C, $LCS(x, y)$ should have a length less than $n - s$.

□

The following is immediate.

Corollary 4.3. *For $C_1 \in \mathcal{C}_{s,n_1}$, $C_2 \in \mathcal{C}_{s,n_1}$, ..., $C_m \in \mathcal{C}_{s,n_1}$, $C_1 \times C_2 \times \cdots \times C_m$ is a $C_{s,\sum_{i=1}^{m} n_i}$ code.*

Proof. We prove this corollary using induction. The base of the induction for $m = 2$ has been proved in the theorem above. For $m > 2$, we can divide the concatenations into two parts. One containing $C_1 \times \cdots \times C_{m-1}$ and the other containing C_m. Using the assumption of induction, we know that $C_1 \times \cdots \times C_{m-1}$ is a $C_{s,\sum_{i=1}^{m-1} n_i}$. By applying the Theorem 4.2, we will have $C_{s,\sum_{i=1}^{m-1} n_i} \times C_m = C_{s,\sum_{i=1}^{m} n_i}$. □

Using concatenation, we can easily generate deletion/insertion correcting codebooks for higher dimensions. Note that decoding cost of these new codebooks is equal to sum of decoding costs of the comprising parts.

4.2.2 A Special Case of Concatenation

Let us explain a simple special case. Consider C_1 to be double-deletion-correcting codebook of length 3 and C_2 to be a larger double deletion correcting codebook. The codewords in C_1 are (000) and (111). Starting from an initial codebook of length n, we can make a new codebook of length $n+3$

by concatenating three bits 000 or 111 with each elements of C_2. This process will take an initial codebook of length n of size m and expand it to a codebook of length $n+3$ of size $2m$.

From the above discussion, we can see that it is important to start concatenation of codes from larger blocks. Hence the superiority of our heuristic algorithm is to create overall larger codebook starting from a large initial set found from the maximal independent set algorithm.

The concatenation method proposed in this section relies on the availability of s-deletion correcting code of shorter length. In the next section we will propose a new concatenation method that uses an $\frac{s}{2}$-deletion correcting code and gives an s-deletion correcting code.

4.2.3 Concatenation of codes using a buffer sequence

Definition 10 Suppose $A \subseteq F_2^n$ is a set of binary sequences partitioned into p disjoint sets, i.e $A = \bigcup_{i=1}^{p} A_i$ and $\forall i \neq j, A_i \cap A_j = \emptyset$. Let $B \subseteq F_2^m$ is another set of binary sequences that has a cardinality of p, $|B| = p$. Consider an ordered set $\beta = (\beta_1, \ldots, \beta_p)$ which is a permutation of elements in the set B, that each $\beta_i \in B$. The extension of set A with the set β, denoted by $A_e(\beta)$, is defined as the following: $A_e(\beta) = \bigcup_{i=1}^{p} A_i(\beta_i)$ where $A_i(\beta_i)$ is the Cartesian product of sequences in A_i with the sequence b_i.

$$A_i(\beta_i) = \{X = x_1x_2 | x_1 \in A_i \text{ and } x_2 = \beta_i\} \text{ for } \beta_i \in B \tag{4.1}$$

Note that $|A_i(\beta_i)| = |A_i|$ and $|A| = \bigcup_{i=1}^{p} |A_i| = \bigcup_{i=1}^{p} |A_i||\beta_i| = |A_e(\beta)|$. For example for $A = \{000, 101\} \cup \{100, 011\} \cup \{010, 111\} \cup \{001, 110\}$ and $B = \{00, 01, 10, 11\}$, the extended set A with the set $\beta = (00, 01, 10, 11)$ is demonstrated in the following.

$$\begin{aligned} A_1(00) &= \{00000, 10100\} \\ A_2(01) &= \{10001, 01101\} \\ A_3(10) &= \{01010, 11110\} \\ A_4(11) &= \{00111, 11011\} \end{aligned} \tag{4.2}$$

Definition 11 An alignment is a sequence of deletion operations transforming sequences x and y into a common subsequence of these two sequences. The minimum number of deletion operations is used when we are aligning x and y to $LCS(x, y)$ sub-sequence. We will show that the alignment is not possible for less than or equal to s number of deletions.

Theorem 4.4. *Consider the sets $A_1, A_2, \ldots, A_p \subset F_2^{n_1}$ and $B_1, B_2, \ldots, B_q \subset F_2^{n_2}$ be such that*

$$\forall x_1 \in A_i, y_1 \in A_j : \begin{cases} d_L(x_1, y_1) > 2s & \text{if } i = j \\ d_L(x_1, y_1) > s & \text{if } i \neq j \end{cases}$$

$$\forall x_2 \in B_i, y_2 \in B_j : \begin{cases} d_L(x_2, y_2) > 2s & \text{if } i = j \\ d_L(x_2, y_2) > s & \text{if } i \neq j \end{cases}$$

The sets are such that $|A_1| \geq |A_2| \geq \cdots \geq |A_p|$ and $|B_1| \geq \cdots \geq |B_q|$. Let $m = \min\{p, q\}$ and $\beta \subseteq C_{s,n_3}$, where n_3 is chosen such that $|\beta| \geq m$. A code C constructed such that $C = \bigcup_{i=1}^{m} C_i = \bigcup_{i=1}^{m} A_i(\beta_i) \times B_i$ can correct up to s number of deletions, where $n = n_1 + n_2 + n_3$.

Proof. Code C can be seen as a two segmented code joined with a buffer sequence. Let's call the first segment A and the second segment B. We are concatenating each of A_i sets with a fixed sequence β_i and then concatenating it with the second code B_i. The first segment of the code is that part the originated from A set and the B segment is the part from B set and the buffer sequence is from the β set. Within segment A, $d_L(A) > s$, hence segment A is an $\lfloor \frac{s}{2} \rfloor$- deletion correcting codes. This means that for any $x_1, y_1 \in A$ and $s' \leq \lfloor \frac{s}{2} \rfloor$, $D_{s'}(x_1) \cap D_{s'}(y_1) = \emptyset$. So, none of the subsequences of length $n_1 - \lfloor \frac{s}{2} \rfloor$ are equal. This property also holds for segment B.

Let $x = x_1\beta_i x_2$ and $y = y_1\beta_j y_2$, where $x_1 \in A_i, y_1 \in B_i$ and $x_2 \in A_j, y_2 \in B_j$ and $\beta_i, \beta_j \in \beta$. Two cases will arise:

Case 1: $i = j$

In this case, $x_1, y_1 \in A_i$ and $x_2, y_2 \in B_i$ where $d_L(A_i) > 2s$ and $d_L(B_i) > 2s$. Hence, A_i is a C_{s,n_1} code and B_i is also a C_{s,n_2} code. C_i is formed from concatenation of a C_{s,n_1} code and a C_{s,n_2} code and a buffer sequence β_i. As a result of Lemma 4.8, it is a $C_{s,n}$ code.

Case 2: $i \neq j$

In this case, $x_1 \neq y_1$ and $x_2 \neq y_2$. Let's assume we sent x and y, and s

deletions occurred. We will show that the subsequences cannot be similar. The number of deletions in each segment can take values from 0 to s. In either of segments $d_L(x_k, y_k) > s$ for $k = 1, 2$, then within each segments $\lfloor \frac{s}{2} \rfloor$ deletions can be corrected.

We will study different cases where deletions can happen and then we will use Lemma 4.1 to show that C is an s-deletion/insertion correcting code. We will show that there is no way of getting a common subsequence of length at least $n - s$ from codewords in code C after deletions. Let $n - s$ be the length of the received code.

