Explicit Nonadaptive Combinatorial Group Testing Schemes

Ely Porat and Amir Rothschild

***Abstract*—Group testing is a long studied problem in combinatorics: A small set of r ill people should be identified out of the whole (n people) by using only queries (tests) of the form "Does set X contain an ill human?" In this paper we provide an explicit construction of a testing scheme which is better (smaller) than any known explicit construction. This scheme has $\Theta\left(\min[r^2 \ln n, n]\right)$ tests which is as many as the best nonexplicit schemes have. In our construction, we use a fact that may have a value by its own right: Linear error-correction codes with parameters $[m, k, \delta m]_q$ meeting the Gilbert-Varshamov bound may be constructed quite efficiently, in $\Theta\left(q^k m\right)$ time.**

I. Introduction

Group testing is an important and well-known tool in combinatorics. Due to its basic nature, it has been found to be applied in a vast variety of situations. In 2006, DIMACS has dedicated a special workshop solely for the problem of group testing [1]. A representative instance of group testing considers a set of items, each of which can be either defective or nondefective, and the task is to identify the defective items using the minimum number of tests. Each test works on a group of items simultaneously and returns whether or not that group contains at least one defective item. A group testing algorithm is said to be *nonadaptive* if all the tests to be performed are specified in advance. A formal definition is given in Section II.

Group testing has a long history dating back to at least 1943 [2]. In this early work, the problem of detecting syphilitic men for induction into the United States military using the minimum number of laboratory tests was considered. While this idea is still relevant today for testing viruses such as HIV, it is only one of the many applications found for group testing: In the effort of mapping genomes, for example, we have a huge library of DNA sequences, and test whether each of them contains a probe from a given set of DNA pieces [3]–[5]. Somewhat less conventional uses for group testing were introduced lately in pattern matching algorithms [6], [7] and in streaming algorithms [8]. For instance, [6] solves the problem of searching for a pattern in a text with a bounded number of mismatches. A recent paper about pattern matching in a streaming model even utilizes group testing twice in the same algorithm [9]. Additional applications of group testing include: compressed sensing [10]–[14], quality control in product testing [15], searching files in storage systems [16], sequential screening of experimental variables [17], efficient contention resolution algorithms for multiple-access communication [16], [18], data compression [19], software testing [20], [21], DNA sequencing [22] and other applications in computational molecular biology [23]–[26]. In most of the algorithms and applications presented here, our group testing algorithm improves the results.

Consider the situation where there are n items out of which at most r are defective. It has been shown that in this situation any nonadaptive combinatorial group testing ((n, r)-GT) procedure must use $\Omega(\min[r^2 \log_r n, n])$ tests [27]. The best known schemes use $\Theta\left(\min[r^2 \ln n, n]\right)$ tests [16], and the best-known explicit (polynomial time constructible) schemes need as much as $\Theta\left(\min[r^2 \log^2_{r \ln n} n, n]\right)$ tests [16]. In this paper, we present an explicit GT scheme which contains merely $t = \Theta\left(\min[r^2 \ln n, n]\right)$ tests (the same as the best known nonexplicit schemes), and takes $\Theta\left(rn \ln n\right)$ time to build, which is linear in the size of the scheme ($\mathcal{O}\left(t\frac{n}{r}\right)$) (as the size of the tests is about $\frac{n}{r}$). Hence, this paper closes the gap between the explicit and nonexplicit group testing schemes.

We claim that our construction is explicit. Unfortunately, the word explicit is ambiguous. The definitions can differ according to the time and space it takes to prepare the construction, and the space required to hold the construction. The strictest way to define an explicit construction requires the construction to take no time to build and logarithmic time to access (i.e., answering whether element i is in test s will take time $\mathcal{O}\left(\log_r n\right)$). The most lenient definition of an explicit construction will require the construction to take $\mathrm{poly} n$ time and space to construct and will require holding a representation of the construction of size $\mathrm{poly} n$ in-order to achieve $\mathcal{O}\left(\log n\right)$ time to access. Our construction is somewhere in the middle between those two definitions: Our construction preparing takes $\mathcal{O}\left(n\right)$ space and time that is linear in the size of the constructed sets ($\mathcal{O}\left(nr \log n\right)$). Our construction requires working space as small as $\mathcal{O}\left(r^2 \log n\right)$ once its prepared in-order to achieve $\mathcal{O}\left(\log n\right)$ time of access. Our construction can even be prepared using only $\mathcal{O}\left(r^2 \log n\right)$ space if we allow the preparation time to be slightly higher ($\mathcal{O}\left(nr \log n \log_r n\right)$ instead of $\mathcal{O}\left(nr \log n\right)$). This is, therefore, how we define the explicitness of the construction in this paper.

A. Error Correction Code (ECC)

An ECC is a method for encoding data in a redundant way, such that any errors which are introduced can be detected and corrected (within certain limitations). Suppose Alice wants to send Bob a string of k letters from an alphabet of size q using some noisy channel. An $(m, k, d)_q$ ECC enables Alice to encode her string to a string of length $m > k$, such that Bob will

Manuscript received October 04, 2010; revised April 03, 2011; accepted June 21, 2011. Date of publication August 01, 2011; date of current version December 07, 2011. This work was supported in part by BSF Grant 2006334 and in part by ISF Grant 1484/08.

The authors are with Bar-Ilan University, Ramat Gan, Israel (e-mail: amirrot@gmail.com)

Communicated by E. Arikan, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2011.2163296

be able to detect whether the received message has less than d errors, and even decode the message if it has less than $\frac{d}{2}$ errors. A linear code (LC) is an important type of ECC which allows more efficient encoding and decoding algorithms than other codes. ECCs are used in a vast variety of fields including information transmission, data preservation, data-structures, algorithms, complexity theory, and more.

One of the most important goals of coding theory is finding codes that can detect many errors, while having little redundancy. The Gilbert-Varshamov (GV) bound shows this can be done to some extent. We define the *rate* of a code $R = \frac{k}{m}$ and the *relative distance* of a code $\delta = \frac{d}{m}$. The GV bound asserts that there are codes with $R \geq 1 - H_q(\delta) - o(1)$ where $H_q(p)$ is the q-ary entropy function

$$H_q(p) = p \log_q \frac{q-1}{p} + (1-p) \log_q \frac{1}{1-p}$$

and $o(1) \xrightarrow[m\to\infty]{} 0$ [28], [29]. Though the GV bound is half a century old, no explicit construction of codes meeting it has yet been found. The best known construction takes polynomial time in q^{m-k} [30].

We present a more efficient deterministic construction for linear codes meeting the GV bound. Our construction takes $\Theta\left(q^k m\right)$ time. The importance of this result is apparent when constructing codes with low rates; First, for small rates the GV bound is the best known lower bound on the rate and relative distance of a code. Second, the lower the rate, the slower the previously known best construction, and the faster our construction.

