SC 402

Introduction To Cryptography

Prof.Manish Gupta

Discipline : Coding Theory

Title: Classification of binary codes as single deletion correcting codes

Group 17

Name	ID
Nilay Patel	201901281
Shubham Kevadiya	201901284
Aditya Srivastava	201901265
Venkatesh Banoth	201901233
Chethan Maloth	201901250

May 14, 2022

Contents

Abstract

We introduce Classification of binary codes on basis of single deletion correcting code in this paper. We also looked upon Varshamov-Tenengolts code for basic understanding of how single deletion works on binary codes classified through Varshamov-Tenengolts code . Our general result might have more applications/implications in information theory, computer science, and mathematics.

1 Introduction

Classifying set of codes that can handle single deletion is generally acknowledged to be a difficult subject. Varshamov-Tenengolts (VT) codes are one of such codes which helps us in classifying set of codes than can be used to rectify a single deletion or insertion . We have tried implementing the VT code algorithm and have tried to verify the working of VT codes for only Binary Field but to make it work for more general field like ternary or quaternary we will need further modification or enhancement to be made in the algorithm developed. VT code algorithm is only limited to Binary codes and to tackle this problem we have tried to classify the codes that can handle single deletion or insertion . For classification of codes we have used the concept of Hamming Distance and generator matrix to extract basics linear codes that acts as basis vector for our classification algorithm . This basic vector are than used to generate a set of codes and set is test on single deletion mainly. Nonetheless, our understanding of these codes with these types of faults is still limited, and there are other open difficulties in the domain, when contemplating deletion correcting code constructs that satisfy additional requirements.

2 VT code

Let $\mathrm{F^n}_2$ be the set of all binary sequences/strings with length n $\in$ N, i.e., $F^n{}_2 = \{x \mid x = x_1x_2 \ldots\ldots x_n, such that x_i \in \{0,1\}, i = 1, \ldots\ldots.n\}$ where n is a positive integer. Varshamov-Tenengolts $\mathrm{VT_a(n)}$ code is the set of all the binary n-tuples $\langle \mathrm{S_1}, \ldots\ldots.., \mathrm{S_n} \rangle$ where:

$$\sum_{i=1}^{n} is_i \equiv a (\mathrm{mod} n + 1)$$

Let $\mathbf{a} \in \mathbf{Z}_{n+1}$ Cardinality will be highest when a=0 , we will see these in below examples

Example 1: n=7 and a=1
$VT_1(7)$=0001011,0001100,0010010,0011111,0100001,0101110,0110101,0111000, 1000000, 1001101,1010011,1010100,1100010,1101111,1110110,1111001

Example 2: n=8 and a=0
$VT_0(8)$={00000000,00001110,00010101,00011000,00100011,00100100, 00111011,00111100,01000010,01010111,01011010,01100110,01101001,01110000, 01111110,10000001,10001111,10010110,10011001,10100101,10101000,10111101, 11000011,11000100,11011011,11011100,11100111,11101010,11110001,11111111}

Table 1: Number of Codes

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

Analysis of above table says that cardinality will be highest for a=0 for any values of n. And same can be formulated as a result of observation in below equation which states that cardinality of for any n will be highest for a=0 and least for n=1.

$$|VT_0(n)| \geq |VT_a(n)| \geq |VT_1(a)| .$$

2.1 Single Deletion correcting code

A single deletion correcting code for a string of length n is a set C (also called as Dset) $\subseteq F_2^{n_2}$ such that $D(x) \cap D(y) = \Phi forallx, y \in C$. A single

deletion that is optimal has the largest cardinality. All strings generated through any code (like VT, HelBerg etc) are than tested for n bit error detection . For single Error detection n=1. Error detection considers all strings generated
Here are some examples of Dset created for different values of a and n to check for deletion of single bit.

Example 1 :
n=5 a=0 q=2 (field Z_2)
Dset created for all codewords generated by VT code are:

00000: 0000
00111: 0011 0111
01010: 0010 0100 0101 0110 1010
10001: 0001 1000 1001
11011: 1011 1101 1111
11100: 1100 1110

Above generated Dsets for n=5 and a=0 will be able to handle single bit deletion as intersection of all possible pairs of Dsets will be equal to NULL.