Consider sequence x and assume s_A, s_B and s_b be the number of deletions in segment A, B and the buffer sequence β respectively, where $s = s_A + s_B + s_b$. Also, we have $s = s'_A + s'_B + s'_b$ for sequence y .

1. $s_A < \lfloor \frac{s}{2} \rfloor$
   (a) $s'_A < \lfloor \frac{s}{2} \rfloor$. Segment A cannot be aligned.
   Let $s' = \max\{s_A, s'_A\}$. Then $D_{s'}(x_1) \cap D_{s'}(y_1) = \emptyset$ since $s' \leq \lfloor \frac{s}{2} \rfloor$. Therefore the sequences x and y do not have a common subsequence of length $n - s$.
   (b) $s'_A \geq \lfloor \frac{s}{2} \rfloor$ ($\Rightarrow s'_B < \lfloor \frac{s}{2} \rfloor$)
      i. $s_B < \lfloor \frac{s}{2} \rfloor$. Segment B cannot be aligned.
      This case is also similar to the previous case where in part B of the sequences x, y after $s' = \max\{s_B, s'_B\} < \lfloor \frac{s}{2} \rfloor$ we always get different subsequences. Therefore, the length of $LCS(x, y) < n - s$.
      ii. $s_B \geq \lfloor \frac{s}{2} \rfloor$
      In this case $s_b, s'_b < \lfloor \frac{s}{2} \rfloor$.The buffer segment is an s-deletion correcting code. After each deletion in segment A the buffer segment will shift one position to the left. Since it can survive s deletions, the shifted version of the buffer segment cannot overlap to create the common subsequence from x and y. Note that we are shifting the buffer segment at most s positions down. Therefore within this segment, we cannot create a common segment from x and y.

The following cases can also be studied the same way we study the previous cases.

2. $s_A \geq \lfloor \frac{s}{2} \rfloor$ ($\Rightarrow s_B < \lfloor \frac{s}{2} \rfloor$)

   (a) $s'_A < \lfloor \frac{s}{2} \rfloor$

      i. $s'_B < \lfloor \frac{s}{2} \rfloor$. Segment B cannot be aligned.

      ii. $s'_B \geq \lfloor \frac{s}{2} \rfloor$. The buffer segment cannot be aligned.

   (b) $s'_A \geq \lfloor \frac{s}{2} \rfloor$ ($\Rightarrow s'_B < \lfloor \frac{s}{2} \rfloor$). Segment B cannot be aligned.

□

Let $A = \bigcup_{i=1}^{p} A_i$. The set A is an independent set in $L_{\lfloor \frac{s}{2} \rfloor, n}$ and each A_i sets are also an independent set in $L_{s,n}$. Let $b \subseteq C_{s,m}$, for m such that $|C_{s,m}| \geq p$, and let b_i be elements of b. We will prove that $m \leq O(\log n)$.

Form a code C of length $n' = 2n + m$ from the concatenation of A_i, b_i and A_i as the following:

Lemma 4.5. *Supposed a code C formed as the following:*

$$C = \bigcup_{i=1}^{p} A_i(b_i) \times A_i.$$

Then the cardinality of the code can be bounded by $\frac{|A|^2}{p}$ from below.

Proof. First, we will obtain an inequality from Cauchy-Schwarz inequality. Then a second inequality will be obtained from the union bound as follows.

$$|C| = \sum_{i=1}^{p} |A_i|^2 \geq \frac{\left(\sum_{i=1}^{p} |A_i|\right)^2}{p} \geq \frac{1}{p} |\bigcup_{i=1}^{p} A_i|^2 = \frac{|A|^2}{p} \tag{4.3}$$

To get the largest cardinality for code C, we should use the largest possible set as A, then partition it into smallest possible number of sets which is p. In the next section, we will see how these optimal values can be calculated. □

The concatenation with buffer sequence can be used more efficiently. Since we only used the distance characteristic of partitions and we didn't used the indices, there is a better way of concatenating the segments in order to get codes of larger sizes. In the next section, we will introduce this optimal concatenation.

4.2.4 Concatenation

Lemma 4.6. *Consider $x_1, x_2 \in F_2^{n_1}$ and $y_1, y_2 \in F_2^{n_2}$. Form two sequences $x, y \in F_2^{n_1+n_2}, x \neq y$ such that $x = x_1y_1$ and $y = y_1y_2$. If $d_L(x_1, x_2) > s$ or $d_L(y_1, y_2) > s$ then $d_L(x, y) > s$.*

Proof. The proof is trivial in case of $x_1 = x_2$ or $y_1 = y_2$. Assume $x_1 \neq x_2$ and consider $z = LCS(x, y)$. The length of z is less than $n - s$ since more than s-deletions are needed to align x_1 and x_2. □

Corollary 4.7. *If $d_L(y, y') > s$ then $d_L(xyz, x'y'z') > s$ where $x, x' \in F_2^{n_1}$, $y, y' \in F_2^{n_2}$ and $z, z' \in F_2^{n_3}$.*

Proof. If $d_L(y, y') > s$, then from Lemma 4.6 $d_L(yz, y'z') > s$. For two sequences yz and $y'z'$, by applying Lemma 4.6 again $d_L(xyz, x'y'z') > s$. □

Lemma 4.8. *Let $C_1 \in \mathcal{C}_{s,n_1}$, $C_2 \in \mathcal{C}_{s,n_2}$ be two sets of s-deletion correcting code. Let $\beta \in F_2^{n_3}$ be a fixed binary sequence and let $n = n_1 + n_2 + n_3$. Code $C = C_1 \times \beta \times C_2$ is an s-deletion correcting code.*

Proof. We will show that if s deletions occurred in a codeword from the C code then each of the sub-sequences of length $n - s$ can be uniquely assigned to a sequence of length n. Assume that the subsequence $z \in F_2^{n-s}$ is given. We know for sure that the first $n_1 - s$ bits are from C_1 code and the last $n_2 - s$ bits are from C_2 code. The number of deletions in the segments from C_1 and from C_2 are at most s. Therefore, each of the codes C_1 and C_2 can correct the deleted bits from them. The remaining bits were deleted from the sequence β, hence we can find the deleted bits by comparing the middle segment with β segment. As a result, there is a way to assign the subsequence of the length $n - s$ uniquely to a sequence and it is an s-deletion and insertion correcting code. □

Corollary 4.9. *Consider the sets $A = \bigcup_{i=1}^{p} A_i \subseteq F_2^n$ where each of A_i sets are disjoint and they have the following distance propoerty:*

$$\begin{aligned} d_L(A_i) &> 2s & 1 &\leq i \leq p \\ d_L(A_i, A_j) &> s & 1 &\leq i, j \leq p, i \neq j. \end{aligned}$$

Let $\beta \subseteq C_{s,m}$, where m is chosen such that $|\beta| \geq p^2$. A code C constructed such that $C = \bigcup_{i,j} C_{ij} = \bigcup_{i,j} A_i \times \beta_{ij} \times A_j$ can correct up to s number of deletions, i.e $C \subseteq C_{s,n'}$,where $n' = 2n + m$.

Proof. Let $x = x_1\beta_{ij}x_2$ and $y = y_1\beta_{kl}y_2$, where $x_1 \in A_i, x_2 \in A_j$ and $y_1 \in A_k, y_2 \in A_l$ and $\beta_{ij}, \beta_{kl} \in \beta$. Two cases will arise:

Case 1: $\beta_{ij} = \beta_{kl}$

In this case, $x_1, y_1 \in A_i$ and $x_2, y_2 \in A_j$ where $d_L(A_i) > 2s$ and $d_L(A_j) > 2s$. Hence, each of the A_i and A_j is a $C_{s,n}$ code. C_{ij} is formed from concatenation of a $C_{s,n}$ code, a buffer sequence β_{ij} and another $C_{s,n}$ code. As a result of Lemma 4.8, it is a $C_{s,n}$ code.

