Foreword

This chapter is based on lecture notes from coding theory courses taught by Venkatesan Guruswami at University at Washington and CMU; by Atri Rudra at University at Buffalo, SUNY and by Madhu Sudan at MIT.

This version is dated **August 15, 2014**. For the latest version, please go to

http://www.cse.buffalo.edu/ atri/courses/coding-theory/book/

The material in this chapter is supported in part by the National Science Foundation under CAREER grant CCF-0844796. Any opinions, findings and conclusions or recomendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

Chapter 20

Finding Defectives: Group Testing

Consider the following situation that arises very frequently in *biological screens*. Say there are N individuals and the objective of the study is to identify the individuals with a certain “biomarker" that could e.g. indicate that the individual has some specific disease or is at risk for a certain health condition. The naive way to do this would be to test each person individually, that is:

1. Draw sample (e.g. a blood or DNA sample) from a given individual,

2. Perform required tests, then

3. Determine presence or absence of the biomarker.

This method of one test per person will gives us a total of N tests for a total of N individuals. Say we had more than $70-75\%$ of the population infected. At such large numbers, the use of the method of individual testing is reasonable. However, our goal is to achieve effective testing in the more likely scenario where it doesn't make sense to test $100,000$ people to get just (say) 10 positives.

The feasibility of a more effective testing scheme hinges on the following property. We can combine blood samples and test a combined sample together to check if at least one individual has the biomarker.

The main question in group testing is the following: If we have a very large number of items to test, and we know that only certain few will turn out positive, what is a nice and efficient way of performing the tests?

20.1 Formalization of the problem

We now formalize the group testing problem. The input to the problem consists of the following:

- The total number of individuals: N.

- An upper bound on the number of infected individuals d.

- The input can be described as a vector $\mathbf{x} = (x_1, x_2, ..., x_n)$ where $x_i = 1$ if individual i has the biomarker, else $x_i = 0$.

Note that $wt(\mathbf{x}) \leq d$. More importantly, notice that the vector $\mathbf{x}$ is an implicit input since we do not know the positions of 1s in the input. The only way to find out is to run the *test*s. Now, we will formalize the notion of a test.

A query/test S is a subset of $[N]$. The answer to the query $S \subseteq [N]$ is defined as follows:

$$A(S) = \begin{cases} 1 & \text{if } \sum_{i \in S} x_i \geq 1; \\ 0 & \text{otherwise.} \end{cases}$$

Note that the answer to the S is the logical-OR of all bits in S, i.e. $A(S) = \bigvee_{i \in S} x_i$.

The goal of the problem is to compute $\mathbf{x}$ and minimize the number of tests required to determine $\mathbf{x}$.

Testing methods. There is another aspect of the problem that we need specify. In particular, we might need to restrict how the tests interact with each other. Below are two commons ways to carry out tests:

1. *Adaptive group testing* is where we test a given subset of items, get the answer and base our further tests on the outcome of the previous set of tests and their answers.

2. *Non-Adaptive group testing* on the other hand is when all our tests are set even before we perform our first test. That is, all our tests are decided a priori.

Non-adaptive group testing is crucial for many applications. This is because the individuals could be geographically spread pretty far out. Due to this, adaptive group testing will require a very high degree of co-ordination between the different groups. This might actually increase the cost of the process.

Notation. We will also need notation to denote the minimum number of tests needed in group testing. Towards this end, we present the following two definitions.

Definition 20.1.1 ($t(d,N)$)**.** Given a subset of N items with d defects represented as $\mathbf{x} \in \{0,1\}^N$, the minimum number of *non-adaptive* tests that one would have to make is defined as $t(d,N)$.

Definition 20.1.2. $t^a(d,N)$: Given a set of N items with d defects, $t^a(d,N)$ is defined as the number of *adaptive* tests that one would have to make to detect all the defective items.

The obvious questions are to prove bounds on $t(d,N)$ and $t^a(d,N)$:

Question 20.1.1. *Prove asymptotically tight bounds on $t(d,N)$.*

Question 20.1.2. *Prove asymptotically tight bounds on $t^a(d,N)$.*

We begin with some simple bounds:

Proposition 20.1.1. *For every $1 \le d \le N$, we have*

$$1 \le t^a(d,N) \le t(d,N) \le N.$$

Proof. The last inequality follows from the naive scheme of testing all individuals with singleton tests while the first inequality is trivial. The reason for $t^a(d,N) \le t(d,N)$ is due to the fact that any non-adaptive test can be performed by an adaptive test by running all of the tests in the first step of the adaptive test. Adaptive tests can be faster than non-adaptive tests since the test can be changed after certain things are discovered. □

Representing the set of tests as a matrix. It turns out that is is useful to represent a non-adaptive group testing scheme as a matrix. Next, we outline the natural such representation. For, $S \subseteq [N]$, define $\chi_S \in \{0,1\}^N$ such that $i \in S$ if and only if $\chi_S(i) = 1$. Consider a non-adaptive group testing scheme with t test $S_1, \dots, S_t$. The corresponding matrix M is a $t \times N$ matrix where the ith row is χ_{S_i}. (Note that the trivial solution of testing each individual as a singleton set would just be the $N \times N$ identity matrix.) In other words, $M = \{M_{ij}\}_{i \in [t], j \in [N]}$ such that $M_{ij} = 1$ if $j \in S_i$ and $M_{ij} = 0$ otherwise.

If we assume that multiplication is logical AND ($\bigwedge$) and addition is logical OR ($\bigvee$), then we have $M \times \mathbf{x} = \mathbf{r}$ where $\mathbf{r} \in \{0,1\}^t$ is the vector of the answers to the t tests. We will denote this operation as $M \odot \mathbf{x}$. To think of this in terms of testing, it is helpful to visualize the matrix-vector multiplication. Here, $\mathbf{r}$ will have a 1 in position i if and only if there was a 1 in that position in both M and $\mathbf{x}$ i.e. if that person was tested with that particular group and if he tested out to be positive.