B. Previous Results

Since the problem of group testing was first introduced in 1943, many problems related to it and generalizations of it were considered including: fully adaptive group testing, two staged group testing and selectors [31]–[36], group testing with inhibitors [33], [37]–[39], group testing in a random case where a distribution is given on the searched set [32], [33], [40]–[42], group testing in the presence of errors [43] and more. Regarding the original problem of group testing, Kautz and Singleton [16] proved the existence of GT schemes of size $\Theta\left(r^2 \ln n\right)$, and showed how to explicitly construct schemes of size $\Theta\left(\min[r^2 \log^2_{r \ln n} n, n]\right)$. They also managed to give an explicit construction of schemes of size $\Theta(\ln n)$ for the special case $r = 2$. Since their work, almost no asymptotic improvements to the size of the GT scheme were found. The one asymptotic improvement that was found presented a construction of size $\Theta\left(r^4 \log n\right)$ (which is incomparable to [16] and therefore improves the bound) and was given in [44]. Another paper, [45], succeeded in improving the size of the *explicit* schemes (but only for *constant* values of r) and showed how to construct an explicit construction of schemes of size $\Theta\left(r^2 \ln n\right)$ in time polynomial in n^r. From the probabilistic perspective, there is no known Las-Vegas algorithm (though one easily stems from our methods) constructing a scheme of size $\Theta\left(r^2 \ln n\right)$. The only known probabilistic constructions are Monte Carlo algorithms.

Regarding ECCs the picture is more complex. The GV bound was first presented by Gilbert in 1952 [28]. He provided a $\Theta(q^m)$ time greedy construction for codes meeting his bound. A few years later Varshamov [29] showed linear codes share this bound and Wozencraft [46] offered a $\Theta(q^m)$ time deterministic construction of such codes. In 1977, Goppa [47] initiated the fruitful study of algebraic geometric codes. Codes eventually found by this study surpass the GV bound for various alphabet sizes and rates [48]. Recently, an explicit, $\Theta\left(m^3 \text{polylog} m\right)$ time construction was given for algebraic geometric codes [49], [50]. The best deterministic construction for alphabet sizes and rates where the GV bound is superior to the algebraic geometric bound, was provided in 1993 by Brualdi and Pless [30]. They presented a $\text{poly}(q^{m-k})$ construction of binary linear codes meeting the GV bound (which is impairer to our construction iff $k > \frac{m}{2}$; codes with such high rates, though, can't help us achieve good group testing results). Their construction can be easily generalized to deal with larger alphabets. Even under hardness assumptions, no explicit construction of codes meeting the GV bound has yet been found, though an effort presenting some worthy results is given in [51].

C. Our Results

We present the first explicit (n, r)-GT scheme which contains $t = \Theta\left(\min[r^2 \ln n, n]\right)$ tests, thus closing the gap between explicit and nonexplicit group testing schemes. Our construction takes $\Theta(rn \ln n)$ time to build, meaning linear time in the size of the sets $\left(\mathcal{O}\left(t\frac{n}{r}\right)\right)$.

Theorem 1: Let n and r be positive integers. It is possible to construct a (n, r)-GT containing $\Theta\left(\min[r^2 \ln n, n]\right)$ tests in $\Theta(rn \ln n)$ time.

We also present the most efficient deterministic construction for linear codes meeting the GV bound. Our construction builds an $[m, k, \delta m]_q$-LC in $\Theta\left(q^k m\right)$ time.

Theorem 2: Let q be a prime power, m and k positive integers, and $\delta \in [0, 1]$. If $k \leq (1 - H_q(\delta))\, m$, then it is possible to construct an $[m, k, \delta m]_q$-LC in time $\Theta\left(mq^k\right)$.

D. The Paper Outline

We start this paper with formal definitions in Section II, and continue by showing a connection between ECCs and group testing schemes in Section III. Then we immediately move to the main result of the paper in Section IV, showing how to efficiently construct small group testing schemes. This construction for group testing schemes uses our construction of a linear code which is given in Section V.

II. Preliminaries

Definition 2.1: Consider a universe U. A family of tests (subsets) $\mathcal{F} \subset \mathcal{P}(U)$ is a group testing scheme of strength r ((n, r)-GT) if for any subset $A \subset U$ of size at most r, and for any element $x \notin A$, there exists a test $B \in \mathcal{F}$ that distinguishes x from A, meaning $x \in B$ while $A \cap B = \emptyset$.

In order to ease reading, we present short notations of an ECC and a linear code.

Definition 2.2: An ECC $\mathcal{C}$ is said to have parameters $(m, k, d)_q$ if it consists of q^k words of length m over alphabet

Σ of q elements, and has Hamming distance d. Such an ECC is denoted as $(m,k,d)_q$-ECC.

Definition 2.3: An $[m,k,d]_q$-LC is a special case of an $(m,k,d)_q$-ECC which is over alphabet $\Sigma = \mathbb{F}_q$ when the codewords are form a linear subspace over $\mathbb{F}_q^m$. Such a linear code is said to have parameters $[m,k,d]_q$. A linear code has a *generator matrix* $\mathcal{G} \in \mathcal{M}_{m\times k}$ which generates it, meaning $\mathcal{C} = \{\mathcal{G}y \mid y \in \mathbb{F}_q^k\}$.

When dealing with linear codes, the weight of a code word and the weight of a code are two useful concepts.

Definition 2.4: The weight of a code word is the number of letters in it which are not 0: $\omega(x) = |\{i \mid x_i \neq 0\}|$. The weight of a code is defined as the minimal weight of all code words in it: $\omega(\mathcal{C}) = \min_{x\in\mathcal{C}} \omega(x)$.

The reason why those concepts are important is that the weight of a linear code is equal to its distance.

III. Background

Our results concerning GT are more natural and straightforward using the combinatorial concepts *selection by intersection* and *strongly selective family* (SSF) [52]. Selection by intersection means distinguishing an element from a set of elements by intersecting it with another set. More precisely, see what follows.

Definition 3.1: Given a subset $A \subset U$ of a universe U, element $x \in A$ is *selected* by subset $B \subset U$ if $A \cap B = \{x\}$. An element is *selected* by a family of subsets $\mathcal{F} \subset \mathcal{P}(U)$ if one of the subsets in $\mathcal{F}$ selects it.

An SSF is a family of subsets that selects any element out of a small enough subset of the universe. More precisely, see what follows.

Definition 3.2: A family $\mathcal{F} \subset \mathcal{P}(U)$ is said to be (n,r)-*strongly-selective* if $|U| = n$ and for every subset $A \subset U$ of size $|A| = r$, all elements of A are selected by $\mathcal{F}$. We call such a family an (n,r)-SSF.

SSFs and GT schemes are strongly connected: On the one hand, an $(n,r+1)$-SSF is a GT scheme of strength r, and on the other hand, a GT scheme of strength r in a universe of size n is an (n,r)-SSF. For a detailed proof, see [16].

In what follows, we will focus on SSF constructions. It is important to note that explicit constructions for SSFs give explicit constructions for GT schemes with the same asymptotic behavior. Next we show how to construct an SSF from an ECC, and analyze how good this construction is. The foundations of the idea that we present were developed in an earlier work by Kautz and Singleton on superimposed codes [16]. The context and formalisms that were employed are quite distinct from those we require, and even though the idea is quite simple, we are not aware of this aspect of their work being developed subsequently. Thus, the following subsection will provide full and complete explanations and proofs of the construction.