Example 2 :
n=3 a=0 q=3(field Z_3)
Dset created for all codewords generated by VT code are:

000: 00
012: 01 02 12
020: 00 02 20
101: 01 10 11
121: 11 12 21
202: 02 20 22
210: 10 20 21
222: 22

Above generated Dsets for n=3 and a=0 will not be able to handle single bit deletion as each and every subsequence formed from supersequence is being repeated in one or more different Dsets , So intersection between Dsets will not be equal to NULL and as a result will not handle single bit deletion.

Despite the fact that the VT codes can only fix a single loss, they and their derivatives have a wide range of uses, including DNA-based data storage, and distributed message synchronisation.

2.2 Algorithm

Algorithm: **generateAllKLength**

```
Input : ch(passed as reference), prefix, q, n, s( passed as
   reference), num, l, a, N
Output: NULL ( all required output from algorithm is stored
   internally in array of string s)

    if n equals to 0 , than
       if num%(N+1) equals to a
           Insert in s => prefix
       end if
    end if


   for  i belong to 0,....,q do

       newPrefix = prefix + ch[i]
       pl=prefix.length()+1
       tmp= (pl^l) *(ch[i]-'0')
       generateAllKLengthRec(ch, newPrefix, q, n -
           1,s,num+tmp,l,a,N)

    end for

return
```

Algorithm: **generateFeildElements**

```
Input : q, ch(passed as reference)
Output : NULL ( as result is directly stored in character array ch)

   if q greater than 5 OR q smaller than equal to1 than
```

```
        return

    Insert in ch =>   0
    Insert in ch =>   1
    if q equals to 3 than
        Insert in ch =>   2
    else if q equals to 5 than
        Insert in ch =>   3
        Insert in ch =>   4
        Insert in ch =>   5
     end if

return
```

Algorithm: **checkSingleDeletion()**

```
Input : Dset (passed as reference ) , s (passed as reference) , n
Output : bool (true => can handle single deletion / false =>
    cannot handle single deletion )

    unordered_map<string,int> stringmap;

    for i belongs to 0;....; size of s do
    {
        for  j belongs to 0;...; n do
        {
            string = si
            Erase tempj
            Insert in Dseti => temp

            // Comment from here till line 84 to print Dset.
            if stringmaptemp equals to end of stringmap tahn
                stringmaptemp=i
            else
                if stringmaptemp not equals to i
                    return false
            end if

         end for
    end for
```

```
return true
```

3 Classification Code theory

Motivation : The VT code was only good for fields of 2 and couldn't handle fields larger than that. Therefore to check which code in terms of (n,k,d) can handle single bit deletion, we wrote a classification code. Generator matrix with (n,k) was generated. A generator matrix is a matrix whose rows constitute the foundation for a linear code in coding theory. The linear code is the row space of its generator matrix, and the codewords are all of the linear combinations of the rows of this matrix.

$\mathbf{G}_{(\mathrm{n,k})} = \left[\mathrm{I}_{(\mathrm{k,k})} \mid \mathbf{P}_{(\mathrm{k,n-k})}\right]$

Where I is identity matrix of order kxk P is all possible permutations of binary matrix of order kx(n-k) Total number of possible matrix for (n,k) are $2^{k^*(n-k)}$

Code for generating matrix is given below:

Algorithm: **GENERATE**

```
Input: n,k
Output: All possible Generator matrix for n,k
  for i belongs to 0,k-1 do
    s arrow generateZeroString(k)
    si arrow 1
    row arrow append(s)
  end for
  perms arrow generatebinaryString(n,n-k)
  permute(n,k,matrix,perms,row,0)
return mat
```

Algorithm: **PERMUTE**

```
Input: n,k,matrix (passed as refernce), perms (passed as
   refernce), row (passed as refernce), row_no
Output: Void function that generate permutation of matrix.
if row_no equals k then
```

```
      mat.push_back(I);
      return
 end if

   for j belongs to 0,  , perms.size do
      Irno = append(permsj)
      general(n,k,mat,perms,I,row_no+1)
      Erase(I row_no, n-k)
   end for
return
```

Table 2: Example: Generator matrix for (4,2)

n	k	d	Generator matrix	Codes
1000	1000	1001	....	1001
0100	0101	0100	....	0111

Total number of generator matrix possible are 16

3.1 Hamming distance

The Hamming distance between two binary strings of equal length is the number of bit places where the two bits vary.

d(a,b) represents the Hamming distance between two strings, a and b.