Thus, our goal is to get to compute $\mathbf{x}$ from $M \odot \mathbf{x}$ with as small a t as possible.

20.2 Bounds on $t^a(d,N)$

In this section, we will explore lower and upper bounds on $t^a(d,N)$ with the ultimate objective of answering Question 20.1.2.

We begin with a lower bound that follows from a simple counting argument.

Proposition 20.2.1. *For every $1 \le d \le N$,*

$$t^a(d,N) \ge d \log \frac{N}{d}.$$

Proof. Fix any valid adaptive group testing scheme with t tests. Observe that if $\mathbf{x} \ne \mathbf{y} \in \{0,1\}^N$, with $wt(\mathbf{x}), wt(\mathbf{y}) \le d$ then $\mathbf{r}(\mathbf{x}) \ne \mathbf{r}(\mathbf{y})$, where $\mathbf{r}(\mathbf{x})$ denotes the result vector for running the tests on $\mathbf{x}$ and similarly for $\mathbf{r}(\mathbf{y})$. The reason for this is because two valid inputs cannot give the same result. If this were the case and the results of the tests gave $\mathbf{r}(\mathbf{x}) = \mathbf{r}(\mathbf{y})$ then it would not be possible to distinguish between $\mathbf{x}$ and $\mathbf{y}$.

The above observation implies that total number of distinct test results is the number distinct binary vectors with Hamming weight at most d, i.e. $Vol_2(d,N)$. On the other hand, the number of possible distinct t-bit vectors is at most 2^t, which with the previous argument implies that

$$2^t \geq Vol_2(d,N)$$

and hence, it implies that

$$t \geq \log Vol_2(d,N).$$

Recall that

$$Vol_2(d,N) \geq \binom{N}{d} \geq \left(\frac{N}{d}\right)^d,$$

where the first inequality follows from (3.23) and the second inequality follows from Lemma B.1.1. So $t \geq d\log(N/d)$, as desired. $\square$

It turns out that $t^a(d,N)$ is also $O\left(d\log\left(\frac{N}{d}\right)\right)$. (See Exercise 20.1.) This along with Proposition 20.2.1 implies that $t^a(d,N) = \Theta\left(d\log\left(\frac{N}{d}\right)\right)$, which answers Question 20.1.2. The upper bound on $t^a(d,N)$ follows from an adaptive group testing scheme and hence does not say anything meaningful for Question 20.1.1. (Indeed, we will see later that $t(d,N)$ cannot be $O(d\log(N/d))$.) Next, we switch gears to talk about non-adaptive group testing.

20.3 Bounds on $t(d,N)$

We begin with the simplest case of $d = 1$. In this case it is instructive to recall our goal. We want to define a matrix M such that given any $\mathbf{x}$ with $wt(\mathbf{x}) \leq 1$, we should be able to compute $\mathbf{x}$ from $M \odot \mathbf{x}$. In particular, let us consider the case when $\mathbf{x} = \mathbf{e}_i$ for some $i \in [N]$. Note that in this case $M \odot \mathbf{x} = M^i$, where M^i is the ith column of M. Hence we should design M such that M^i uniquely defined i. We have already encountered a similar situation before in Section 2.6 when trying to decode the Hamming code. It turns out that is suffices to pick M as the parity check matrix of a Hamming code. In particular, we can prove the following result:

Proposition 20.3.1. $t(1,N) \leq \lceil \log(N+1) \rceil + 1$

Proof. We prove the upper bound by exhibiting a matrix that can handle non adaptive group testing for $d = 1$. The group test matrix M is the parity check matrix for $[2^m-1, 2^m-m-1, 3]_2$, i.e. H_m where the i-th column is the binary representation of i (recall Section 2.4). This works because when performing $H_m \odot \mathbf{x} = \mathbf{r}$, if $wt(\mathbf{v}) \leq 1$ then $\mathbf{r}$ will correspond to the binary representation of i. Further, note that if $wt(\mathbf{x}) = 0$, then $\mathbf{r} = \mathbf{0}$, which is exactly $\mathbf{x}$. Hence, $M \odot \mathbf{x}$ uniquely identifies $\mathbf{x}$ when $wt(\mathbf{x}) \leq 1$, as desired.

If $N \neq 2^m - 1$ for any m, the matrix H_m corresponding to the m such that $2^{m-1} - 1 < N < 2^m - 1$ can be used by adding 0s to the end of $\mathbf{x}$. By doing this, decoding is "trivial" for both cases since the binary representation is given for the location. So the number of tests is at most $\lceil \log(N+1) \rceil + 1$, which completes the proof. $\square$

Note that Propositions 20.1.1 and 20.2.1 imply that $t(d,N) \ge \log N$, which with the above result implies that $t(1,N) = \Theta(\log N)$. This answers Question 20.1.1 for the case of $d = 1$. We will see later that such a tight answer is not known for larger d. However, at the very least we should try and extend Proposition 20.3.1 for larger values of d. In particular,

Question 20.3.1. *Prove asymptotic upper bounds on $t(d,N)$ that hold for every $1 < d \le N$.*

We would like to point out something that was implicitly used in the proof of Proposition 20.3.1. In particular, we used the implicitly understanding that a non-adaptive group testing matrix M should have the property that given any $\mathbf{x} \in \{0,1\}^N$ such that $wt(\mathbf{x}) \le d$, the result vector $M \odot \mathbf{x}$ should uniquely determine $\mathbf{x}$. This notion is formally captured in the following definition of non-adaptive group testing matrix:

Definition 20.3.1. A $t \times N$ matrix M is *d-separable* if and only if for every $S_1 \ne S_2 \subseteq [N]$ such that $|S_1|, |S_2| \le d$, we have

$$\bigcup_{j \in S_1} M^j \ne \bigcup_{i \in S_2} M^i.$$

In the above we think of a columns $M^i \in \{0,1\}^t$ as a subset of $[t]$ and for the rest of this chapter we will use both views of the columns of a group testing matrix. Finally, note that the above definition is indeed equivalent to our earlier informal definition since for any $\mathbf{x} \in \{0,1\}^N$ with $wt(\mathbf{x}) \le d$, the vector $M \odot \mathbf{x}$ when thought of as its equivalent subset of $[t]$ is exactly the set $\cup_{i \in S} M^i$, where S is the support of $\mathbf{x}$, i.e. $S = \{i | x_i = 1\}$.