Case 2: $\beta_{ij} \neq \beta_{kl}$ The set β is an s deletion/insertion correcting code. Therefore, if $\beta_{ij} \neq \beta_{kl}$, $d_L(\beta_{ij}, \beta_{kl}) > 2s$. From Corollary 4.7, for any x and y constructed β_{ij}, β_{kl} as the middle segment, $d_L(x, y) > 2s$.

□

Algorithm 2 will select such disjoint sets. It starts with the graph $L_{s,n}$, and finds an independent set in this graph and assign it to the first A_i set. Then it forms a new graph by removing all the nodes in the independent set and their neighbors within the Levenshtein distance of s of the selected nodes. It repeats the same procedure for the new graph.

Algorithm 2. Selection of s-distance disjoint sets

Input: Graph $G(V, E)$ and MIS(G) as a subroutine for maximal independent set problem:

```
1. Initialize $G_0 = G$;
2. Output $A_0 = \text{MIS}(G_0)$;
3. $i = 0$;
4. while ($|G_i| \neq 0$) {
5. $G_{i+1} = G_i / \{A_i \cup \{\bigcup_{i=1}^{s} N_i(A_i)\}\}$;
6. $A_{i+1} = \text{MIS}(G_{i+1})$;
7. $i = i + 1$;
8. Output $A_i$; }
```

The partitioning of A and B sets is important in the proposed construction. In the next section, we will see how the disjoint sets A_i can be generated. We will relate this problem to the graph coloring problem.

4.3 Partitioning and vertex coloring of the graph

Lemma 4.10. *Let A be an independent set in $L_{\lceil \frac{s}{2} \rceil, n}$ graph. The minimum coloring of the $L_{s,n}(A)$ graph is a partitioning of A set into s-distance disjoint sets with minimum number of sets.*

Proof. From equation (4.3), and the fact that the set A is an independent set in $L_{\lceil \frac{s}{2} \rceil, n}$, the set A should be chosen such that it is the maximum independent set in $L_{\lceil \frac{s}{2} \rceil, n}$. Furthermore, the number of parts we get in the partitioning of A also should be the smallest possibe number.

Consider the graphs $L_{\lceil \frac{s}{2} \rceil, n}$, $L_{s,n}$ and the set A as the set of their vertices. Since $L_{s,n}$ has the same vertex set and also contains all edges of $L_{\lceil \frac{s}{2} \rceil, n}$ the graph $L_{\lceil \frac{s}{2} \rceil, n}$ is in $L_{s,n}$. For nodes $u, v \in A$, $d_L(u, v) > s$. Consider the graph $L_{s,n}(A)$ which is the subgraph of $L_{s,n}$ induces by the set A. Choose the A set such that it is an independent set in $L_{\lceil \frac{s}{2} \rceil, n}$. A coloring for the graph $L_{s,n}(A)$ will give the required partitioning of A. In this partitioning, each of the A_i sets is a color class in $L_{s,n}(A)$. Therefore they are independent sets in $L_{s,n}$ and can satisfy the distance properties. □

In the next section, we will calculate bounds on the size of the code generated from the concatenation and we will compare the bounds by the asymptotic bounds of Levenshtein [3].

4.4 Bounds on the size of the code

Levenshtein [3] provide asymptotic upper and lower bounds on the size of codebook with s-deletions/insertions. For fixed s, as $n \to \infty$,

$$\frac{2^s (s!)^2 2^n}{n^{2s}} \leq |C_{s,n}| \leq \frac{s! 2^n}{n^s}$$

Example For $s = 2$, this formula will reduce to

$$\frac{2^{n+4}}{n^4} \leq |C| \leq \frac{2^{n+1}}{n^2}.$$

Leveneshtein also proved that any codebook that can correct s-deletions (or insertion), can correct s number of deletions and insertions [3]. The concatenation method discussed in the previous section can be used to construct multiple deletion/insertion correcting codes in a recursive way. Given an $\lceil \frac{s}{2} \rceil$-deletion/insertion correcting code of length n, one can construct an s-deletion/insertion code of length $2n + \alpha \log n$.

Suppose $A_1, A_2, \ldots, A_p$ are p disjoint sets of sequences in F_2^n. The A_i sets were selected such that

$$\forall x, y \in A_i : \qquad d_L(x, y) > 2s \tag{4.4}$$

$$\forall x \in A_i, y \in A_j, i \neq j : \qquad d_L(x, y) > s \tag{4.5}$$

Lemma 4.11. *Let $A = \bigcup_{i=1}^{p} A_i$, the set A is an independent set in $L_{\lceil \frac{s}{2} \rceil, n}$ and each A_i sets are also an independent set in $L_{s,n}$. Let $\beta \subseteq C_{s,m}$ for m such that $|C_{s,m}| \geq p$ and let β_{ij} be elements of β. For a code C of length $n' = 2n + m$ formed from the concatenation of A_i, β_{ij} and A_j, the cardinality equals to the following: $|C| = |A|^2$.*

Proof.

$$\begin{aligned} |C| &= \sum_{i,j} |A_i| \cdot |\beta_{ij}| \cdot |A_j| = \sum_{i,j} |A_i||A_j| \\ &= \sum_{i=1}^{p} |A_i| \sum_{j=1}^{p} |A_j| = \left(\sum_{i=1}^{p} |A_i| \right)^2 = |A|^2 \end{aligned}$$

□

The A set is an independent set in $L_{\lceil \frac{s}{2} \rceil, n}$ graph and A_1 is an independent set in $L_{s,n}$. To get the largest cardinality for code C, we should use the largest possible set as A, then partition it into smallest possible number of sets which is p. From Lemma 4.10, the optimum partitioning of the A set into s-distance disjoint sets is from the coloring of induced subgraph of $L_{s,n}(A)$. Thus the minimum number of partitions equal to chromatic number of the graph, so $p = \chi(L_{s,n})$.

4.4.1 A lower bound on the independence number of $L_{s,n}$

Lemma 4.12. *For fixed s and asymptotic n, the chromatic number of $L_{s,n}$ graph can be bounded by the following*

$$\chi(L_{s,n}) \leq \frac{e^2}{2\pi}\frac{sn^{2s}}{(s!)^2e^{2s}} \tag{4.6}$$

Proof. Using the Brooks theorem, $\chi(G) \leq \Delta(G)$. An upper bound on the degree of graph can be derived easily by multiplying the number of deletions by the number of insertions. For all integer n, the Stirling formula states that:

$$\sqrt{2\pi}n^{n+\frac{1}{2}}e^{-n} \leq n! \leq en^{n+\frac{1}{2}}e^{-n} \tag{4.7}$$

Then by using Stirling approximation, we can get the following:

$$\begin{aligned}
\Delta(L_{s,n}) &\leq |D_s(X)| \cdot |I_s(X)| \\
&\leq \binom{\tau+s-1}{s} \cdot \sum_{i=0}^{s}\binom{n+s}{i} \\
&\leq s\binom{n+s-1}{s}\binom{n+s}{s} \\
&= \frac{sn}{(n+s)}\binom{n+s}{s}^2 = \frac{sn}{(n+s)}\left(\frac{1}{s!}\frac{(n+s)!}{n!}\right)^2 \\
&\leq \frac{sn}{(n+s)}(\frac{1}{s!}\frac{e\,(n+s)^{(n+s+\frac{1}{2})}\,e^{-(n+s)}}{\sqrt{2\pi}n^{n+\frac{1}{2}}e^{-n}})^2 \\
&= \frac{e^2}{2\pi}\frac{sn^{2s}}{(s!)^2e^{2s}}\left(\frac{n+s}{n}\right)^{2(n+s)} \asymp \frac{e^2}{2\pi}\frac{sn^{2s}}{(s!)^2e^{2s}}
\end{aligned}$$