A. Reducing ECCs to SSFs

As it turns out, one can build small SSFs from good ECCs having large distance. Both the construction and the proof are

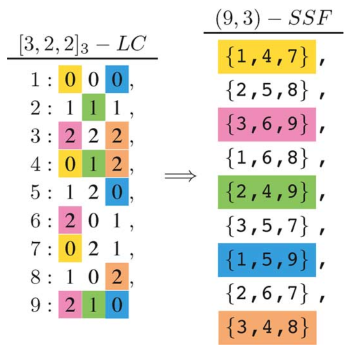

Fig. 1. The reduction from an ECC to an SSF. The left hand side of the figure contains the code words of the Reed-Solomon $[3,2,2]_3$-LC. The right hand side contains the tests in the generated $(9,3)$-SSF. The colors help us see that each test contains the indexes of code words which have the same symbol in the same position. For example, the first test contains the indexes of the code words which have 0 in the first position.

```
foreach $i \in [n]$ do
    foreach $p \in [m]$ do
        insert $i$ into $s_{p,w_i[p]}$ ;
```

Algorithm 1. Constructing an SSF from an ECC.

given in this subsection. In a few words, the idea behind the construction is that taking a small set of codewords from the ECC and another codeword w, there must be positions in which w differs from all the words in this set. This is because w differs from any other word in the code in many positions, and so, in a small set of codewords, there must be some shared positions in which all codewords differ from w. Therefore we will get an SSF if we first translate elements of $[n]$ to codewords, and second, find tests which isolate a codeword w from a set of codewords if it differs from this set in a certain position. We construct such tests by assembling a test for each possible letter in each possible position in the word. A detailed construction follows.

Suppose $\mathcal{C} = \{w_1,\ldots,w_n\}$ is an $(m,\log_q n,\delta m)_q$-ECC. The constructed SSF $\mathcal{F}(\mathcal{C})$ will be assembled from all the sets of indexes of codewords that have a certain letter in a certain position. More accurately, for any $p \in [m]$ and $v \in [q]$, define $s_{p,v} = \{i \in [n] \mid w_i[p] = v\}$. Define $\mathcal{F}(\mathcal{C})$ as the set of all such $s_{p,v}$-s: $\mathcal{F}(\mathcal{C}) = \{s_{p,v} \mid p \in [m] \text{ and } v \in [q]\}$. Fig. 1 demonstrates how our algorithm generates a (9,3)-SSF using a Reed-Solomon $[3,2,2]_3$-LC.

The size of $\mathcal{F}(\mathcal{C})$ is at most mq. Notice that this construction may be performed in time $\Theta(nm)$ (linear in the size of the sets of $\mathcal{F}$) using Algorithm 1.

The following Lemma shows that this construction really does result in a small SSF, and more specifically, that $\mathcal{F}(\mathcal{C})$ is an $(n,\lceil\frac{1}{1-\delta}\rceil)$-SSF.

Lemma 3.1: Let $\mathcal{C}$ be an $(m,\log_q n,\delta m)_q$-ECC. Then $\mathcal{F}(\mathcal{C})$ is an $(n,\lceil\frac{1}{1-\delta}\rceil)$-SSF.

Proof: Let $r = \lceil\frac{1}{1-\delta}\rceil$. Let $i_1,\ldots,i_r \in [n]$ be any r distinct indexes in $[n]$. W.l.o.g. we prove that i_1 is selected from $\{i_1,\ldots,i_r\}$ by $\mathcal{F}(\mathcal{C})$. For any $j \neq 1$, the number of positions $p \in [m]$ where $w_{i_j}[p] = w_{i_1}[p]$ is at most $(1-\delta)m$. Thus, the number of positions where $w_{i_1}[p] \in \{w_{i_2}[p],\ldots,w_{i_r}[p]\}$

is at most $(r-1)(1-\delta)m < m$. Therefore, there exist a position p where $w_{i_1}[p] \notin \{w_{i_2}[p], \ldots, w_{i_r}[p]\}$. This means that $i_1 \in s_{p,w_{i_1}[p]}$ while all other i_j-s are not. Thus, i_1 is selected by $s_{p,w_{i_1}[p]}$. ■

Notice that if one wants to reduce the space required to hold the tests representation, one doesn't have to run algorithm 1 and may just hold the matrix of the ECC (whose space consumption in our case is merely $\mathcal{O}\left(r^2 \log n\right)$). Naturally, in that case checking whether an element i is in set s will take $\log_r n$ time.

IV. Main Theorem

Theorem 1: Let n and r be positive integers. It is possible to construct an (n,r)-SSF of size $\Theta\left(\min[r^2 \ln n, n]\right)$ in $\Theta(rn \ln n)$ time.

Proof: If $r^2 \ln n \geq n$, simply return the n tests $\{i\}_{i=1}^n$. We continue the proof assuming that $r^2 \ln n < n$. Set $\delta = \frac{r-1}{r}$ (which is equivalent to $r = \frac{1}{1-\delta}$), $q \in [2r, 4r)$ a prime power, $k = \log_q n$ and $m = \frac{k}{1-H_q(\delta)}$.

We first prove that $m = \Theta(rk \ln r) = \Theta\left(r \log_q n \ln r\right) = \Theta(r \ln n)$

$$\begin{aligned}1 - H_q(\delta) &= 1 - \left(1-\frac{1}{r}\right)\log_q \frac{q-1}{1-\frac{1}{r}} - \frac{1}{r}\log_q r = \\ &= \left(1-\frac{1}{r}\right)\left(1-\log_q(q-1)\right) \\ &\quad + \left(1-\frac{1}{r}\right)\log_q\left(1-\frac{1}{r}\right) \\ &\quad + \frac{1}{r}(1-\log_q r) \\ &= \frac{1}{\ln q}\left[\left(1-\frac{1}{r}\right)(\ln q - \ln(q-1))\right. \\ &\qquad + \left(1-\frac{1}{r}\right)(\ln(r-1)-\ln r) \\ &\qquad \left. + \frac{1}{r}\ln\frac{q}{r}\right]\end{aligned}$$

Using first order approximation we see that

$$\ln x - \ln(x-1) = \frac{1}{x} + o\left(\frac{1}{x}\right). \tag{1.1}$$

Using (1.1) in the expression for $1 - H_q(\delta)$ we get:

$$\begin{aligned}1 - H_q(\delta) &= \frac{1}{\ln q}\left[\left(1-\frac{1}{r}\right)\frac{1}{q} - \left(1-\frac{1}{r}\right)\frac{1}{r} + \frac{1}{r}\ln\frac{q}{r}\right] \\ &\quad + o\left(\frac{1}{r\ln q}\right) \\ &= \frac{1}{\ln q}\left[\frac{1}{q} - \frac{1}{r} + \frac{1}{r}\ln\frac{q}{r}\right] + o\left(\frac{1}{r\ln q}\right) \\ &= \frac{1}{r\ln q}\left[\frac{r}{q} - 1 + \ln\frac{q}{r}\right] + o\left(\frac{1}{r\ln q}\right).\end{aligned}$$

Denote $c = \frac{q}{r}$. Therefore $2 \leq c \leq 4$. Analyzing the function $\frac{1}{c} - 1 + \ln c$, one can easily be convinced that $0 < \frac{1}{2} - 1 + \ln 2 \leq \frac{1}{c} - 1 + \ln c \leq \frac{1}{4} - 1 + \ln 4$. Hence we get

$$\frac{1}{1-H_q(\delta)} = \Theta(r \ln r).$$

```
Input: $m, k \in \mathbb{N}$, $\delta \in [0,1]$ s.t. $k \leq (1-H_q(\delta))m$
Pick entries of the $m \times k$ generator matrix $\mathcal{G}$
uniformly and independently at random from $\mathbb{F}_q$;
Output: $\mathcal{G}$
```

Algorithm 2. Probabilistic Construction of a Linear Code.