Like in the coding setting, where we cared about the run time of the decoding algorithm, we also care about the time complexity of the decoding problem (given $M \odot \mathbf{x}$ compute $\mathbf{x}$) for group testing. We will now look at the obvious decoding algorithm for d-separable matrices: just check all possible sets that could form the support of $\mathbf{x}$. Algorithm 28 has the details.

The correctness of Algorithm 28 follows from Definition 20.3.1. Further, it is easy to check that this algorithm will run in $N^{\Theta(d)}$ time, which is not efficient for even moderately large d. This naturally leads to the following question:

Question 20.3.2. *Do there exists d-separable matrices that can be efficient decoded?*

We would like to remark here that the matrices that we seek in the answer to Question 20.3.2 should have small number of tests (as otherwise the identity matrix answers the question in the affirmative).

Algorithm 28 Decoder for Separable Matrices

```
INPUT: Result vector $\mathbf{r}$ and $d$-separable matrix $M$
OUTPUT: $\mathbf{x}$ if $\mathbf{r} = M \odot \mathbf{x}$ else Fail

1: $R \leftarrow \{i | r_i = 1\}$.
2: FOR Every $T \subseteq [N]$ such that $|T| \le d$ DO
3:     $S_T \leftarrow \cup_{i \in T} M^i$
4:     IF $R = S_T$ THEN
5:         $\mathbf{x} \leftarrow (x_1, \ldots, x_N) \in \{0,1\}^N$ such that $x_i = 1$ if and only $i \in T$.
6:         RETURN $\mathbf{x}$
7: RETURN Fail
```

20.3.1 Disjoint Matrices

We now define a stronger notion of group testing matrices that have a more efficient decoding algorithm than d-separable matrices.

Definition 20.3.2. A $t \times N$ matrix M is d-disjunct if and only if for every $S \subset [N]$ with $|S| \le d$ and for every $j \notin S$, there exist an $i \in [t]$ such that $M_{ij} = 1$ but for all $k \in S$, $M_{ik} = 0$. Or equivalently

$$M^j \not\subseteq \bigcup_{k \in S} M^k.$$

For an illustration of the definition, see Figure 20.3.2.

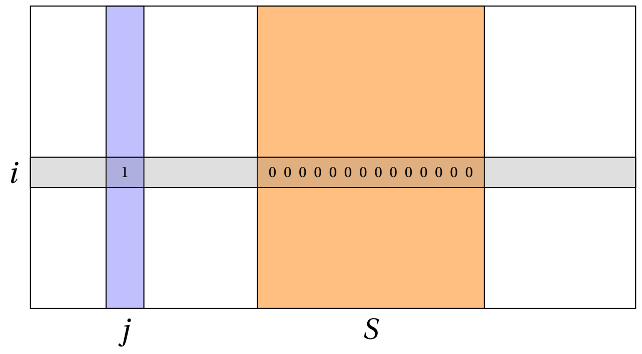

Figure 20.1: Pick a subset S (not necessarily contiguous). Then pick a column j that is not present in S. There will always be i such that row i has a 1 in column j and all zeros in S.

Next we argue that disjunctness is a sufficient condition for separability.

Lemma 20.3.2. *Any d-disjunct matrix is also d-separable.*

Proof. For contradiction, assume M is d-disjunct but not d-separable. Since M is not d-separable, then union of two subset $S \neq T \subset [N]$ of size at most d each are the same; i.e.

$$\bigcup_{k\in S} M^k = \bigcup_{k\in T} M^k.$$

Since $S \neq T$, there exists $j \in T \setminus S$. But we have

$$M^j \subseteq \bigcup_{k\in T} M^k = \bigcup_{k\in S} M^k,$$

where the last equality follows from the previous equality. However, since by definition $j \notin S$, the above contradicts the fact that M is d-disjunct. □

In fact, it turns out that disjunctness is also almost a necessary condition: see Exercise 20.2.

Next, we show the real gain of moving to the notion of disjunctness from the notion of separability.

Lemma 20.3.3. *There exists a $O(tN)$ time decoding algorithm for any $t \times N$ matrix that is d-disjunct.*

Proof. The proof follows from the following two observations.

First, say we have a matrix M and a vector $\mathbf{x}$ and $\mathbf{r} = M \odot \mathbf{x}$ such that $r_i = 1$. Then there exists a column j in matrix that made it possible i.e. if $r_i = 1$, then there exists a j such that $M_{ij} = 1$ and $x_j = 1$.

Second, let T be a subset and j be a column not in T where $T = \{\ell \mid x_\ell = 1\}$ and $|T| \le d$. Consider the ith row such that T has all zeros in the ith row, then $r_i = 0$. Conversely, if $r_i = 0$, then for every $j \in [N]$ such that $M_{ij} = 1$, it has to be the case that $x_j = 0$. This naturally leads to the decoding algorithm in Algorithm 29.