□

Theorem 4.13. *The cardinality of code can be bounded from below as the following:*

$$\alpha(L_{s,n}) \geq \frac{2^n(s!)^2e^{2s}}{sn^{2s}}\frac{2\pi}{e^2} \tag{4.8}$$

Proof. In general graphs, the size of independent set multiplied by the chromatic number is greater than or equal to the number of vertices in the graph.

$$\alpha(L_{s,n})\chi(L_{s,n}) \geq 2^n$$

Lemma 4.12 gives the chromatic number. Therefore, we have the following:

$$\alpha(L_{s,n}) \geq \frac{2^n}{\chi(L_{s,n})} \geq \frac{2^n}{\frac{e^2}{2\pi}\frac{sn^{2s}}{(s!)^2e^{2s}}} = \frac{2^n(s!)^2e^{2s}}{sn^{2s}}\frac{2\pi}{e^2}$$

□

Lemma 4.14. *For all fixed s, the lower bound obtained from the coloring is greater than the Levenshtein's lower bound.*

Proof. The ratio between the bound from coloring and the Levenshtein's lower bound is always greater than one:

$$\frac{\frac{2^n(s!)^2e^{2s}}{sn^{2s}}\frac{2\pi}{e^2}}{\frac{2^s(s!)^2 2^n}{n^{2s}}} = \frac{2\pi}{se^2}\left(\frac{e^2}{2}\right)^s \geq 1 \tag{4.9}$$

□

Example For the case of $s = 2$, the lower bound on the size of the codebook is $\frac{4\pi e^2 2^n}{n^4} \approx 23\frac{2^n}{n^4}$, which is greater than the Leveneshtein's lower bound of $\frac{2^{n+4}}{n^4}$.

In the following we are going to see how a single deletion correcting code can be used to construct a double deletion code.

4.4.2 Construction of single deletion correcting code from concatenation

Suppose a set $A = \bigcup_{i=1}^{p} A_i$, $A \subseteq F_2^n$ is given such that $\forall 1 \leq i \leq p : d_L(A_i) > 2$ and $\forall i \neq j : A_i \cap A_i = \emptyset, d_L(A_i, A_j) > 1$. Let β is also a single deletion correcting code of length m with cardinality at least p. We will prove that $m \leq O(\log n)$. For a code $C = \bigcup_{i=1}^{p} A_i(\beta_i) * A_i$, from equation (4.3) the size can be bounded as $|C| = \sum_{i=1}^{p} |A_i|^2 \geq \frac{|A|^2}{p}$.

Sloane [6] proved that for fixed n, $VT_0(n)$ has the largest cardinality between all VT-codes, for any $0 \leq a \leq n$. A lower bound on the size $VT_0(n)$ can be obtained from the following:

$$|VT_0(n)| \geq \frac{2^n}{n+1} \tag{4.10}$$

For sequences with the same length the Levenshtein distance is always an even number since the number of deletions and insertions should be equal.

Suppose $u, v \in F_2^n, u \neq v$, then $d_L(u,v) \geq 2$. So the condition $d_L(A_i, A_j) > 1$ is automatically satisfied. If we take the set A to be F_2^n, then partition the induces graph on A vertices is $L_{1,s}$ graph and the A_i sets are color classes in the graph. A possible coloring for the binary sequences of length n are the coloring from the VT-codes. In this coloring the codes are divided into $n+1$ groups, thus $p = n+1$. The set be should be chosen such that it is a one deletion correcting code. Choosing set β from $VT_0(m)$ suggest that $|VT_0(m)| \geq \frac{2^m}{m+1} > p = n+1$. By taking $m = 2\log n$, we can satisfy the previous equation. The bounds of Levenshtein for one deletion correcting codes

$$\frac{2^{n+1}}{n^2} \leq |C_{1,n}| \leq \frac{2^n}{n}.$$

A code C of length $2n+m$ can be constructed with cardinality:

$$|C| = \sum_{i=1}^{p} |A_i|^2 = |A|^2 = (2^n)^2 \tag{4.11}$$

The upper bound of Levenshtein for a code with equal length says that the cardinality of the optimal code is less than $\frac{2^{2n+m}}{2n+m}$. To see how far we are from the VT-code, we will calculate the ratio of the sizes.

Lemma 4.15. *For a code C of length $n' = 2n+m$ from the concatenation of A_i, β_i and A_i the cardinality is bounded by:*

$$O(n \cdot \log n) \leq \frac{|C_{1,n'}|}{|C|} \leq O(n^2)$$

Proof. The bounds can be calculated from the following:

From the results of concatenation we have $|C| \leq |C_{1,m}| \cdot |A_1|^2$, and since $|A_1| \leq |VT_0(n)| \leq \frac{2^n}{n}$ and $m = 2\log n$

$$\frac{|C_{1,n'}|}{|C|} \geq \frac{\frac{2^{2n+m}}{2n+m+1}}{\frac{2^m}{m+1} \cdot \frac{2^{2n}}{n^2}} = \frac{(m+1)\cdot n^2}{(2n+m+1)}$$
$$= \frac{(2\log n + 1)n^2}{(2n + 2\log n + 1)} = O(n \cdot \log n)$$

$$\frac{|C_{1,n'}|}{|C|} \leq \frac{\frac{2^{2n+m}}{2n+m}}{\frac{2^{2n}}{n+1}} = \frac{(n+1)2^m}{2n+m} = \frac{(n+1)n^2}{2n+2\log n} = O(n^2)$$

□

Definition 12 The rate of a block code C of length n is defined as the ratio of the cardinality of C divided by n, $R = \frac{|C|}{n}$.

Theorem 4.16. *For single deletion, the rate of the code formed from the concatenation asymptotically achieves the rate of optimum code.*

Proof. Suppose $C_{1,n'}$ is the optimum code and C is the code formed from the concatenation of two codes of length n and a buffer of length $m = 2\log n$. From Lemma 4.15, we can see the following:

$$\frac{\log\left(O(n \cdot \log n)\right)}{n} \leq \frac{\log\left(\frac{|C_{1,n'}|}{|C|}\right)}{n} \leq \frac{\log\left(O(n^2)\right)}{n}$$

As n goes to infinity, from the sandwich theorem, $\lim_{n\to\infty} R_{opt} - R = 0$. □

4.4.3 Construction of 2-deletion correcting codes from coloring of VT codes

In this section, we will study the recursive construction of double deletion correcting codes, $C_{2,n}$ code from $s = 1$ deletion correcting codes. Fortunately, for $s = 1$, there is an algebraic construction for these codes. The VT-code, introduced by Levenshtein is conjectured to be the optimal code, which is the maximum independent set in $L_{1,n}$ graph [6].

Suppose we are interested in finding double deletion correcting codes. The recursive construction introduced in this chapter is such that the set A is one deletion correcting code. If we take $VT_a(n)$ as this A set, and find a coloring for the graph induced by these vertices, we would be able to find the A_i partitions. As proved in [6], $VT_0(n)$ has the largest size between all VT-codes, we would take this code as the A set.