When data is transferred across computer networks, it is utilized for mistake detection and repair. It's also used in coding theory to compare data words of comparable length.

We conduct their XOR operation, $(\mathbf{a} \oplus \mathbf{b})$, and then count the total number of 1s in the generated string to determine the Hamming distance between two strings, and.

Minimum Hamming Distance The least Hamming distance between all feasible pairs of strings in a collection of equal length strings is the minimal Hamming distance.

Let's say you have four strings: 010, 011, 101, and 111.

$010 \oplus 011 = 001, d(010, 011) = 1$

$010 \oplus 101 = 111, d(010, 101) = 3$
$010 \oplus 111 = 101, d(010, 111) = 2$
$011 \oplus 101 = 110, d(011, 101) = 2$
$011 \oplus 111 = 100, d(011, 111) = 1$
$101 \oplus 111 = 010, d(011, 111) = 1$

As a result, $d_{\min} = 1$ is the Minimum Hamming Distance.

Algorithm: **minHammingDistance**

```
Input: string codeword
Output: integer
dist belong to INT_MAX
   for i belong to 0,  , codewords.size do
      for j belong to 0,  , codewords.size do
         dist = min(dist, countDiffernce(codewordsi, codewordsj))
          end for
    end for
return dist
```

3.2 Check single Deletion

Extracting the encoded string from the rows of G and then checking single handling Single bit error detection algorithm is given by:

Algorithm: **checkSingleDeletion**

```
Input: row, length
Output: true or false
   for i belongs to 0, , rowsize do
      for j belongs to 0, , n do
         temp = si
         erase tempj
         Dseti insert(temp)
                    if temp is not in stringmap then
                         stringmaptemp=i
                    end if
                  else then
                         if stringmaptemp not equals to I then
```

```
                        return false
                    end if
                end else
            end for
    end for
return true
```

Table 3: Codes detecting single bit error

n	k	d	Generator matrix	Codes
4	2	2	1001	0000
			0110	1001
				1111
				0110
5	2	2	10110	00000
			01001	10110
				11111
				01001
5	2	3	10011	00000
			01101	10011
				11110
				01101
6	2	2	101100	000000
			010010	101100
				111110
				010010

3.3 Time Complexity

Time complexity: $O\left(2^{(n-k)} + k^2 n\right)$
Space complexity: $O\left(nk^{*}2^{(n-k)}\right)$
Here n is size/ length of binary codewords and k is size of identity matrix used for generator matrix .

4 Conclusion

We tried to verify the implementation of VT codes in this paper and came to the conclusion that VT codes are only applicable to binary codes, and that non-binary codes require additional modifications so that codewords generated when passed through dirty channels can handle up to single bit deletion. Furthermore, we attempted to create an algorithm for classifying binary codes and determining whether or not they can withstand single-bit deletion. The approach used above is not totally efficient or ideal, and it will remain an "open problem" for further optimization. The classification problems time complexity is exponential which can further be reduced to polynomial using dynamic programming. We shall strive to improve the above-implemented binary code classification method in the future so that it can generate results for varied values of n (total length of the binary codeword) and k.

5 Learnings

We were able to investigate the coding theory field and attempt to implement some of its useful notions through the creation of algorithms, as well as validate the VT code Theory and its application through functional code. We learned how to classify binary code in order to build a set of codewords that can handle single-bit deletion when delivered across a dirty channel. We also came across some of the beautiful papers which helped us to gain knowledge about coding theory and its importance in the world of cryptography.

6 Contributions

Name	Contributions
Nilay Patel	Developed algorithm and code for VT code and Classification Code Theory.
Shubham Kevadiya	Developed algorithm and code for VT code and Classification Code Theory.
Aditya Srivastava	Helped in developing code for Classification Code Theory and documentation.
Venkatesh Banoth	Developed Latex Code and report.
Chetan Maloth	Developed Latex Code and report.

7 References

[1] N.J.A.Sloane , on Single-Deletion-Correcting Codes
[2] https://ieeexplore.ieee.org/abstract/document/8573841
[3] https://www.sciencedirect.com/science/article/pii/S0012365X13004524
[4] Oxford Applied Mathematics and Computing Science Series- Raymond Hill