The correctness of Algorithm 29 follows from the above observation and it can be checked that the algorithm runs in time $O(tN)$– see Exercise 20.3. □

Modulo the task of exhibiting the existence of d-disjunct matrices, Lemmas 20.3.3 and 20.3.2 answer Question 20.3.2 in the affirmative. Next, we will tackle the following question:

Question 20.3.3. *Design d-disjunct matrices with few rows.*

As we will see shortly answering the above question will make connection to coding theory becomes even more explicit.

20.4 Coding Theory and Disjunct Matrices

In this section, we present the connection between coding theory and disjunct matrices with the final goal of answering Question 20.3.3. First, we present a sufficient condition for a matrix to be d-disjunct.

Algorithm 29 Naive Decoder for Disjunct Matrices

INPUT: Result vector $\mathbf{r}$ and d-disjunct matrix M
OUTPUT: $\mathbf{x}$ if $M \odot \mathbf{x} = \mathbf{r}$ else `Fail`

```
1: Initialize $\mathbf{x} \in \{0,1\}^N$ to be the all ones vector
2: FOR every $i \in [t]$ DO
3:     IF $r_i = 0$ THEN
4:         FOR Every $j \in [N]$ DO
5:             IF $M_{ij} = 1$ THEN
6:                 $x_j \leftarrow 0$
7: IF $M \odot \mathbf{x} = \mathbf{r}$ THEN
8:     RETURN $\mathbf{x}$
9: ELSE
10:    RETURN Fail
```

Lemma 20.4.1. *Let $1 \le d \le N$ be an integer and M be a $t \times N$ matrix, such that*

(i) For every $j \in [N]$, the jth column has at least $w_{\min}$ ones in it, i.e. $\left|M^j\right| \ge w_{\min}$ and

(ii) For every $i \ne j \in [N]$, the i and j'th columns have at most $a_{\max}$ ones in common, i.e. $\left|M^i \cap M^j\right| \le a_{\max}$

for some integers $a_{\max} \le w_{\min} \le t$. Then M is a $\left\lfloor \frac{w_{\min}-1}{a_{\max}} \right\rfloor$-disjunct.

Proof. For notational convenience, define $d = \left\lfloor \frac{w_{\min}-1}{a_{\max}} \right\rfloor$. Fix an arbitrary $S \subset [N]$ such that $|S| \le d$ and a $j \notin S$. Note we have to show that

$$M^j \not\subseteq \cup_{i \in S} M^i,$$

or equivalently

$$M^j \not\subseteq \cup_{i \in S} \left(M^i \cap M^j\right).$$

We will prove the above by showing that

$$\left|M^j \setminus \cup_{i \in S}\left(M^i \cap M^j\right)\right| > 0.$$

Indeed, consider the following sequence of relationships:

$$\begin{aligned}
\left|M^j \setminus \cup_{i \in S}\left(M^i \cap M^j\right)\right| &= \left|M^j\right| - \left|\cup_{i \in S}\left(M^i \cap M^j\right)\right| \\
&\ge \left|M^j\right| - \sum_{i \in S} \left|\left(M^i \cap M^j\right)\right| && (20.1) \\
&\ge w_{\min} - |S| \cdot a_{\max} && (20.2) \\
&\ge w_{\min} - d \cdot a_{\max} && (20.3) \\
&\ge w_{\min} - \frac{w_{\min}-1}{a_{\max}} \cdot a_{\max} && (20.4) \\
&= 1.
\end{aligned}$$

In the above, (20.1) follows from the fact that size of the union of sets is at most the sum of their sizes. (20.2) follows from the definitions of $w_{\min}$ and $a_{\max}$. (20.3) follows from the fact that $|S| \le d$ while (20.4) follows from the definition of d. The proof is complete. $\Box$

Next, we present a simple way to convert a code into a matrix. Let $C \subseteq [q]^t$ be a code such that $C = \{\mathbf{c}_1, \dots, \mathbf{c}_N\}$. Then consider the matrix M_C whose i'th column is $\mathbf{c}_i$, i.e.

$$M_C = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}.$$

Thus, by Lemma 20.4.1, to construct an $\left\lfloor \frac{w_{\min}-1}{a_{\max}} \right\rfloor$-disjunct matrix, it is enough to design a binary code $C^* \subseteq \{0,1\}^t$ such that (i) for every $\mathbf{c} \in C^*$, $wt(\mathbf{c}) \ge w_{\min}$ and (ii) for every $\mathbf{c}^1 \neq \mathbf{c}^2 \in C^*$, we have $|\{i | c^1_i = c^2_i = 1\}| \le a_{\max}$. Next, we look at the construction of such a code.

20.4.1 Kautz-Singleton Construction

In this section, we will prove the following result:

Theorem 20.4.2. *For every integer $d \ge 1$ and large enough $N \ge d$, there exists a $t \times N$ matrix is d-disjunct where $t = O\left(d^2 \left(\log_d N\right)^2\right)$.*

Note that the above result answers Question 20.3.3. It turns out that one can do a bit better: see Exercise 20.4.

Towards this end, we will now study a construction of C^* as in the previous section due to Kautz and Singleton. As we have already seen in Chapter 10, concatenated codes are a way to design binary codes. For our construction of C^*, we will also use code concatenation. In particular, we will pick $C^* = C_{\text{out}} \circ C_{\text{in}}$, where C_{out} is a $[q, k, q-k+1]_q$ Reed-Solomon code (see Chapter 5) while the inner code $C_{\text{in}} : \mathbb{F}_q \to \{0,1\}^q$ is defined as follows. For any $i \in \mathbb{F}_q$, define $C_{\text{in}}(i) = \mathbf{e}_i$. Note that $M_{C_{\text{in}}}$ is the identity matrix and that $N = q^k$ and $t = q^2$.

