

Math 3500-101
Instructor: Dr. V. Rykov

End of Class Report

Group Testing and Superimposed Codes

Final Report

Evaluating Covered words of Generated Reed-Solomon Codes & Permutations of Randomly Chosen Codeword.

Reginald Dawson Jr.

University of Nebraska at Omaha

rdawson@mail.unomaha.edu,

June 29, 20007

Preface

Group testing has been around for over fifty years. Its premise starts in testing blood for pathogens and abnormalities. Rather than testing individuals, economics motivations suggested testing large collective groups of blood simultaneously. Given its successes in optimizing the number of tests required, group testing methods have been applied to multiple areas of engineering, physical sciences, and social sciences. Group theory requires an understanding of both combinatorial and probabilistic methods and applications.

The basis of group testing is to batch together multiple units in a single test, resulting in a minimization on the number of tests required to find all significant units. Obviously, testing everything individually is often inefficient and costly. As such, group theory offers an optimized and less expensive testing method. There are both adaptive algorithms, which alter the units involved in subsequent tests based on the results of previous tests, and static, non-adaptive algorithms. Non-adaptive group testing lends itself to super-imposed coding theory.

1 Introduction

Codes are special mathematical constructions consisting of a set of unique subsequences. Each subsequence may be represented by an n -dimensional vector, often called a word. An important property of these subsequences is that they must be a minimum distance from one another. While there are many different methods for defining distance, all distance functions must hold the following properties:

- (i) $d(x, y) = 0 \iff x = y$;
- (ii) $d(x, y) = d(y, x)$ for all x, y
- (iii) $d(x, y) \leq d(x, z) + d(z, y)$ for all $x, y, \& z$.

Often this distance is represented by the Boolean sum of two words. In such cases, the sum of any two words is unique! Depending on the code, sometimes the Boolean sum of p words is unique, a property that lends itself to many networking situations.

Hence, modeling computer networks as Codes is invaluable to networking optimization. These models are known as Superimposed Codes. In [6], the Aloha network is thoroughly explored using Superimposed Coding Theory.

The Aloha Network consists of t computers that simultaneously communicate with a central station. The central station must be able to distinguish what information is being requested by the network computers. Rather than interpreting requests as a serial queue, all requests are combined into a single message of constant length N . By limiting requests to the codewords of a Superimposed Code, a single message received at the Central Station uniquely contains all the requests of the Network computers (i.e. Since only codewords are used, Central Station can uniquely reconstruct the original set of messages). The question then becomes, which Code gives us an optimum network performance? There are countless methods mathematicians use to construct codes. We'll explore one of the simplest methods, utilizing Latin Squares, before proceeding onto an optimized Reed-Solomon Code.

2 Superimposed Coding

2.1 Latin Squares Method

From wikipedia.com the definition of a Latin square "is an $n \times n$ table filled with n different symbols in such a way that each symbol occurs exactly once in each row and exactly once in each column". A Latin square is generated from the polynomial $y = ax + b$. Where for each x (1.. n) there is a different table generated with a and b representing the rows and columns of the table from (0.. $n-1$). The following table is a example of a Latin square with $n = 5$, $x = 2$

$y=a(2)+b$	$b=0$	1	2	3	4
$a=0$	0	1	2	3	4
1	2	3	4	0	1
2	4	0	1	2	3
3	1	2	3	4	0
4	3	4	0	1	2

From this table a 25-word code can be generated by creating (in the table below) each word from a Row, a Column, and the associated Row-Column value.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	0	0	0	0	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3	4	4	4	4	4
0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
0	1	2	3	4	2	3	4	0	1	4	0	1	2	3	1	2	3	4	0	3	4	0	1	2

While looking at this code, it is easy to verify all codewords are unique. However, to further explore the properties of this code, we will recreate the code in a binary format. The reason for this is two-fold. First, computers use a binary system, and second, we shall define our distance as the exclusive OR function. To recreate the code in binary, we represent each 0, 1, 2, 3, and 4 as 00001, 00010, 00100, 01000, & 10000 respectively. The binary representation of the above code is displayed on the following page.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1
0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0
0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0

Using this Superimposed code, codewords of length $N = 15$ could represent $t = 25$ different parameters. To restate this in terms of group testing, each row represents a test r , each column represents a person c , and a 1 implies person c was involved in test r . As long as the number of infected people is less than 3, we can identify which of the 25 people are infected, using only fifteen tests. With this in mind, realize the value of a code is determined by more than just N & t , but also why the number of infected people must be ≤ 2 .