Therefore

$$m = \Theta(kr \ln r).$$

Use Theorem 3 to construct an $[m,k,\delta m]_q$-LC in time $\Theta(nm)$. This is possible since $k \leq (1-H_q(\delta))\,m$.

According to Lemma 3.1, we can now construct an (n,r)-SSF of size $mq = \Theta\left(r^2 \ln n\right)$. The time this construction will take is $\Theta(nm) = \Theta(rn \ln n)$. ■

V. Meeting the GV Bound More Efficiently

Here, we demonstrate a deterministic construction of LCs which meets the GV bound. We developed this deterministic algorithm by taking a randomized algorithm and derandomizing it using the method of conditional probabilities (a full discussion concerning this method is given in [53]). Using this method requires the randomized algorithm to have several nontrivial attributes. First, there need to be a goal function $\mathrm{goal}: LinearCodes \rightarrow \mathbb{R}$ which returns a large result whenever the randomized algorithm fails. Second, this function has to have low expectation—lower than the minimum value returned by it when the algorithm fails. Third, the random selections of the algorithm have to be divided into stages with a small number of options to choose from in each. Finally, there should be an efficient algorithm for computing in each stage of the algorithm the option minimizing the expectation of goal given all the selections done until that point. In Section V-A, we will show the randomized algorithm, present the goal function *goal*, show that the algorithm fails iff $\mathrm{goal}(\mathcal{G}) \geq 1$ (where $\mathcal{G}$ is the generator matrix returned by the algorithm), and show that $E(\mathrm{goal}) < 1$. In Section V-B, we will present the derandomized algorithm more accurately, showing how to divide it to the small stages. We will also prove it should work, and show how to compute the option minimizing the expectation of goal in each stage. We will finish this subsection having an algorithm taking time polynomial in the complexity we desire, we improve it in Section V-C to acquire the desired complexity.

A. The Probabilistic Algorithm

Algorithm 2 is a standard probabilistic algorithm for building linear codes with rate meeting the GV bound.

Definition 5.1: Given a codeword x of length m, and a distance parameter $\delta \in [0,1]$, we define $\mathcal{B}_\delta(x)$ as the bad event that the weight of x is less than δm, $\omega(x) < \delta m$. By abuse of notation we refer to $\mathcal{B}_\delta(x)$ also as the indicator of the same event.

If we manage to choose a code with no bad event (not considering the 0 codeword, of course), then the weight of the generated code is larger than δm. As the weight and distance of a linear code are equal, the algorithm succeeds. Therefore, our goal function will be $\mathrm{goal}(\mathcal{G}) = \sum_{0 \neq y \in \mathbb{F}_q^k} \mathcal{B}_\delta(\mathcal{G}y)$. The algorithm succeeds iff $\mathrm{goal}(\mathcal{G}) = 0$. We now need to show that $E(\mathrm{goal})$ is small. Therefore, we are interested in proving that the probability of a bad event is sufficiently small. In order to do so, we use the following version of the Chernoff bound.

Theorem 2 (Chernoff Bound [54]): Assume random variables $X_1, \dots, X_m$ are i.i.d. and $X_i \in [0,1]$. Let $\mu = E(X_i)$, and $\epsilon > 0$. Then $Pr\left(\frac{1}{m}\sum X_i \geq \mu + \epsilon\right) \leq \left(\left(\frac{\mu}{\mu+\epsilon}\right)^{\mu+\epsilon}\left(\frac{1-\mu}{1-\mu-\epsilon}\right)^{1-\mu-\epsilon}\right)^m = e^{-D(\mu+\epsilon\,|\,|\,\mu)m}$ where $D(x\,|\,|\,y) = x\ln\frac{x}{y} + (1-x)\ln\frac{1-x}{1-y}$.

Lemma 5.1: Let y be a nonzero vector in $\mathbb{F}_q^k$. Let $\mathcal{G}$ be a random generator matrix chosen according to algorithm 2. Then $\log_q\left(Pr\left(\mathcal{B}_\delta(\mathcal{G}y)\right)\right) \leq -m\left(1 - H_q\left(\delta\right)\right)$.

Proof: We start by showing that $x = \mathcal{G}y$ is a random vector in $\mathbb{F}_q^m$: As $y \neq 0$, it has an entry which is not 0. Suppose, w.l.o.g, that this entry is y_1. Hence, $x = y_1\mathcal{G}_{\downarrow 1} + \sum_{j=2}^m y_j\mathcal{G}_{\downarrow j}$. As $\mathcal{G}_{\downarrow 1}$ is a random vector in $\mathbb{F}_q^m$, and any non zero element in a field is invertible, $y_1\mathcal{G}_{\downarrow 1}$ is randomly distributed. Therefore, and as the entries of $\mathcal{G}$ are independent, for any fixed $\mathcal{G}_{\downarrow 2}, \dots, \mathcal{G}_{\downarrow m}$, we get that x is randomly distributed. Hence, even if we don't fix $\mathcal{G}_{\downarrow 2}, \dots, \mathcal{G}_{\downarrow m}$, x will still be randomly distributed.

As x is randomly distrubuted, $\omega(x)$ is binomially distributed; $\omega(x) \sim B\left(m, 1 - \frac{1}{q}\right)$. In other words, $\omega(x) = m - \sum_{i=1}^m X_i$ where $X_i \in \{0,1\}$ are i.i.d and $E(X_i) = \frac{1}{q}$. Using the Chernoff bound (Theorem 2) we get

$$\begin{aligned} Pr\left(\mathcal{B}_\delta(x)\right) &= Pr\left(\omega(x) \leq \delta m\right) \\ &= Pr\left(m - \sum_{i=1}^m X_i \leq \delta m\right) \\ &= Pr\left(\sum_{i=1}^m X_i \geq (1-\delta)m\right) \\ &\leq e^{-D\left(1-\delta\,|\,|\,\frac{1}{q}\right)m}. \end{aligned}$$