Lemma 4.17. *The size of the 2-deletion correcting code obtained by the coloring of the induced subgraph of the $L_{2,n}$ graph on $VT_0(n)$ is greater than $\frac{4\pi e^2 2^n}{n^4(n+1)}$.*

Proof. In the previous section we discussed about the partitioning and its relation to the chromatic number of a graph. We saw that the number of parts is equal to the chromatic number of the induced subgraph of $L_{2,n}$. Moreover, the chromatic number of the induced subgraph of a graph is not more than the chromatic number of the original graph. Therefore, $\chi\left(L_{2,n}\left(VT_0(n)\right)\right) \leq \chi\left(L_{2,n}\right) \leq \frac{e^2}{2\pi}\frac{2n^{2*2}}{(2!)^2e^{2*2}} = \frac{n^4}{4\pi e^2}$. Hence, we can bound the size of the largest color class by dividing the size of the subgraph by chromatic number. The size of $VT_0(n)$ code is greater than $\frac{2^n}{n+1}$ [6]. Therefore, we will get the $\frac{4\pi e^2 2^n}{n^4(n+1)}$ bound. $\square$

4.5 Conclusion

In this chapter, we studied a concatenation method for the construction of larger deletion and insertion correcting codes based on smaller codes. Specifically we showed how our construction works for special cases such as single and double deletion correcting codes. Moreover, we proved a new lower bound on the size of the general codes based on a coloring argument. The new lower bound on the size of s-deletion correcting code is larger than the lower bound of Levenshtein [3].

CHAPTER 5

CONCLUSION

We introduced a new method to search for multiple insertion/deletion error correcting codes. Our approach works for any number of deletion and insertions s and any code length n. Our approach first recovers a codebook based on a heuristic algorithm which finds the maximal independent set in an appropriately defined graph and then uses an optimal concatenation technique to find codebooks that accommodate a larger code length n.

The heuristic algorithm does far better than the optimal concatenation approach if it could be applied to larger n. However, the complexity of the algorithm renders this method infeasible for n beyond 25 (although this is already an improvement over the existing work). Hence, for larger n, concatenation technique is employed. It is obvious that the concatenation principle is not particularly efficient for constructing codebooks of larger length from codebooks of smaller length. However, given lack of alternatives, by starting from a larger initial codebook found through the heuristic algorithm, we construct codebooks of much larger cardinalities compared to previous works. We surmise that the superior performance of the heuristic algorithm is in fact the result of special structure of the graph. In future, we plan to use this special structure to come up with better constructions for deletion and insertion codes. It will be interesting to see if the structure of the graph may also be utilized to find efficient decoding algorithms.

In future, we can use this special structure to come up with better constructions for deletion and insertion codes. The concatenation method gives some idea for designing an exact algorithm that uses the special structure of these graphs. We hope that the structure of the graph may also be utilized to find efficient decoding algorithms.

APPENDIX A

COMPLEXITY OF THE GREEDY ALGORITHM

A.1 Time complexity and performance of the greedy algorithm

Using Min-Heap data structure [24] to store the vertices indexed by the values of $d(v)$, we can easily find the vertex with minimum degree in $O(1)$. Initializing the heap time costs $O(|V|)$ and we can remove the elements or change the index values ($d(v)$), and then update the Heap in $O(\lg(|V|))$. The algorithm starts with a continuous range of numbers from 0 to $2^n - 1$ representing vertices in V, so we can use an index array to locate the nodes in Min-Heap in $O(1)$.

Considering the fact that each node is removed exactly once from the heap ($O(|V|.\lg(|V|))$) and the total number of updates on the values of $d(v)$ cannot exceed the number of edges in the graph ($O(|E|.\lg(|V|))$), we can compute the time complexity of Min-Degree Selection algorithm (since each node has at least one neighbor, then we have $|V| \in O(|E|)$):

$$O(|V|) + O(|V|.\lg(|V|)) + O(|E|.\lg(|V|))$$

$$= O(|E|.\lg(|V|)) = O(|E|.n). \qquad \text{(A.1)}$$

Although this algorithm is polynomial regarding the size of the graph ($|V|$), we should keep mind that linear increment of n leads to exponential growth of the size of the graph (V and E). Hence, this algorithm is non-polynomial regarding the length of the codes, which is the real parameter of the problem. As a result, using this algorithm for large values of n is not practical. However, it is worth mentioning that this approach executed on a normal PC, can provide us with codebooks of length 20 in less than one hour and codebooks of length 25 in less than one day.

APPENDIX B

DEGREE OF ONE DELETION GRAPHS

B.1 Definitions and background

Let $G = (V, E)$ be an undirected graph where each vertex $v \in V$ is a binary sequence of length n. There are $|F_2^n| = 2^n$ vertices in this graph. An edge is drawn between two vertices u and v if any of the subsequences they produce after any s number of deletions are the same. We are interested in finding a formula for the degree of this graph. For $s = 1$ we proved the exact formula for the degree of each vertices by using the results Levenshtein proved in [3] and [18].

For a binary sequence X of length n, let $D_s(X)$ and $I_s(X)$ denote the set of all binary sub-sequences and super-sequences obtained from X by s number of deletions and insertions respectively. Let τ be the number of runs in the sequence X. Levenshtein proved formulas for the maximum number of sub and super sequence, any two binary sequences of length n can produce after s-deletions and insertions [3], [18].

For all binary sequences of length n, let $N^-(n, s)$ and $N^+(n, s)$ denote the maximum size of the common subsequences and super sequences of length $n - s$ and $n + s$ respectively.

$$N^-(n,s) = \max_{x,y \in F_2^n} |D_s(x) \cap D_s(y)| = 2\sum_{i=0}^{s-1} \begin{pmatrix} n-s-1 \\ i \end{pmatrix} \tag{B.1}$$

$$N^+(n,s) = \max_{x,y \in F_2^n} |I_s(x) \cap I_s(y)| = 2\sum_{i=0}^{s-1} \begin{pmatrix} n+s-1 \\ i \end{pmatrix} \tag{B.2}$$

In the case where $s = 1$ the above formulas will be reduced to the following: $D(X) = \tau$, $I(X) = n + 2$ and $N^+(n, 1) = N^-(n, 1) = 2$.

The Levenshtein distance $d_L(x, y)$ between two sequence x and y is defined as the minimum number of insertions, deletions and substitutions required to convert one sequence to the other.

B.2 Degree for 1-deletion graph

Lemma B.1. $\forall u, v \in V : (u, v) \in E \iff d_L(u, v) \leq 2s$

Proof. If $(u, v) \in E$ then $D_s(u) \cap D_s(v) \neq \emptyset$. Suppose $z \in D_s(u) \cap D_s(v)$ then $\{u, v\} \in I_s(z)$ then there is a sequence of s deletion and s insertion operations to convert u to v, therefore $d_L(u, v) \leq 2s$. Conversely, if $d_L(u, v) \leq 2s$ since u and v have the same length, the number of insertions and deletions are equal. Then, there is a sequence of at most s deletions and s insertions converting u to v. If we consider the deletion operations first, then $\exists z \in D_s(u)$ such that $v \in I_s(z)$. Therefore, $z \in D_s(v)$ and $D_s(u) \cap D_s(v) \neq \emptyset \Rightarrow (u, v) \in E$. □

Lemma B.2. *Let $x, y \in F_2^n$ then*

$$\max_{x,y \in F_2^n} |I(x) \cap I(y)| = 2$$

$$\max_{x,y \in F_2^n} |D(x) \cap D(y)| = 2$$

Proof. This is the special case of (B.1) and (B.2) when $s = 1$. □

Lemma B.3. *Let $x \in F_2^n$ and for $m \geq 3$, $\{x_1, \ldots, x_m\} \subseteq D(x)$ be a set of m subsequences of x after one deletion then*

$$|I(x_1) \cap \ldots I(x_m)| = 1 \tag{B.3}$$

Proof. Levenshtein [18] proved that for a given sequence x, $N^-(n, s) + 1$ number of subsequences of x is enough to uniquely construct it and he gave an algorithm for the construction. In one deletion case, for any $m \geq N^-(n, 1) + 1 = 2 + 1$, it is possible to construct x from m members of $D(x)$ and that construction is unique. As a result, $|I(x_1) \cap \ldots I(x_m)|$ should be at most one.