Example 20.4.3. *Let $k = 1$ and $q = 3$. Note that by our choice of $[3,1]_3$ Reed-Solomon codes, we have $C_{\text{out}} = \{(0,0,0),(1,1,1),(2,2,2)\}$. In other words,*

$$M_{C_{\text{out}}} = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix}.$$

Then the construction of M_{C^*} can be visualized as in Figure 20.4.3.

Next, we instantiate parameters in the Kautz-Singleton construction to prove Theorem 20.4.2.

Proof of Theorem 20.4.2. We first analyze the construction to determine the parameters $w_{\min}$ and $a_{\max}$. Then we pick the parameters q and k in terms of d to complete the proof.

Recall that $N = q^k$ and $t = q^2$. It is easy to check that every column of M_{C^*} has exactly q ones in it. In other words, $w_{\min} = q$. Next, we estimate $a_{\max}$.

$$\underset{M_{C_{\text{out}}}}{\begin{bmatrix}0&1&2\\0&1&2\\0&1&2\end{bmatrix}} \circ \underset{M_{C_{\text{in}}}}{\begin{bmatrix}0&0&1\\0&1&0\\1&0&0\end{bmatrix}} \rightarrow \underset{M_{C^*}}{\begin{bmatrix}0&0&1\\0&1&0\\1&0&0\\0&0&1\\0&1&0\\1&0&0\\0&0&1\\0&1&0\\1&0&0\end{bmatrix}}$$

Figure 20.2: Construction of the final matrix M_{C^*} from $M_{C_{\text{out}}}$ and $M_{C_{\text{in}}}$ from Example 20.4.3. The rows in M_{C^*} that correspond to the same row in $M_{C_{\text{out}}}$ have the same color.

Divide the rows into q sized chunks, and index the $t = q^2$ rows by pairs in $[q] \times [q]$. Recall that each column in M_{C^*} corresponds to a codeword in C^*. For notational convenience, we will use M for M_{C^*}. Note that for any row $(i, j) \in [q] \times [q]$ and a column index $\ell \in [N]$, we have $M_{(i,j),\ell} = 1$ if and only if $\mathbf{c}_\ell(j) = j$ (where we use some fixed bijection between $\mathbb{F}_q$ and $[q]$ and $\mathbf{c}_\ell$ is the ℓ'th codeword in C_{out}). In other words, the number of rows where the ℓ_1th and ℓ_2th columns both have a one is exactly the number of positions where $\mathbf{c}_{\ell_1}$ and $\mathbf{c}_{\ell_2}$ agree, which is exactly $q - \Delta(\mathbf{c}_{\ell_1}, \mathbf{c}_{\ell_2})$. Since C_{out} is a $[q, k, q-k+1]_q$ code, the number of rows where any two columns agree is at most $k-1$. In other words, $a_{\max} = k-1$.[1]

Lemma 20.4.1 implies that M_{C^*} is d-disjunct if we pick

$$d = \left\lfloor \frac{q-1}{k-1} \right\rfloor.$$

Thus, we pick q and k such that the above is satisfied. Note that we have $q = O(kd)$. Further, since we have $N = q^k$, we have

$$k = \log_q N.$$

This implies that $q = O(d \cdot \log_q N)$, or $q \log q = O(d \log N)$. In other words we have

$$q = O(d \log_d N).$$

Recalling that $t = q^2$ completes the proof. □

An inspection of the proof of Theorem 20.4.2 shows that we only used the distance of the Reed-Solomon code and in particular, any C_{out} with large enough distance suffices. In particular, if we pick C_{out} to be a random code over an appropriate sized alphabet then one can obtain $t = O(d^2 \log N)$. (See Exercise 20.5 for more on this.) Note that this bound is incomparable to the bound in Theorem 20.4.2. It turns out that these two are the best known upper bounds on $t(d, N)$. In particular,

[1]The equality is obtained due to columns that corresponds to codewords that agree in exactly $k-1$ positions.

Open Question 20.4.1. *Can we beat the upper bound of* $O\left(d^2 \cdot \min\left(\log(N/d), \log_d^2 N\right)\right)$ *on* $t(d,N)$*?*

It turns out that the quadratic dependence on d in the upper bounds is tight. In fact it is known that $t(d,N) \geq \Omega(d^2 \log_d N)$. (See Exercises 20.7 and 20.8.)

Next, we present an application of group testing in the field of data stream algorithms, which in turn will bring out another facet of the connection between coding theory and group testing.

20.5 An Application in Data Stream Algorithms

Let us consider the problem of tracking updates on stock trades. Given a set of trades $(i_1, u_1), \cdots,$ (i_m, u_m), where i_j is the stock id for the j^{th} trade, u_j is the amount of the stocks in the j^{th} trade. The problem is to keep track of the top d stocks. Such a problem is also called hot items/ heavy hitters problem.

Let N be the total number of stocks in the market. This problem could be solved in $O(m) + O(N \log N) \approx O(m)$ time and $O(N)$ space by setting a $O(N)$ size buffer to record the total number of trading for each stock and then sort the buffer later. However, m could be of the order of millions for one minute's trading, e.g. in the first minute of April 19, 2010, there are 8077600 stocks were traded. Taking the huge amount of trades into consideration, such an algorithm is not practical.

A more practical model of efficient algorithms in this context is one of *data stream algorithm*, which is defined as follows.

Definition 20.5.1. A data stream algorithm has four requirements listed below:

1. The algorithm should make one sequential pass over the input.
2. The algorithm should use poly-log space. (In particular, it cannot store the entire input.)
3. The algorithm should have poly-log update time, i.e. whenever a new item appears, the algorithm should be able to update its internal state in poly-log time.
4. The algorithm should have poly-log reporting time, i.e. at any point of time the algorithm should be able to return the required answers in poly-log time.