Next, we show that $D(1-\delta\,|\,|\,\frac{1}{q}) = \ln q(1 - H_q(\delta))$

$$\begin{aligned} D\left(1-\delta\,|\,|\,\frac{1}{q}\right) &= (1-\delta)\ln((1-\delta)q) + \delta\ln\frac{\delta}{1-\frac{1}{q}} \\ &= \ln q\left((1-\delta)\log_q((1-\delta)q)\right. \\ &\quad \left. + \delta\log_q\frac{\delta q}{q-1}\right) \\ &= \ln q\left((1-\delta)(\log_q(1-\delta)+1)\right. \\ &\quad \left. + \delta\left(\log_q\frac{\delta}{q-1} + 1\right)\right) \\ &= \ln q\left(1 + (1-\delta)\log_q(1-\delta)\right. \\ &\quad \left. + \delta\log_q\frac{\delta}{q-1}\right) \\ &= \ln q\left(1 - (1-\delta)\log_q\frac{1}{1-\delta}\right. \\ &\quad \left. - \delta\log_q\frac{q-1}{\delta}\right) \\ &= \ln q\left(1 - \delta\log_q\frac{q-1}{\delta}\right. \\ &\quad \left. - (1-\delta)\log_q\frac{1}{1-\delta}\right) \\ &= \ln q(1 - H_q(\delta)). \end{aligned}$$

```
Input: $m, k \in \mathbb{N}$, $\delta \in [0,1]$ s.t. $k \leq (1 - H_q(\delta))m$
Initialize $\mathcal{G}$ to be an $m \times k$ matrix;
foreach $i \in [m]$ do
    foreach $j \in [k]$ do
        Set $\mathcal{G}[i,j]$ so as to minimize the expected
        value of goal($\mathcal{G}$) assuming the entries of $\mathcal{G}$
        which were not yet chosen distribute
        uniformly on $\mathbb{F}_q$, and independently of one
        another.
Output: $\mathcal{G}$
```

Algorithm 3. Finding a code having no Bad Events.

Finally, employ the fact that $D(1-\delta\,|\,|\,\frac{1}{q}) = \ln q(1-H_q(\delta))$, to get that

$$\log_q\left(Pr\left(\mathcal{B}_\delta(x)\right)\right) \leq -\frac{1}{\ln q}D\left(1-\delta\,|\,|\,\frac{1}{q}\right)m = -m(1-H_q(\delta))$$

■

We will now show that for an appropriate choice of parameters, the expected number of bad events $E(\text{goal})$ is smaller than 1.

Lemma 5.2: Suppose $\mathcal{G}$ is a random generator matrix chosen according to algorithm 2. Suppose that $k \leq (1-H_q(\delta))m$. Then $E(\text{goal}) < 1$.

Proof: By linearity of the expectation

$$\begin{aligned} E(\text{goal}) &= E\left(\sum_{y\neq 0}\mathcal{B}_\delta(\mathcal{G}y)\right) = \sum_{y\neq 0}E\left(\mathcal{B}_\delta(\mathcal{G}y)\right) \\ &= \sum_{y\neq 0}Pr\left(\mathcal{B}_\delta(\mathcal{G}y)\right). \end{aligned}$$

Next, employ Lemma 5.1 to acquire that

$$E(\text{goal}) \leq (q^k - 1)q^{-m(1-H_q(\delta))} < q^{k-m(1-H_q(\delta))}.$$

And finally, use our assumption $k \leq (1 - H_q(\delta))m$ to achieve the desired result $E(\text{goal}) < 1$ ■

B. Derandomizing the Algorithm

Next we will show how to derandomize the algorithm. Algorithm 3 will determine the entries of the generator matrix one by one, while trying to minimize the expectation of the number of bad events, goal.

Two questions arise from the above description of the algorithm: First, will this algorithm find a code with no bad events? Second, how can we find the value of $\mathcal{G}[i,j]$ in each step of the algorithm?

The answer to the first question is, of course, positive. The presented algorithm works according to the derandomization scheme of conditional probabilities, and so, the number of bad events in the returned solution will be no more than the expectation of this number before fixing any of the letters. We will delve into the proof after introducing some additional notations concerning the algorithm.

Definition 5.2: We assert that the algorithm is in *step*-(i,j) when it is about to choose the entry (i,j) in $\mathcal{G}$. We denote the step following (i,j) by $(i,j)+1$.

$$\begin{pmatrix} y_0 & y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 \end{pmatrix} = $$
$$= \begin{pmatrix} 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \end{pmatrix}$$

Fig. 2. The lexicographic order of the vectors in $\mathbb{F}_3^2$.

Note that the above definition results in a natural total order on the couples $((i,j)$: $(1,1) < (1,1)+1 < (1,1)+1+1 < \cdots < (m,k))$.

Definition 5.3: $ST_{(i,j)}$ will denote the state of the matrix $\mathcal{G}$ at step (i,j)—i.e., the values to which entries (a,b) have been fixed for $(a,b) < (i,j)$.

Lemma 5.3: The above algorithm will find a code with no bad events, i.e., goal$(\mathcal{G}) = 0$.

Proof: Suppose the algorithm is in some step (i,j).

$$Pr(\mathcal{B}_\delta(\mathcal{G}y) \mid ST_{(i,j)}) =$$
$$= \frac{1}{q}\sum_{v\in\Sigma} Pr\left(\mathcal{B}_\delta(\mathcal{G}y) \mid ST_{(i,j)},\ \mathcal{G}[i,j]=v\right)$$

Consequently

$$\begin{aligned} E\left(\text{goal} \mid ST_{(i,j)}\right) &= E\left(\sum_{y\neq 0}\mathcal{B}_\delta(\mathcal{G}y) \mid ST_{(i,j)}\right) \\ &= \sum_{y\neq 0} Pr\left(\mathcal{B}_\delta(\mathcal{G}y) \mid ST_{(i,j)}\right) \\ &= \frac{1}{q}\sum_{v\in\Sigma}\sum_{y\neq 0} Pr\left(\mathcal{B}_\delta(\mathcal{G}y) \mid \right. \\ &\qquad \left. ST_{(i,j)},\ \mathcal{G}[i,j]=v\right) \\ &\geq \min_{v\in\Sigma}\sum_{y\neq 0} Pr\left(\mathcal{B}_\delta(\mathcal{G}y) \mid \right. \\ &\qquad \left. ST_{(i,j)},\ \mathcal{G}[i,j]=v\right) \\ &= \sum_{y\neq 0} Pr\left(\mathcal{B}_\delta(\mathcal{G}y) \mid ST_{(i,j)+1}\right) \\ &= E\left(\sum_{y\neq 0}\mathcal{B}_\delta(\mathcal{G}y) \mid ST_{(i,j)+1}\right) \\ &= E\left(\text{goal} \mid ST_{(i,j)+1}\right). \end{aligned}$$

Therefore, if the values of the entries are chosen one by one, so as to minimize the expectation of *goal*, this value can not increase. Since this value is smaller than 1 at the beginning according to Lemma 5.2, it follows that it is smaller than 1 in the end. But at the end all entries are chosen, and hence the value of goal will be exactly the number of bad events that hold for the codewords we have chosen. This number must be an integer, hence, it is 0. ■

The answer to the second question, regarding finding what the value of $\mathcal{G}[i,j]$ should be, requires additional work. It would be convenient to order the vectors $y \in \mathbb{F}_q^k$ according to the lexicographic order, setting y_ℓ to be the ℓ-th vector according to the lexicographic order. For instance, the order of the vectors of $\mathbb{F}_3^2$ is shown in Fig. 2.