The sequence x belongs to this intersection so the cardinality should be at least one. If there were another sequence y in the intersection then $x_1, \ldots, x_m$ are also subsequences of y. It is a contradiction since the construction from subsequences is unique, then $x = y$. □

Suppose x is a sequence of length n with τ number of runs. Let $D(x)$ be the set of all subsequences of x after one deletion. Since sequence x has τ number of runs, $|D(x)| = \tau$ and each subsequence is obtained from deletion of one bit from a given run. If a subsequence was obtained from deletion of a bit from i'th run, we call that subsequence x_i. Therefore we can right the set $D(x)$ as the following: $D(x) = \{x_1, \dots, x_\tau\}$. Define $A_i = I(x_i)$ the set of all supersequences of x_i after one insertion where x_i is a subsequence of x.

Lemma B.4. *Let $N(x)$ denotes the set of all vertices in the neighborhood of x. Then we have the following:*

$$N(x) \cup \{x\} = A_1 \cup \cdots \cup A_\tau$$

Proof. The sequences in the neighborhood of x have Levenshtein distance of 2 to it. The set $A_1 \cup \cdots \cup A_\tau$ has all possible sequences obtained from x by one deletion and one insertion. So, excluding the sequence itself, all the members of this set are within distance 2 as a result of lemma B.1, they are in neighborhood of x. □

Suppose $x \in F_2^n$ and is written in the run-length representation. For $1 \le i \le n-2$ define p_i as the number of i consecutive single runs excluding the extreme runs. In other words, in the run length representation, p_1 is the number of $-1-$ patterns, p_2 is the number of $-11-$ patterns, p_3 is the number of $-111-$ patterns and so on.

For example, if $x = 01011010$, the run length representation is $x = 1112111$. The values of p_i can be calculated as the following. We can easily see $p_1 = 4$ because it is the number of middle runs with length 1, $p_2 = 2$ since we have two of $-11-$ patterns in the sequence as depicted by an over line $1\overline{11}2\overline{11}1$ and $p_3 = p_4 = ... = p_6 = 0$ since there are no more than 2 consecutive one-runs in the sequence.

Lemma B.5. *Let $x \in F_2^n$, $x_i \in D(x)$, $A_i = I(x_i)$ and $[\tau] = \{1, 2, \dots, \tau\}$ then*

$$\sum_{i,j} |A_i \cap A_j| = \binom{\tau}{2} + \tau - 1 + \sum_{i=1}^{n-2} p_i$$

Proof. For all i, $x \in A_i$ then $|A_i \cap A_j| \ge 1$ and from lemma B.2,

$\max_{x_i,x_j \in D(x)} |I(x_i) \cap I(x_j)| = \max_{i,j} |A_i \cap A_j| = 2.$

$$\sum_{i,j} |A_i \cap A_j| = \sum_{i,j:|A_i \cap A_j|=1} |A_i \cap A_j| + \sum_{i,j:|A_i \cap A_j|=2} |A_i \cap A_j|$$

Therefore, the vertices in these sets can be divided into two multi-sets N_1 and N_2 as the following:

$$N_1 = \{z | z \in A_i \cap A_j, |A_i \cap A_j| = 1, \forall i, j \in [\tau]\}$$
$$N_2 = \{z | z \in A_i \cap A_j, |A_i \cap A_j| = 2, \forall i, j \in [\tau]\}$$

Let $N = N_1 \cup N_2$ be another multi-set. From the above definitions, we can conclude that $\sum_{i,j} |A_i \cap A_j| = |N|$. For all $i, j \in [\tau]$, $x \in A_i \cap A_j$ therefore, $x \in N_1 \cup N_2$ So the number of appearance of x is equal to $\binom{\tau}{2}$. Since we already counted all elements in N_1, the remaining elements should belong to N_2. Next, we will demonstrate that $|N_2/\{x\}| = \tau - 1 + \sum_{i=1}^{n-2} p_i$ and as a result the proof is complete.

$$\sum_{i,j} |A_i \cap A_j| = |N_1 \cup N_2|) = |N| = (\binom{\tau}{2}) + \tau - 1 + \sum_{i=1}^{n-2} p_i$$

To show $|N_2/\{x\}| = \tau - 1 + \sum_{i=1}^{n-2} p_i$, we need to count all possible cases where $|I(x_i) \cap I(x_j)| = 2$. This can occur in two different scenarios.

In the first senario, there are $\tau - 1$ ways to get a new sequence if we are substituting two bits of consecutive runs. It is equivalent of deleting one bit from the sequence and then insert the complement of that bit to the next position. If x was the original sequence and y is the new sequence made with substitution operation then there are two sequences x_i and x_j such that $\{x, y\} = I(x_i) \cap I(x_j)$ for all i, j such that $j = i + 1$. We should note that x_i is the subsequence obtained from x by deleting a bit from i'th run. There are $\tau - 1$ ways to choose two consecutive runs and as a result, $\tau - 1$ different sequences can be produced this way.

The second scenario is not that obvious. We will prove that generating two different super-sequence from two subsequences of a given x is possible only if we have consecutive one-runs. Suppose x_i and x_j are two subsequences of x obtained from deletion of one bit of the i'th and the j'th runs in x

respectively. Here assume $j \geq i+2$ since we have already considered the case where $j = i+1$ in the first scenario. If we denote x in run-length representation as $x = R_1 R_2 \dots R_\tau$ where $R_i \geq 1, \sum_{i=1}^{\tau} R_i = n$. Then $x_i = R_1 \dots R_{i-1} R'_i R_{i+1} \dots R_\tau$ and $x_j = R_1 \dots R_{j-1} R'_j R_{j+1} \dots R_\tau$, $R'_i = R_i - 1$ and $R'_j = R_j - 1$.

$$\begin{array}{l}
x_i = \boxed{R_1 \dots R_{i-1}}\, R'_i\, \boxed{R_{i-1} \dots R_{j-1}}\, R_j\, \boxed{R_{i-1} \dots R_\tau} \\
x_j = \boxed{R_1 \dots R_{i-1}}\, R_i\, \boxed{R_{i-1} \dots R_{j-1}}\, R'_j\, \boxed{R_{i-1} \dots R_\tau}
\end{array}$$

Figure B.1: Sequence alignment

We should insert a bit into each sequence in a way that the resulting super sequences are equal. One way of getting the same super-sequence is to insert back the bit we already deleted. We have already considered this case. The other way is inserting the bit in a way that two strings match each other. We claim that it is possible only if the string $R_{i-1} \dots R_{j-1}$ is $111 \dots 1$ string. In other word, the only possible case for this is $0101 \dots 01$ and $1010 \dots 10$ sequence. So where ever we have a string of the form $11 \dots 1$ in the sequence, we can get two subsequences with this string as the middle string and as a result we can get two common super-sequences. The number of such cases is equal to the number of different p_i patterns in the original sequence x. $\square$

Theorem B.6. *The degree of x is*

$$Deg(x) = \tau(n-1) - \sum_{i=0}^{n-2} p_i + 1$$

Proof.