When comparing codes, we care about the amount maximum amount of information that can be transmitted at one time. Given the Boolean sum of any two of the codewords above, we can determine exactly which two unique codewords were used to create the given sum. This property, called the Strength S of the code, is an extremely important property while comparing codes. A code X has a strength S if and only if the Boolean sum of any s codewords does not cover any other codeword in X (not in the s -set) [6]. The strength of a code has an upper bound given by:

$$S \leq \left\lfloor \frac{w-1}{\lambda_{\max}} \right\rfloor$$

Where w refers to the weight of a codeword, and λ_{\max} refers to the maximum intersection of a codeword, and equality holds for perfect codes only. The weight of a codeword refers to how many nonzero digits it contains. A quick glance will show every codeword above has a weight of 3. The intersection is the number of bits two codewords have in common. From our construction above, the maximum intersection between any two codewords is 1. Using these two parameters the Strength of this code is ≤ 2 . To show this, notice that the Boolean sum of 3 codewords (like, words 1, 7, & 13) sometimes cover an extra word (they covered words 2 & 12). Since this Code only has a strength of 2, it would make a poor choice for the Aloha System. The Aloha System needs a code construction where more than 2 computers can speak at a time!

3 Our Contribution

Our contribution involved constructing a computer program to generate a Reed-Solomon code when given the parameters q & k . Then, given a number of p codewords, to determine (i) how many codewords are covered on average by p codewords, (ii) what the minimum number of codewords covered by p codewords, and (iii) what the maximum number of codewords covered by p codewords. Due to the factorial growth of combinations to test, this task proved quite lengthy.

The first step for our computer program was to generate a Reed-Solomon Code given the parameters q & k . This is done by setting up an array of functions, with each function of the form $f(x) = A_0 + A_1x + \dots + A_{k-1}x^{k-1}$. By appropriately incrementing our coefficients, we end up with a complete list of every possible function. The list of functions is represented by an array of its coefficients, which is printed to the file `c:\temp\FunctQ##K##P##.txt`. The `##` is replaced by the appropriate q -value, k -value, and p -value. Printing out the file provides a method for verifying all possible functions are accounted for. A sample of the file for $q = 9$ & $k = 4$ is printed below. Only 98 of the $8^4 = 4096$ functions are shown here.

Function 0:	0	0	0	0
Function 1:	0	0	0	1
Function 2:	0	0	0	2
Function 3:	0	0	0	3
Function 4:	0	0	0	4
Function 5:	0	0	0	5
Function 6:	0	0	0	6
Function 7:	0	0	0	7
Function 8:	0	0	0	8
Function 9:	0	0	1	0
Function 10:	0	0	1	1
Function 11:	0	0	1	2
Function 12:	0	0	1	3
Function 13:	0	0	1	4
Function 14:	0	0	1	5
Function 15:	0	0	1	6
Function 16:	0	0	1	7
Function 17:	0	0	1	8
Function 18:	0	0	2	0
Function 19:	0	0	2	1
Function 20:	0	0	2	2
Function 21:	0	0	2	3
Function 22:	0	0	2	4
Function 23:	0	0	2	5
Function 24:	0	0	2	6
Function 25:	0	0	2	7
Function 26:	0	0	2	8
Function 27:	0	0	3	0
Function 35:	0	0	3	8
Function 36:	0	0	4	0
Function 37:	0	0	4	1
Function 38:	0	0	4	2
Function 39:	0	0	4	3
Function 40:	0	0	4	4
Function 41:	0	0	4	5
Function 42:	0	0	4	6
Function 43:	0	0	4	7
Function 44:	0	0	4	8
Function 45:	0	0	5	0
Function 46:	0	0	5	1
Function 47:	0	0	5	2
Function 48:	0	0	5	3
Function 49:	0	0	5	4
Function 50:	0	0	5	5
Function 51:	0	0	5	6
Function 52:	0	0	5	7
Function 53:	0	0	5	8
Function 54:	0	0	6	0
Function 55:	0	0	6	1
Function 56:	0	0	6	2
Function 57:	0	0	6	3
Function 58:	0	0	6	4
Function 59:	0	0	6	5
Function 60:	0	0	6	6
Function 61:	0	0	6	7
Function 62:	0	0	6	8
Function 70:	0	0	7	7
Function 71:	0	0	7	8
Function 72:	0	0	8	0
Function 73:	0	0	8	1
Function 74:	0	0	8	2
Function 75:	0	0	8	3
Function 76:	0	0	8	4
Function 77:	0	0	8	5
Function 78:	0	0	8	6
Function 79:	0	0	8	7
Function 80:	0	0	8	8
Function 81:	0	1	0	0
Function 82:	0	1	0	1
Function 83:	0	1	0	2
Function 84:	0	1	0	3
Function 85:	0	1	0	4
Function 86:	0	1	0	5
Function 87:	0	1	0	6
Function 88:	0	1	0	7
Function 89:	0	1	0	8
Function 90:	0	1	1	0
Function 91:	0	1	1	1
Function 92:	0	1	1	2
Function 93:	0	1	1	3
Function 94:	0	1	1	4
Function 95:	0	1	1	5
Function 96:	0	1	1	6
Function 97:	0	1	1	7

Once we created the list of all possible functions, we generate our code utilizing these functions. Each Codeword $x = \{ f_x(0), f_x(1), f_x(2), \dots, f_x(q-1), f_x(\infty) \}$. Just to clarify, we use the value ∞ to represent, $f_x(\infty) = A_{k-1}$. Again, to verify our codes, the program prints the entire code to `"c:\temp\QnaryCodeQ09K04P03.txt"`. Note, the file names change to represent the values of q , k , & p . A sample of the file is shown below, where each function number represents one of the 4096 codewords