Concentrate for a moment on how this algorithm progresses: When entering step (i,j) all entries $\{(a,b) \mid (a,b) < (i,j)\}$ have been fixed, while all other entries have not been fixed. Moving our attention to the code words, one can see that fixing those entries in $\mathcal{G}$ actually fixes some of the letters in some of the code words while leaving all other letters distributing uniformly (though not independently) on $\mathbb{F}_q$. The reason is that each letter of a code word is of the form $(\mathcal{G}y_\ell)[a] = \sum_{b=1}^{k}\mathcal{G}[a,b]y_\ell[b]$. Each letter for which all nonzero entries of y_ℓ correspond to fixed entries in $\mathcal{G}$ will be fixed. All other letters will distribute uniformly. The code words letters which are fixed are therefore the $i-1$ first letters in each code word, the i-th letter in code words $\{\mathcal{G}y_\ell \mid \ell < q^{j-1}\}$, and, of course, all the letters of the 0 code word. An illustration is given in Fig. 3.

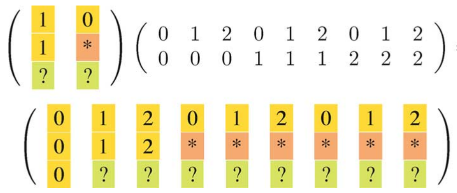

Fig. 3. The process of choosing the generating matrix of a code. This figure shows step $(2,2)$ of choosing a 3×2 generating matrix over $\mathbb{F}_3$ (for a code with 9 code words of length 3). One can see the entries which were fixed in both the generating matrix and the code words, the entries which are about to be fixed, and the entries which will only be fixed in later stages.

We need to know for any codeword the number of positions in which its letters have been fixed to zero. For this purpose maintain an array A of q^k entries throughout the algorithm. Entry $A[\ell]$ in this array will hold the number of positions in which the letters of $\mathcal{G}y_\ell$ have been fixed to zero so far. Maintaining this array will require overall $\Theta\left(mq^k\right)$ time and $\Theta\left(q^k\right)$ space. This is due to the fact that in each step (i,j) we only need to consider changing the values $A[y_\ell]$ for $q^{j-1} \leq \ell < q^j$ since the only letters we fix during this step belong to these words. Another option we have in order to get the number of zeros in each such position is simply to calculate it in each word at the point we need it, without storing the array. This will cost km time in each turn making the time complexity of our algorithm kmq^k, but will cost no extra memory. We claim that the number of positions where the word $\mathcal{G}y_\ell$ vanishes determines the conditioned probability of $\mathcal{B}_\delta(y_\ell)$.

Lemma 5.4: Consider a codeword $\mathcal{G}y_\ell$ for which all entries up to i are fixed (by the entries selected in $\mathcal{G}$), and entries i to m are not fixed yet. In other words, there exists a word $f \in \mathbb{F}_q^i$ of length i, such that for each possible $\mathcal{G}$, and $\forall t \leq i : (\mathcal{G}y_\ell)[t] = f[t]$, and the same is not true for $i+1$. Also suppose that until now, exactly c positions of $\mathcal{G}y_\ell$ were fixed to nonzeros ($c = |\{t \leq i \mid f[t] \neq 0\}|$). Then $\omega(\mathcal{G}y_\ell) - c \sim B(m-i, 1-\frac{1}{q})$, and $Pr(\mathcal{B}_\delta(\mathcal{G}y_\ell) \mid \forall t \leq i : (\mathcal{G}y_\ell)[t] = f[t])$ is the probability that such a binomial variable is smaller than $\delta m - c$.

Proof: Any entry which wasn't fixed in $\mathcal{G}y_\ell$, has a probability of $1-\frac{1}{q}$ to vanish. The entries in $\mathcal{G}y_\ell$ are independent of one another, and thus, $\omega(\mathcal{G}y_\ell) - c \sim B(m-i, 1-\frac{1}{q})$. ■

Now, in step (i,j), For any codeword $\mathcal{G}y_\ell$ s.t. $q^{j-1} \leq \ell < q^j$, we can compute the probabilities $Pr(\mathcal{B}_\delta(y_\ell) \mid ST_{(i,j)},\ \mathcal{G}[i,j]=v)$ for all $v \in [q]$ in poly(q^k, m) time using Lemma 5.4. Consequently, we can compute all the expectations $E(\sum_{q^{i-1}\leq\ell<q^i}\mathcal{B}_\delta(y_\ell) \mid ST_{(i,j)}, G[i,j]=v)$ for all $v \in [q]$ in poly(q^k, m) time and find the value of v which minimizes

this expectation. Hence, we can complete Algorithm 3 in $\text{poly}(q^k, m)$ time. In the following subsection, we improve this algorithm, showing how to achieve the desired complexity.

C. Improving the Deterministic Algorithm

In order to find the letter v which minimizes $E_{i,j,v} = E(\sum_{q^{j-1} \leq \ell < q^j} \mathcal{B}_\delta(y_\ell) \mid ST_{(i,j)}, \mathcal{G}[i,j] = v)$, we do not actually have to compute the q expectations $E_{i,j,v}$. It is enough to calculate the differences of those expectations and a constant value. We will use the constant value which is the expected number of bad events given $ST_{(i,j)}$ and that $(\mathcal{G}y_\ell)[i] \neq 0$ for all $q^{j-1} \leq \ell < q^j$ (Of course, it is improbable that no letters would vanish in step (i,j), as the purpose of this assumption is only to help us with the proof). We denote this constant value $E_{i,j}$.

According to Lemma 5.4, for any vector y the following holds:

$$\begin{gathered} Pr(\mathcal{B}_\delta(\mathcal{G}y) \mid ST_{i,j}, (\mathcal{G}y)[i] = 0) \\ - Pr(\mathcal{B}_\delta(\mathcal{G}y) \mid ST_{i,j}, (\mathcal{G}y)[i] \neq 0) = \\ \binom{m-i}{\delta m - c} \left(1 - \frac{1}{q}\right)^{\delta m - c} \left(\frac{1}{q}\right)^{(m-i)-(\delta m - c)}. \end{gathered}$$

Denote the above expression $\text{Dif}_{i,j}(y)$. Let T be the time it takes to calculate this expression. Now, we can calculate all q differences $E_{i,j,v} - E_{i,j}$ quite efficiently in the following manner: Initialize a size q array W. Then, run over the vectors y_ℓ for $q^{j-1} \leq \ell < q^j$, and for each subtract the difference $\text{Dif}_{i,j}(y_\ell)$ from cell $v = -y_\ell[j]^{-1} \sum_{t=0}^{j-1} \mathcal{G}[i,t] y_\ell[t]$ in W (since this cell means setting $(\mathcal{G}y_\ell)[i] = \sum_{t=0}^{j-1} \mathcal{G}[i,t] y_\ell[t] + \mathcal{G}[i,j] y_\ell[j] = 0$). After considering all values of y_ℓ, the position with the maximal value in W is the letter we should set for $\mathcal{G}[i,j]$. Each entry number v can be calculated in amortized constant time for all $q^{j-1} \leq \ell < q^j$ if we traverse over the ℓ-s in each step according to simple ascending order. Overall, the program will calculate mq^k entries, and so, it will take $mq^k T$ time.