$$\begin{aligned}
Deg(x) &= |A_1 \cup \dots \cup A_\tau| - 1 \\
&= \sum_{i=1}^{\tau} |A_i| - \sum_{i,j} |A_i \cap A_j| + \dots + (-1)^{\tau+1} |A_1 \cap \dots \cap A_\tau| - 1 \\
&= \sum_{i=1}^{\tau} |A_i| - \sum_{i,j} |A_i \cap A_j| + \left(\sum_{m=3}^{\tau} (-1)^{m+1} \sum_{T \subseteq [\tau], |T| = m} |\bigcap_{j \in T} A_j| \right) - 1 \\
&= \tau(n+1) - \left(\binom{\tau}{2} + \tau - 1 + \sum_{i=1}^{n-2} p_i \right) + \left(1 - \tau + \binom{\tau}{2} \right) - 1
\end{aligned}$$

$$= \quad \tau(n-1) - \sum_{i=0}^{n-2} p_i + 1$$

From (2.6) we have $|A_i| = |I(x_i)| = n+1$. In lemma B.3 we proved that for all $m \geq 3$ sets of subsequences, $|A_1 \cap \cdots \cap A_m| = 1$. Let $[\tau] = \{1, 2, \ldots, \tau\}$

$$\sum_{\substack{T \subseteq [\tau] \\ |T| = m}} |\bigcap_{j \in T} A_j| = \sum_{\substack{T \subseteq [\tau] \\ |T| = m}} 1 = \binom{\tau}{m}$$

$$\begin{aligned}\sum_{m=3}^{\tau} (-1)^{m+1} \sum_{T \subseteq [\tau], |T|=m} |\bigcap_{j \in T} A_j| &= -\sum_{m=3}^{\tau} (-1)^m \sum_{T \subseteq [\tau], |T|=m} 1 \\ &= -\sum_{m=3}^{\tau} (-1)^m \binom{\tau}{m} \\ &= -\left(\sum_{m=0}^{\tau} \binom{\tau}{m} (-1)^m - \left(1 - \tau + \binom{\tau}{2}\right)\right) \\ &= 1 - \tau + \binom{\tau}{2}\end{aligned}$$

□

Example 1 For $X = 01011010$ from simulations, the degree is 44. We can calculate it from the above formula as the following: first, let's convert it to a run length representation: $X = 1112111$. Then we should find out the values of p_i. It is easy to see $p_1 = 4$ because it is the number of middle runs with length 1, $p_2 = 2$ since there is two of $-11-$ patterns in the sequence as depicted by an over line $1\overline{11}2\overline{11}1$ and $p_3 = p_4 = ... = p_{n-2} = 0$ since there are no more than 2 consecutive one-runs in the sequence. We also know that $\tau = 7$ and $n = 8$.

$$Deg(1112111) = 7 * (8-1) - (4+2) + 1 = 44.$$

Example 2 For $X = 01010101$ the degree is 36 from the simulation, the run length representation is $X = 11111111$, $n = 8$ and $\tau = 8$, $p_1 = 6$, $p_2 = 5$,

$p_3 = 4$, $p_4 = 3$, $p_5 = 2$, and $p_6 = 1$. The degree is :

$$Deg(11111111) = 8 * (8-1) - (6+5+4+3+2+1) + 1 = 36.$$

Consider X as a binary sequence of length n, which is written in run-length representation. $X = x_1x_2\dots x_n = R_1R_2\dots R_\tau$ where $\sum_{i=1}^{\tau} R_i = n$ and $R_i \geq 1$ for $i = 1, 2, \dots, \tau$. There are $\binom{n-1}{\tau-1}$ of such sequences in $\{0,1\}^n$.

Proposition 1. *Starting from a sequence of length n with τ runs, we can obtain sequences with $\tau, \tau+1, \dots, \tau+2s$ runs after insertion of$s$ bits.*

Proof. In order to prove this proposition, we will consider the effect of insertion of one symbol, on the number of runs in a binary sequence.

I Insertion to existing runs

There are τ number of runs in the sequence and we can obtain a different sequence by insertion of one bit to each of them. So that, there are τ number of sequences of length $n+1$ with τ number of runs that can be obtained from a sequence of length n with τ runs.

II Insertion to extrems

There are two ways of insertion of one bit to the extreme of $X = R_1R_2\dots R_\tau$, insertion to the start of the string or insertion to the end of the string. Two sequences with $\tau+1$ runs will be the result of this insertion: $1R_1R_2\dots R_\tau$ and $R_1R_2\dots R_\tau 1$.

III Insertion in the middle of a run and split it.

There are $R_i - 1$ ways to split a run of R_i bits. For sequence $X = R_1R_2\dots R_\tau$, there are $\sum_{i=1}^{\tau}(R_i - 1) = \sum_{i=1}^{\tau} R_i - \tau = n - \tau$ ways of obtaining a sequence with $\tau+2$ number of runs by one insertion.

By a recursive way of insertion, we will get the result in the proposition.

□

When $s = 2$, the number of super-sequences can be classified based on the number of runs as the following:

1. $\tau \rightarrow \tau$

   $\binom{\tau}{1} + \binom{\tau}{2}$

   The first term is the case when we insert both symbols to one run and the second term is when we insert into two different runs.

2. $\tau \rightarrow \tau + 1$

   $\binom{\tau}{1}\binom{2}{1} + 2$ There are two cases, the first case is when we insert one bit to the extreme and the other to the existing runs, the second case is when we insert two of the same symbol to one of the extremes.

3. $\tau \rightarrow \tau + 2$

   $3 + (n - \tau)(\tau + 1) - 1$

4. $\tau \rightarrow \tau + 3$

   $2(n - \tau)$ One bit is inserted into one of the existing runs and split it and the other was added to one of the extremes. There are $n - \tau$ ways to split a run and there are two ways of choosing one of the extremes.

5. $\tau \rightarrow \tau + 4$

   $\binom{n-\tau}{2}$

To confirm that the number of super-sequences we have counted above covers all possible cases of insertion of two bits, we show that the summation over all cases is equal to the formula that levenshtein obtained for number of super-sequences of a binary sequence of length n.

$$\binom{\tau}{1} + \binom{\tau}{2} + \binom{\tau}{1}\binom{2}{1} + 2 + 3 + (n - \tau)(\tau + 1) - 1 + 2(n - \tau) + \binom{n - \tau}{2}$$
$$= \frac{n^2 + 5n + 8}{2}$$

APPENDIX C

NUMERICAL RESULTS

C.1 Some numerical results

In the following, we have listed some of the results for small n and $s = 2$.

- n=6
  - Optimum Solution = $[000000, 000111, 111000, 111111]$

    $deg = [29, 72, 115, 126]$ respectively.

    $\sum_{i=1}^{4} deg(c_i) = 342 = 2.6719 * 64$
  - Greedy Algorithm = $[101000, 010011, 111000, 111111]$

    $deg = \{47, 64, 61, 60\}$

    $\sum_{i=1}^{4} deg(\bar{c_i}) = 232 = 3.6250 * 64$
- n=7
  - Optimum Solution = $[0110101, 1000000, 1000111, 1111000, 1111111]$

    $deg = [103, 122, 124, 127, 128]$.

    $\sum_{i=1}^{5} deg(c_i) = 604 = 4.7188 * 128$.
  - Greedy Algorithm = $[0110010, 1111111, 1111000, 0001111, 0101010]$

    $deg = [105, 118, 118, 124, 124]$

    $\sum_{i=1}^{5} deg(c_i) = 589 = 4.6016 * 128$.
- n=8
  - Optimum Solution

    [00111110,01001100,10000000 ,10000111,11010101,11110000,11111111]

    $deg = [128, 171, 202, 202, 214, 225, 225]$.

    $\sum_{i=1}^{7} deg(c_i) = 1367 = 5.3398 * 256$.