Thus, ideally we would like to design a data stream algorithm to solve the hot items problem that we discussed earlier. Next, we formally define the hot items problem. We begin with the definition of frequencies of each stock:

Definition 20.5.2. Let f_ℓ denote the total count for the stock id ℓ. Initially $f_\ell = 0$, given (ℓ, u_ℓ), $f_\ell \leftarrow f_\ell + u_\ell$.

Next, we define the hot items problem.

Definition 20.5.3 (Hot Items Problem)**.** Given N different items, for m input pairs of data (i_ℓ, u_ℓ) for $1 \le \ell \le m$, where $i_\ell \in [N]$ indicates the item index and u_ℓ indicates corresponding count. The problem requires updating the count $f_\ell (1 \le \ell \le m)$ for each item, and to output all item indices j such that $f_j > \frac{\sum_{\ell=1}^{N} u_\ell}{d}$. (Any such item is called a *hot item.*)

Note that there can be at most d hot items. In this chapter, we will mostly think of d as $O(\log N)$. Hot items problem is also called heavy hitters problems. We state the result below without proof:

Theorem 20.5.1. *Computing hot items exactly by a deterministic one pass algorithm needs* $\Omega(n)$ *space (even with exponential time).*

This theorem means that we cannot solve the hot items problem in poly-log space as we want. However, we could try to find solutions for problems around this. The first one is to output an approximate solution, which will output a set that contains all hot items and some non-hot items. For this solution, we want to make sure that the size of the output set is not too large (e.g. outputting $[N]$ is not a sensible solution).

Another solution is to make some assumptions on the input. For example, we can assume Zipf-like distribution of the input data, which means only a few items appear frequently. More specifically, we can assume *heavy-tail distribution* on the input data, i.e.:

$$\sum_{\ell:\text{not hot}} f_\ell \le \frac{m}{d}. \tag{20.5}$$

This is reasonable for many applications, such as hot stock finding, where only a few of them have large frequency. Next, we will explore the connection between group testing and hot items problem based on this assumption.

20.5.1 Connection to Group Testing

Let us recall the naive solution that does not lead to a data stream algorithm: for each item $j \in [N]$, we maintain the actual count of number of trades for stock j. In other words, at any point of time, if C_j is the count for stock j, we have $C_j = f_j$. Another way to represent this is if M is the $N \times N$ identity matrix, then we maintain the vector of counts via $M \cdot \mathbf{f}$, where $\mathbf{f}$ is the vector of the frequencies of the items. Paralleling the story in group testing where we replace the identity matrix with a matrix with fewer rows, a natural idea here would be to replace M by matrix with fewer rows that utilizes the fact that there can be at most d hot items. Next, we show that this idea works if the heavy tail distribution holds. In particular, we will reduce the hot items problem to the group testing problem.

We now show how we solve the hot items problem from Definition 20.5.3. Let M be an $t \times N$ matrix that is d-disjunct. We maintain counters $C_1, \dots, C_t$, where each C_i is the total count of any item that is present in the ith row. We also maintain the total number of items m seen so far. Algorithm 30 and Algorithm 31 present the initialization and update algorithms. The reporting algorithm then needs to solve the following problem: at any point of time, given the counts $C_1, \dots, C_t$ and m output the at most d hot items.

Algorithm 30 Initialization

OUTPUT: Initialize the counters

```
1: $m \leftarrow 0$
2: FOR every $j \in [t]$ DO
3:     $C_j \leftarrow 0$
```

Algorithm 31 Update

INPUT: Input pair (i,u), $i \in [N]$ and $u \in \mathbb{Z}$
OUTPUT: Update the Counters

```
1: $m \leftarrow m+1$,
2: FOR every $j \in [t]$ DO
3:     IF $M_{ij} = 1$ THEN
4:         $C_j \leftarrow C_j + u$
```

Next, we reduce the problem of reporting hot items to the decoding problem of group testing. The reduction essentially follows from the following observations.

Observation 20.5.2. *If j is a hot item and $M_{ij} = 1$, then $C_i > \frac{m}{d}$.*

Proof. Let $i \in [t]$ be such that $M_{ij} = 1$. Then note that at any point of time,

$$C_i = \sum_{k:M_{ik}=1} f_k \geq f_j.^{2}$$

Since j is a hot item, we have $f_j > \frac{m}{d}$, which completes the proof. □

Observation 20.5.3. *For any $1 \leq i \leq t$, if all j with $M_{ij} = 1$ are not hot items, then we have $C_i \leq \frac{m}{d}$.*

Proof. Fix an $\in [t]$ such that every $j \in [N]$ such that $M_{ij} = 1$ is not a hot item. Then by the same argument as in proof of Observation 20.5.2, we have

$$C_i = \sum_{k:M_{ik}=1} f_k.$$

The proof then follows by the choice of i and (20.5). □

Armed with these observations, we now present the reduction. Define $\mathbf{x} = (x_1, x_2, \ldots, x_N) \in \{0,1\}^N$ with $x_j = 1$ if and only if j is a hot item, and $\mathbf{r} = (r_1, r_2, \ldots, r_t) \in \{0,1\}^t$ with $r_i = 1$ if and

[2]The equality follows e.g. by applying induction on Algorithm 31.

only if $C_i > \frac{m}{d}$, we will have $r_i = \vee_{j:M_{ij}=1} x_j$. The latter claim follows from Observations 20.5.2 and 20.5.3 above. This means we have:

$$M \odot \mathbf{x} = \mathbf{r}. \tag{20.6}$$

Note that by definition, $wt(\mathbf{x}) < d$. Thus reporting the hot items is the same as decoding to compute **x** given M and **r**, which successfully changes the hot items problem into group testing problem. Algorithm 32 has the formal specification of this algorithm.