Finally, this T factor is a bit annoying and we can get rid of it. As it comes to it, all we need here is to evaluate the product of 3 expressions: $\binom{m-i}{\delta m - c}$, $\left(1 - \frac{1}{q}\right)^{\delta m - c}$, and $\left(\frac{1}{q}\right)^{(m-i)-(\delta m - c)}$. The evaluation of this product will take constant time if these expressions are precomputed. Therefore, lets prepare these 3 expression in preprocessing:

- Evaluate all the values of $a!$ for any $a \in [m]$ in preprocess in time $\mathcal{O}(m)$. This will allow you to evaluate $\binom{m-i}{\delta m - c}$ in constant time.
- Evaluate all the values of $\left(1 - \frac{1}{q}\right)^a$ for $a \in [m]$ in preprocess in time $\mathcal{O}(m)$. This will allow you to calculated $\left(1 - \frac{1}{q}\right)^{\delta m - c}$ in constant time.
- Evaluated all the values of $\left(\frac{1}{q}\right)^a$ for $a \in [m]$ in preprocess in time $\mathcal{O}(m)$. This will allow you to calculated $\left(\frac{1}{q}\right)^{\delta m - c}$ in constant time.

One might claim that our basic calculation need more than $\mathcal{O}(1)$ space and time as the numbers involved are much too big or too small. This is not the case as one can do all the above calculations using a floating point representation with the $\mathcal{O}(\log n)$ most significant bits and the results would not suffer at all. This is due to the fact that we only subtract $poly(n)$ numbers and compare between the different sub-sums. Moreover choosing the "wrong" value in one of the comparisons, might increase the goal function by merely the difference between the numbers we should have compared. This difference must be very small if we chose the wrong value, therefore not making the goal function value bigger than 1 and not hurting our algorithm.

We conclude the discussion with the following theorem.

Theorem 3: Let q be a prime power, m and k positive integers, and $\delta \in [0,1]$. If $k \leq (1 - H_q(\delta))\, m$, then it is possible to construct an $[m, k, \delta m]_q$-LC in time $\Theta\left(mq^k\right)$.

VI. Conclusion and Open Problems

We have presented a simple and intuitive construction of linear codes meeting the GV bound. Our construction is the most efficient known construction of such linear codes. We used our codes construction to construct explicitly, in $\Theta(rn \ln n)$ time, very good GT schemes of $\Theta\left(r^2 \ln n\right)$ tests. It would be interesting to study whether our linear code construction can be made more efficient, or whether it can be improved to construct better codes. While we managed to close the gap between the sizes of explicit and nonexplicit group testing schemes, the gap in the important generalization of selectors is still open; closing it is an interesting and important problem. We believe that other important special cases of group testing worth studying include the problem of minimizing the sets accumulative size rather than their number, and also, solely for algorithmic purposes—the case where the tests answers tell not only if there exists an element in the intersection or not, but rather, how many elements are there in it.

References

[1] The Center for Discrete Mathematics and Theoretical Computer Science (DIMACS), in *Proc. DIMACS Workshop on Combinator. Group Testing*, May 2006.

[2] R. Dorfman, "The detection of defective members of large populations," *Ann. Math. Statist.*, vol. 14, no. 4, pp. 436–440, 1943.

[3] E. Barillot, B. Lacroix, and D. Cohen, "Theoretical analysis of library screening using an n-dimensional pooling strategy," *Nucleic Acids Res.*, pp. 6241–6247, 1991.

[4] W. Bruno, D. Balding, E. Knill, D. Bruce, C. Whittaker, N. Dogget, R. Stalling, and D. Torney, "Design of efficient pooling experiments," *Genomics*, vol. 26, pp. 21–30, 1995.

[5] T. Berger, M. J. W. , and S. P. , "Maximally efficient two-stage screening," *Biometrics*, vol. 56, pp. 833–840(8), Sep. 2000.

[6] R. Clifford, K. Efremenko, E. Porat, and A. Rothschild, "k-mismatch with don't cares," in *Proc. ESA*, 2007, pp. 151–162.

[7] A. Amir, O. Kapah, and E. Porat, "Deterministic length reduction: Fast convolution in sparse data and applications," in *Proc. CPM*, 2007, pp. 183–194.

[8] G. Cormode and S. Muthukrishnan, "What's hot and what's not: Tracking most frequent items dynamically," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 249–278, 2005.

[9] B. Porat, E. Porat, and A. Rothschild, Pattern Matching in a Streaming Model.

[10] S. Muthukrishnan, "Some algorithmic problems and results in compressed sensing," in *Proc. 44th Allerton Conf. Commun., Contr. Comput.*, 2006.

[11] P. Indyk, "Explicit constructions for compressed sensing of sparse signals," in *Proc. SODA: ACM-SIAM Symp. Discrete Algorithms (A Conf. Theoret. Exper. Anal. Discr. Algorithms)*, 2008.

[12] A. C. Gilbert, M. J. Strauss, J. A. Tropp, and R. Vershynin, "One sketch for all: Fast algorithms for compressed sensing," in *Proc. 39th Ann. ACM Symp. Theory of Computing (STOC'07)*, New York, 2007, pp. 237–246.