Function 0:	0	0	0	0	0	0	0	0	0	0
Function 1:	0	1	8	0	1	8	0	1	8	1
Function 2:	0	2	7	0	2	7	0	2	7	2
Function 3:	0	3	6	0	3	6	0	3	6	3
Function 4:	0	4	5	0	4	5	0	4	5	4
Function 5:	0	5	4	0	5	4	0	5	4	5
Function 6:	0	6	3	0	6	3	0	6	3	6
Function 7:	0	7	2	0	7	2	0	7	2	7
Function 8:	0	8	1	0	8	1	0	8	1	8
Function 9:	0	1	4	0	7	7	0	4	1	0
Function 10:	0	2	3	0	8	6	0	5	0	1
Function 11:	0	3	2	0	0	5	0	6	8	2
Function 12:	0	4	1	0	1	4	0	7	7	3
Function 13:	0	5	0	0	2	3	0	8	6	4
Function 14:	0	6	8	0	3	2	0	0	5	5
Function 15:	0	7	7	0	4	1	0	1	4	6
Function 16:	0	8	6	0	5	0	0	2	3	7
Function 17:	0	0	5	0	6	8	0	3	2	8
Function 18:	0	2	8	0	5	5	0	8	2	0

After we generate the code in q-nary format, we also recreate it in a binary format. This is printed to the file "e:\temp\BnaryCodeQ07K04S03.txt". We convert the code to a binary format to enable Boolean sums. Once in this form, we are ready to test how many words are covered by a given number of words. We test our program first with a randomly selected group of s words. We then compare the Boolean sum of these s words with every single word in the code. The tells us how many words they cover. Again, to verify it, we print the results to a file. A sample output is listed below:

```

For Function 84: [0] M [1] M [2]
For Function 85: [0] M [1]
For Function 86: [0] M [1]
For Function 87: [0] M [1] M [2] M [3] M [4]
For Function 88: [0] M [1] M [2]
For Function 89: [0] M [1]
For Function 90: [0] M [1]
For Function 91: [0] M [1]
For Function 92: [0] M [1] M [2]
For Function 93: [0] M [1]
For Function 94: [0] M [1]
For Function 95: [0] M [1] M [2]
For Function 96: [0] M [1] M [2]
For Function 97: [0] M [1]
For Function 98: [0] M [1]
For Function 99: [0] M [1]
For Function 100: [0] M [1] M [2]
For Function 101: [0] M [1]

```

Once we've thoroughly tested our code, we systematically select groups of S codewords, sum them together and see how many words they cover. For a given code, there are q^k choose s possible combinations we must test. For modest values, like $q = 9$, $k = 4$, & $s = 3$, we are looking at 6561 choose 3 = 47,050,068,420 combinations to test. My 3 GHz AMD 64 has been running for a full week and has yet to complete the above calculations. Given the enormous amount of calculations, we require multiple computers and an extended period of time to test a large variety of q, k, & s. Our results are printed to the file "e:\temp\BnaryCodeQ07K04S03.txt".

4 Conclusion

The results of our trials have been printed to the Excel file Results. We had trouble comparing our calculated results to that of theory. The reason for this, is the theoretical values relating the average number of words covered are is a function dependent upon a shortened Reed-Solomon Code. Since our code is not shortened, and the function depends on a minimum shortening value, the comparison seemed flawed. Further investigation into the relationship between the shortened & unshortened codes is necessary.

REFERENCES

- [1] D'yachkov, Arkadii G. Lectures on Designing Screening Experiments. Department of Probability Theory, Moscow State University. 1-64.
- [2] D'yachkov, A, V Rykov, D Torney, and S Yekhanin. Partition Codes. 1-12.
- [3] Du, Ding-Zhu, and Frank K. Hwang. Combinatorial Group Testing and Its Applications. Vol. 3. River Edge, New Jersey: World Scientific, 1993. 1-249.
- [4] Gyori, Sandor. Signature Coding for OR Channel with Asynchronous Access. Department of Computer Science and Information Theory, Budapest University of Technology and Economics. 2005.
- [5] Kautz, W H., and R C. Singleton. Nonrandom Binary Superimposed Codes. IEEE. 1964. 363-377.
- [6] Rykov, Vyacheslav V., and Vladimir V. Ufimtsev. Group Testing and Its Application to Multiple Access Information Transmission Models. University of Nebraska At Omaha. 2006. 1-15.
- [7] "Reed-Solomon Error Correction." Wikipedia. Wikimedia Foundation, 2007. Omaha. 28 June 2007 <<http://en.wikipedia.org/wiki/Reed-Solomon>>.
- [8] Rykov, Vyacheslav V., and Vladimir V. Ufimtsev. Group Testing and Its Application to Multiple Access Information Transmission Models. University of Nebraska At Omaha. 2006. 1-15.