Algorithm 32 Report Heavy Items

```
INPUT: Counters $m$ and $C_1, \dots, C_t$
OUTPUT: Output the heavy items

1: FOR every $j \in [t]$ DO
2:     IF $C_t > \frac{m}{d}$ THEN
3:         $r_j \leftarrow 1$
4:     ELSE
5:         $r_j \leftarrow 0$
6: Let $\mathbf{x}$ be the result of decoding (for group testing) $\mathbf{r}$
7: RETURN $\{i | x_i = 1\}$
```

Next, we will design and analyze the algorithm above and check if the conditions in Definition 20.5.1 are met.

Analysis of the Algorithm

In this part, we will review the requirements on data stream algorithm one by one and check if the algorithm for the hot items problem based on group testing satisfies them. In particular, we will need to pick M and the decoding algorithm. We will pick M to be the d-disjunct matrix from Theorem 20.4.2.

1. *One-pass requirement*
   If we use non-adaptive group testing, the algorithm for the hot items problem above can be implemented in one pass, which means each input is visited only once. (If adaptive group testing is used, the algorithm is no longer one pass, therefore we choose non-adaptive group testing.) We note that by definition, our choice of M satisfies this condition.

2. *Poly-log space requirement*
   In the algorithm, we have to maintain the counters C_i and m. The maximum value for them is mN, thus we can represent each counter in $O(\log N + \log m)$ bits. This means we need $O((\log N + \log m)\, t)$bits to maintain the counters. Theorem 20.4.2 implies that $t = O(d^2 \log_d^2 N)$. Thus, the total space we need to maintain the counters is $O(d^2 \log_d^2 N(\log N + \log m))$.

On the other hand, if we need to store the matrix M, we will need $\Omega(tN)$ space. Therefore, poly-log space requirement can be achieved only if matrix M is not stored directly. (We will tackle this issues in the next point.)

3. *Poly-log update time*
   As mentioned in the previous part, we cannot store the matrix M directly in order to have poly-log space. Since RS code is strongly explicit (see Exercise 6.4), we do not need to explicitly store M (we just need to store the parameters of the code C_{out} and C_{in}, which can be done in poly-log space). In the following, we will argue that the runtime of Algorithm 31 is $O(t \times \text{poly}\log t)$. It is easy to check the claimed time bound is correct as long as we can perform the check in Step 3 in $\text{poly}\log(t)$ time. In particular, we would be done if given $j \in [N]$, we can compute the column M^j in $O(t \times \text{poly}\log t)$ time. Next, we argue that the latter claim is correct.

   Recall that $M = M_{C^*}$, with $C^* = C_{\text{out}} \circ C_{\text{in}}$, where C_{out} is a $\left[q, k, q-k+1\right]_q$ RS code and C_{in} chosen such that $M_{C_{\text{in}}}$ is the $q \times q$ identity matrix. Recall that codewords of C^* are columns of the matrix M, and we have $n = q^k$, $t = q^2$.

   Since every column of M corresponds to a codeword of C^*, we can think of j equivalently as a message $\mathbf{m} \in \mathbb{F}_q{}^k$. In particular, M^j then corresponds to the codeword $C_{\text{out}}(\mathbf{m})$. On the other hand, the column M^j can be partitioned into q chunks, each chunk is of length q. Notice that $(C_{\text{out}}(\mathbf{m}))_{i_1} = i_2$ if and only if the i_1th chunk has 1 on its i_2th position and 0 on other positions (recall the definition of C_{in}). Therefore, we can compute M^j by computing $C_{\text{out}}(\mathbf{m})$. Because C_{out} is a linear code, $C_{\text{out}}(\mathbf{m})$ can be computed in $O(q^2 \times \text{poly}\log q)$ time,[3] implies that M^j can be computed in $O(q^2 \times \text{poly}\log q)$ time. Since we have $t = q^2$, the update process can be finished with $O(t \times \text{poly}\log t)$ time. (We do not need C_{out} to be strongly explicit: as long as C_{out} is linear the arguments so far work just as well.)

4. *Reporting time*
   It is easy to check that the run time of Algorithm 32 is dominated by Step 6. So far, the only decoding algorithm for M that we have seen is Algorithm 29, which runs in time $\Omega(tN)$, which does not satisfy the required reporting time requirement. In Exercise 20.11, we show that using the fact that C_{out} is the Reed-Solomon code, one can solve the decoding problem in $\text{poly}(t)$.

Thus, we have argued that

Theorem 20.5.4. *There exists a data streaming algorithm that computes d hot items with one pass, $O(t \log N)$ space for $t = O(d^2 \log_d^2 N)$, $O(t\text{poly}\log t)$ update time and* $\text{poly}(t)$ *reporting time.*

[3]This follows from Proposition 2.3.2 and the fact that C_{out} is strongly explicit

20.6 Exercises

Exercise 20.1 (Upper bound on $t^a(d,N)$). In this problem we will show that $t^a(d,N) = O(d\log(N/d))$. We begin by trying to prove a weaker bound of $O(d\log N)$:

- Show that one can identify at least one i such that $x_i = 1$ (or report none exist) with $O(\log N)$ adaptive tests.

  (*Hint:* Use binary search.)

- Using the scheme above argue that one can compute $\mathbf{x}$ with $O(wt(\mathbf{x})\cdot\log N)$ adaptive tests. Conclude that $t^a(d,N) \le O(d\log N)$.

Next we argue that we can tighten the bound to the optimal bound of $O(d\log(N/d))$:

- Argue that any scheme that computes $\mathbf{x} \in \{0,1\}^N$ with $O(wt(\mathbf{x})\cdot\log N)$ adaptive tests can be used to compute $\mathbf{x}$ with $O(d\log(N/d))$ adaptive tests where $wt(\mathbf{x}) \le d$.

- Conclude that $t^a(d,N) \le O(d\log(N/d))$.

Exercise 20.2. Show that every d-separable matrix is also $(d-1)$-disjunct.

Exercise 20.3. Prove that Algorithm 29 is correct and runs in time $O(tN)$.