[13] G. Cormode and S. Muthukrishnan, "Combinatorial algorithms for compressed sensing," in *Proc. SIROCCO*, 2006, pp. 280–294.
[14] M. A. Iwen, "A deterministic sub-linear time sparse fourier algorithm via non-adaptive compressed sensing methods," in *Proc. CoRR*, 2007, vol. abs/0708.1211.
[15] M. Sobel and P. Groll, "Group testing to eliminate efficiently all defectives in a binomial sample," *Bell Syst. Tech. J.*, vol. 38, pp. 1179–1252, 1959.
[16] W. Kautz and R. Singleton, "Nonrandom binary superimposed codes," *IEEE Trans. Inf. Theory*, vol. 10, pp. 363–377, 1964.
[17] C. Li, "A sequential method for screening experimental variables," *J. Amer. Stat. Assoc.*, vol. 57, pp. 455–477, 1962.
[18] J. Wolf, "Born again group testing: Multiaccess communications," *IEEE Trans. Inf. Theory*, vol. 31, no. 2, pp. 185–191, 1985.
[19] E. Hong and R. Ladner, "Group testing for image compression," in *Proc. Data Compress. Conf.*, 2000, pp. 3–12 [Online]. Available: citeseer.ist.psu.edu/article/hong00group.html
[20] A. Blass and Y. Gurevich, "Pairwise testing," *Bull. EATCS*, vol. 78, pp. 100–132, 2002.
[21] D. Cohen, S. Dalal, M. Fredman, and G. Patton, "The AETG system: An approach to testing based on combinatiorial design," *Software Eng.* vol. 23, no. 7, pp. 437–444, 1997 [Online]. Available: citeseer.ist.psu.edu/cohen97aetg.html
[22] P. A. Pevzner and R. J. Lipshutz, "Towards DNA sequencing chips," in *Proc. 19th Int. Symp. Math. Found. Comput. Sci. MFCS'94*, London, U.K., 1994, pp. 143–158.
[23] D. Du and F. Hwang, *Combinatorial Group Testing and its Applications*, ser. Ser. Appl. Math., 2nd ed. Singapore: World Scientific, 2000, vol. 12.
[24] M. Farach, S. Kannan, E. Knill, and S. Muthukrishnan, "Group testing problems with sequences in experimental molecular biology," in *Proc. Compress. Complex. Seq.*, 1997, p. 357.
[25] H. Ngo and D. Du, "A survey on combinatorial group testing algorithms with applications to DNA library screening," in *Proc. DIMACS Ser. Discrete Math. Theor. Comput. Sci. 55 (AMS 2000)*, 2000, pp. 171–182.
[26] D. J. Balding, W. J. Bruno, E. Knill, and D. C. Torney, "A comparative survey of non-adaptive pooling designs," *Inst. Math. Appl.*, vol. 81, pp. 133–+, 1996.
[27] S. Chaudhuri and J. Radhakrishnan, "Deterministic restrictions in circuit complexity," in *Proc. ACM Symp. Theory Comput. (STOC)*, 1996, pp. 30–36 [Online]. Available: citeseer.ist.psu.edu/chaudhuri96deterministic.html
[28] E. Gilbert, "A comparison of signalling alphabets," *Bell Syst. Tech. J.*, vol. 31, pp. 504–522, 1952.
[29] R. Varshamov, "Estimate of the number of signals in error correcting codes," *Doklady Akadamii Nauk*, vol. 117, pp. 739–741, 1957.
[30] R. A. Brualdi and V. Pless, "Greedy codes," *J. Comb. Theory, Ser. A*, vol. 64, no. 1, pp. 10–30, 1993.
[31] M. Chrobak, L. Gasieniec, and W. Rytter, "Fast broadcasting and gossiping in radio networks," in *Proc. IEEE Symp. Found. Comput. Sci.*, 2000, pp. 575–581 [Online]. Available: citeseer.ist.psu.edu/chrobak00fast.html
[32] Knill, "Lower bounds for identifying subset members with subset queries," in *Proc. SODA: ACM-SIAM Symp. Discrete Algorithms (A Conf. Theoret. Exper. Anal. Discrete Algorithms)*, 1995 [Online]. Available: citeseer.ist.psu.edu/knill95lower.html
[33] A. D. Bonis, L. Gasieniec, and U. Vaccaro, "Generalized framework for selectors with applications in optimal group testing," in *Proc. ICALP*, 2003, pp. 81–96.
[34] A. E. F. Clementi, A. Monti, and R. Silvestri, "Selective families, superimposed codes, and broadcasting on unknown radio networks," in *Proc. SODA*, 2001, pp. 709–718.
[35] A. D. Bonis and U. Vaccaro, "Constructions of generalized superimposed codes with applications to group testing and conflict resolution in multiple access channels," *Theor. Comput. Sci.*, vol. 306, no. 1–3, pp. 223–243, 2003.
[36] A. D. Bonis, L. Gasieniec, and U. Vaccaro, "Optimal two-stage algorithms for group testing problems," *SIAM J. Comput.*, vol. 34, no. 5, pp. 1253–1270, 2005.
[37] M. Farach, S. Kannan, E. Knill, and S. Muthukrishnan, IEEE Computer Soc., "Group testing problems with sequences in experimental molecular biology," in *Proc. Compress. Complex. Sequences (SEQUENCES'97)*, Washington, DC, 1997, p. 357.
[38] P. Damaschke, "Randomized group testing for mutually obscuring defectives," *Inf. Process. Lett.*, vol. 67, no. 3, pp. 131–135, 1998.
[39] A. D. Bonis and U. Vaccaro, "Improved algorithms for group testing with inhibitors," *Inf. Process. Lett.*, vol. 67, no. 2, pp. 57–64, 1998.
[40] T. Berger and V. I. Levenshtein, "Asymptotic efficiency of two-stage disjunctive testing," *IEEE Trans. Inf. Theory*, vol. 48, no. 7, pp. 1741–1749, 2002.
[41] T. Berger and V. I. Levenshtein, "Application of cover-free codes and combinatorial designs to two-stage testing," *Discr. Appl. Math.*, vol. 128, no. 1, pp. 11–26, 2003.
[42] M. A. J. , "Probabilistic nonadaptive and two-stage group testing with relatively small pools and DNA library screening," *J. Combinator. Optimiz.*, vol. 2, pp. 385–397(13), 1998.
[43] E. Knill, W. J. Bruno, and D. C. Torney, "Non-adaptive group testing in the presence of errors," *Discr. Appl. Math.*, vol. 88, no. 1–3, pp. 261–290, 1998.
[44] V. Guruswami and P. Indyk, "Linear-time list decoding in error free settings: (Extended abstract)," in *Proc. ICALP'04*, 2004, pp. 695–707.
[45] N. Alon, D. Moshkovitz, and S. Safra, "Algorithmic construction of sets for restrictions," *ACM Trans. Algorithms*, vol. 2, no. 2, pp. 153–177, 2006.
[46] J. Wozencraft, "Threshold decoding," *Personal Commununication [55] Section 2.5*, 1963.
[47] V. Goppa, "Codes associated with divisors," *Probl. Inf. Transmiss.*, vol. 13, no. 1, pp. 22–26, 1977.
[48] M. Tsfasman, S. Vladut, and T. Zink, "Modular curves, Shimura curves, and codes better then the Varshamov-Gilbert bound," *Math. Nachrichten*, vol. 109, pp. 21–28, 1982.
[49] K. Shum, "A Low-complexity construction of algebraic geometry codes better than the Gilbert-Varshamov bound," Ph.D. dissertation, Univ. Southern Calif., Los Angeles, Dec. 2000.
[50] K. Shum, I. Aleshnikov, P. Kumar, H. Stichtenoth, and V. Deolalikar, "A low-complexity algorithm for the construction of algebraic geometric codes better than the Gilbert-Varshamov bound," *IEEE Trans. Inf. Theory*, vol. 47, no. 6, pp. 2225–2241, Sep. 2001.
[51] M. Cheraghchi, A. Shokrollahi, and A. Wigderson, "Computational hardness and explicit constructions of error correcting codes," in *Proc. 44th Allerton Conf. Commun., Contr. Comput.*, 2006 [Online]. Available: http://www.csl.uiuc.edu/allerton/
[52] A. Clementi, A. Monti, and R. Silvestri, "Distributed broadcast in radio networks of unknown topology," *Theoret. Comput. Sci.*, vol. 302, no. 1–3, pp. 337–364, 2003.
[53] N. Alon and J. Spencer, *The Probabilistic Method*, 2nd ed. New York: Wiley, 2001.
[54] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *J. Amer. Statist. Assoc.*, vol. 58, no. 301, pp. 13–30, Mar. 1963.
[55] J. L. Massey, *Threshold Decoding*. Cambridge, MA: MIT Press, 1963.

E. Porat, biography not available at the time of publication.

A. Rothschild, biography not available at the time of publication.