  - Greedy Algorithm

    [01000110,11111111,11111000,00011111,11000001,00110100,10101011]

    $deg = [113, 126, 128, 128, 128, 128, 128]$

    $\sum_{i=1}^{7} deg(c_i) = 879 = 6.8672 * 256.$

- n=9

  - Optimum Solution

    [000000000,000000111,000111110,001101000,010001100

    ,011011011,100100101,111000000,111000111,111111000,111111111]

    degree

    $[37, 98, 187, 245, 248, 256, 256, 256, 256, 256, 256]$

    $\sum_{i=1}^{11} deg(c_i) = 2351 = 9.1836 * 512$

  - Greedy Algorithm = [001100100,111111111,111111000,111000000, 000111111,

    000000111,000111000,111000111, 010100100,101110101]

    $deg = \{25229834937743446547150350751 1\}$

    $\sum_{i=1}^{11} deg(\bar{c}_i) = 4167 = 8.1387 * 512$

- n=10

  - Optimum Solution

    [0000000000 ,0000000111 ,0000101010 ,0000111111 ,0001111000 ,0101010111 ,0110010001 ,0111110011 ,1000110011 ,1010110000 ,1100111101 ,1110000000 ,1110000111 ,1110101010 ,1111111000 ,1111111111]

    degree

    [56 , 162 , 393 , 472 , 586 , 744 , 871,951 , 965 , 989 , 1013 , 1017 , 1018 , 1020,1023,1024]

    $\sum_{i=1}^{16} deg(c_i) = 12304 = 12.0156 * 1024$

  - Greedy Algorithm

    [0010010110, 1111111111, 1110000000, 0000000111, 1111110000, 0001111111, 0000111100, 1100111110, 1111000111, 1110110101, 0101010111, 1000110011, 1010001010, 0110010001, 0010110000] degree

{56 ,112, 218 ,313, 412, 508, 634, 740 ,823, 896, 938, 978 ,1008, 1022 ,1024 }

$\sum_{i=1}^{15} deg(\bar{c_i}) = 9682 = 9.4551 * 1024$

- $n = 11$

  Greedy Algorithm

  [00000000000 ,11111111111 ,11111111000 ,00011111111 ,11111000000 ,00000011111 ,11000000001 ,11100000111 ,01000000110 ,00000111000 ,00110100000 ,10100110000 ,00001010101 ,11000101100 ,00101100011 ,11001011111 ,00110011110 ,10101001101 ,10011101010 ,11111001011 ,01111001100]

  Degree=[67 ,134 ,267 ,388 ,520 ,652 ,795 ,965,1120, 1251, 1378 ,1513, 1630, 1728, 1827, 1909, 1960, 2002, 2034, 2044, 2048]

  $\sum_{i=1}^{21} deg(\bar{c_i}) = 26232 = 12.8086 * 2048$

- $n = 12$

  Greedy Algorithm

  [000000000000 ,111111111111 ,111000000000 ,000000000111 ,111111000000 ,111111111000 ,000111111111 ,000000111111 ,000000111000 ,000111000000 ,000111111000 ,111000000111 ,111000111111 ,111111000111 ,110000111100 ,001111000011 ,101011111101 ,011110011110 ,111110101010 ,111010111001 ,001101110110 ,101110011000 ,011011010000 ,110001100001 ,010100000010 ,100101000100 ,000011000110 ,010011101001 ,100000110101 ,110100100110 ,100101100111 ,010100010111]

  deg=[79, 158, 321, 471, 633, 770, 920, 1056,1292, 1498 1682 1901 2093 2260 2479 2698 2913 3056 3214 3321 3449 3568 3642 3757 3852 3921 3980 4026 4055 4080 4094 4096]

  $\sum_{i=1}^{32} deg(\bar{c_i}) = 79335 = 19.3689 * 4096$

REFERENCES

[1] J. Orth and S. Houghten, “Optimizing the salmon algorithm for the construction of dna error-correcting codes.” *IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology.*, vol. 17, 2011.

[2] J. Sun, S. Houghten, and J. Ross, “Bounds on edit metric codes with combinatorial dna constraints.” *Congressus Numerantium*, vol. 204, pp. 65–92, 2010.

[3] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions and reversals,” *Soviet Physics - Doklady*, vol. 10, no. 8, pp. 707–710, February 1966.

[4] F. McWilliams and N. Sloane, *The Theory of Error Correcting Codes.* Amsterdam, The Netherland, North Holland: North Holland publishing company, 1977.

[5] J. D. Ullman, “On the capabilities of codes to correct synchronization errors,” *IEEE Trans. Inf. Theory*, vol. IT-13, no. 1, pp. 95–105, January 1967.

[6] N. Sloane, “On single-deletion-correcting codes,” *Codes and Designs, Ohio State University*, pp. 273–292, May 2002.

[7] T. Swart and H. Ferreira, “A note on double insertion/deletion correcting codes,” *IEEE Transaction on Information Theory*, vol. 49, no. 1, pp. 269–273, 2003.

[8] M. Mitzenmacher, “A survey of results for deletion channels and related synchronization channels,” *Probab. Surveys*, vol. 6, pp. 1–33, 2009.

[9] N. Sloane., “Challenge problems: Independent sets in graphs.” [Online]. Available: http://www2.research.att.com/ njas/doc/graphs.html

[10] D. J. M.R. Gary, *Computers and interactability: A Guide to the Theory of NP-Completeness.* New York: Freeman, 1979.

[11] F. Khajouei, M. Zolghadr, and N. Kiyavash, "An algorithmic approach for finding deletion correcting codes," *Information Theory Workshop (ITW), 2011 IEEE*, pp. 25 –29, oct. 2011.

[12] S. Haykin, *Communication Systems.* John Wiley and Sons Inc, 2001.

[13] H. Mercier, V. Bhargava, and V. Tarokh, "A survey of error-correcting codes for channels with symbol synchronization errors," *IEEE Communications Surveys and Tutorials*, pp. 87–96, 2010.

[14] R. R. Varshamov and G. M. Tenengol'ts, "Correction code for single asymmetrical errors," *Automat. Telemekh.*, vol. 26, pp. 288 – 292, 1965.

[15] A. S. J. Helberg and H. C. Ferreira, "On multiple insertion/deletion correcting codes," *IEEE Trans. Inform. Theory*, vol. 48, pp. 305–308, January 2002.

[16] S. Butenko, P. Pardalos, I. Sergienko, V. Shylo, and P. Stetsyuk., "Finding maximum independent sets in graphs arising from coding theory," *Proceedings of the 2002 ACM Symposium on Applied Computing*, pp. 542–546, 2002.

[17] R. L. Brooks, "On colouring the nodes of a network," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 37, pp. 194–197, 1941.

[18] V. I. Levenshtein, "Efficient reconstruction of sequences from their subsequences or supersequences," *J. Comb. Theory*, vol. 93, no. 2, pp. 310 – 332, 2001.

[19] P. M. Pardalos and J. Xue, "The maximum clique problem," *Journal of Global Optimization*, vol. 4, no. 3, pp. 301–328, 1994-04-01. [Online]. Available: http://dx.doi.org/10.1007/BF01098364

[20] N. Z. Shor, "Dual quadratic estimates in polynomial and boolean programming." *Ann. Oper. Res.*, vol. 25, no. 1, p. 163168, 1990.

[21] P. M. Pardalos and G. P. Rodgers., "A branch and bound algorithm for the maximum clique problem." *Computers Ops Res.*, vol. 19, p. 363375, 1992.

[22] J. Abello, S. Butenko, P. Pardalos, and M. Resende, "Finding independent sets in a graph using continuous multivariable polynomial formulations," *Journal of Global Optimization*, vol. 21, no. 2, pp. 111–137, 2001.

[23] T. S. Motzkin and E. G. Straus., "Maxima for graphs and a new proof of a theorem of turan." *Canad. J. Math.*, vol. 17, p. 533540, 1965.

[24] R. L. R. Thomas H. Cormen, Charles E. Leiserson, *Introduction to algorithms.* MIT Press / McGraw-Hill., 1990.