Exercise 20.4. For every integer $d \ge 1$ and large enough integer $N \ge d$ show that there exists a d-disjunct matrix with $O(d^2\log(N/d))$ rows.

(*Hint:* Use the probabilistic method. It might help to pick each of tN bits in the matrix independently at random with the same probability.)

Exercise 20.5. We first begin by generalizing the argument of Theorem 20.4.2:

- Let C_{out} be an $(n,k,D)_q$ code. Let C_{in} be defined such that $M_{C_{\text{in}}}$ is the $q\times q$ identity matrix. Let $M_{C_{\text{out}}\circ C_{\text{in}}}$ be a $t\times N$ matrix that is d-disjunct. Derive the parameters d, t and N.

Next argue that it is enough to pick an outer random code to obtain a d-disjunct matrix with the same parameters obtained in Exercise 20.4:

- Pick $q = \Theta(d)$. Then using the previous part or otherwise show that if C_{out} is a random $[n,k,D]_q$ code, then the resulting $t\times N$ matrix $M_{C_{\text{out}}\circ C_{\text{in}}}$ is d-disjunct with $t = O(d^2\log N)$ for large enough N.

  (*Hint:* Use Theorem 4.2.1 and Proposition 3.3.5.)

Exercise 20.6. For every integer $d \ge 1$ and large enough $N \ge d$, construct a d-disjunct matrix with $O(d^2\log N)$ rows in (deterministic) time poly(N).

Hint: Recall Exercise 4.7.

Exercise 20.7 (Lower Bound on $t(d,N)$ due to Bassalygo). In this problem we will show that $t(d,N) \ge \min\left\{\binom{d+2}{2}, N\right\}$. In what follows let M be a $t\times N$ matrix that is d-disjunct.

(a) Argue that if $wt(M^j) < d$ then M^j has a *private row* i.e. there exists a row $i \in [t]$ such that $M_{ij} = 1$ but $M_{ij'} = 0$ for every $j' \neq j$.

(b) Using part (a) or otherwise, argue that if all columns of M have Hamming weight at most $d-1$, then $t \geq N$.

(c) Let M^{-j} for $j \in [N]$ be the matrix M with M^j as well as all rows $i \in [t]$ such that $M_{ij} = 1$ removed. Then argue that M^{-j} is $(d-1)$-disjunct.

(d) Argue that $t(1, N) \geq \min\{3, N\}$.

(e) Using induction with parts (b)-(d) or otherwise, argue that $t \geq \min\left\{\binom{d+2}{2}, N\right\}$.

Exercise 20.8 (Lower Bound on $t(d, N)$ due to Ruszinkó and Alon-Asodi). In this problem, we will show that

$$t(d, N) \geq \Omega\left(\min\{d^2 \log_d N, N\}\right). \tag{20.7}$$

In what follows let M be a $t \times N$ matrix that is d-disjunct.

(a) Argue that any $j \in [N]$ such that $wt(M^j) < \frac{2t}{d}$ has a *private subset* of size $\lceil 4t/d^2 \rceil$, i.e. there exists a subset $S \subseteq [N]$ with $|S| = \lceil 4t/d^2 \rceil$ such that M^j has ones in all $i \in S$ but for every $j \neq j'$, $M^{j'}$ has at least one row $i' \in S$ such that $M_{i'j'} = 0$.

(b) Using part (a) or otherwise, argue:

$$N - \frac{d}{2} \leq \binom{t}{\lceil 4t/d^2 \rceil}.$$

(c) Using Exercise 20.7 and part (b) or otherwise argue (20.7).

Exercise 20.9. In this exercise and the ones that follow it, we will consider the following equivalent version of the decoding problem: given $\mathbf{r} = M \odot \mathbf{x}$ with $wt(\mathbf{x}) \leq d$, output $\{i | x_i = 1\}$. Now consider the following easier version of the problem. In addition to $\mathbf{r}$ and M assume that one is also given a set S such that $\{i | x_i = 1\} \subseteq S$. Modify Algorithm 29 to design a decoding algorithm that computes $\{i | x_i = 1\}$ in time $O(t \cdot |S|)$.

Exercise 20.10. A $t \times N$ matrix M is called (d, L)-list disjunct if and only if the following holds for every disjoint subsets $S, T \subset [N]$ such that $|S| = d$ and $|T| = L - d$, there is a row in M where all columns in S have a 0 but at least one column in T has a 1.

- What is a $(d, d+1)$-list disjunct matrix?
- Let C_{out} be an $(n, k)_q$ code that is $(0, d, L)$-list recoverable code (recall Definition 16.3.2). Let C_{in} be the inner code such that $M_{C_{\text{in}}}$ is the $q \times q$ identity matrix. Argue that $M_{C_{\text{out}} \circ C_{\text{in}}}$ is (d, L) list disjunct.

Exercise 20.11. Using Exercises 20.9 and 20.10 or otherwise prove the following. Let M_{C^*} be the Kautz-Singleton matrix from Section 20.4.1. Then given $M_{C^*} \odot \mathbf{x}$ with $wt(\mathbf{x}) \leq d$, one can compute $\{i | x_i = 1\}$ in poly(t) time.
(*Hint:* Theorem 16.3.2 could be useful.)

20.7 Bibliographic Notes

Robert Dorfman's paper in 1943 [12] introduced the field of (Combinatorial) Group Testing. It must be noted that though this book covers group testing as an application of coding theory, it took off long before coding theory itself.

The original motivation arose during the Second World War when the United States Public Health service and the Selective Service embarked upon a large scale project. The objective was to weed out all syphilitic men called up for induction. [12].

The connection between group testing and hot items problem considered in Section 20.5 was established by Cormode and Muthukrishnan [11]. More details on data stream algorithms can be bound in the survey by Muthukrishnan [55].