The goal of this course is to introduce and study some tools and techniques that, for whatever reason, turn out to be important and often essential in (theoretical) computer science. It is a bit different from the kinds of courses that you have studied before. When you take a course on advanced algorithms you may expect to find out what is the fastest algorithm for computing, say, maximum flow; if this is too complicated, you would at least hope to discover some ingredients that would go towards building such an algorithm. If you study cryptography, you could begin with the goal of constructing a public-key encryption scheme and then learn about ideas that go into the construction and analysis of such schemes.

In contrast, in this course we will not aim to become experts at any specific topic. Instead we will try to introduce some ideas and techniques that may enable you to look at your favorite problem from a new perspective. Computer science is a vast field, but a few basic themes show up time and again, in places where you usually don't expect them to play a role. Some examples we'll see in this class are error-correcting codes, boolean functions, and expander graphs. There are many others like convex programming, metric geometry, additive combinatorics and so on. Many of these originate from outside fields like information engineering, operations research and various branches of mathematics. Why they turn out to be so important in computer science is somewhat mysterious and a source of great research projects.

Tolstoy famously said that all happy families are alike, but every unhappy one is unhappy in its own way. Research is the opposite: Experienced researchers have their own way of solving problems, but beginners often share the same suffering. You hear about a great research problem, think about it for a couple of days, and you get stuck. What should you do? There is rarely a magic formula that will give you the solution. What works better is a new perspective and and some tools you can try to apply, so that even if you are not making progress right away at least you may come up with some new insights.

In the first few lectures we will talk about coding theory. We will not do a comprehensive study of the subject but focus on some aspects of it that are commonly used in computer science. Although codes were invented in information theory, the ways they are used in computer science are often different than the ones envisioned by the information theorists. We will see some examples of this. But first, let us introduce a problem in theoretical computer science from the area of randomness extraction that on the surface has nothing to do with error-correcting codes. Yet knowing some coding theory can help us come up with better solutions.

1 Randomness extraction

Many computer applications depend on the use of randomness. It is used in physical systems as well as in low-level communication protocols. Randomness is especially important in cryptography; without it there would be no secrets and no security.

Yet coming up with random numbers is not a trivial task. Computers have access to a variety of processes that look random, but it is not always clear how to take advantage of them. For instance

consider the following sequence of data

$$19\ 15\ 15\ 17\ 18\ 19\ 22\ 20\ 15\ 13$$

These are the mean temperatures measured in Hong Kong in the first ten days of December last year. These temperatures certainly look random, but they also contain a large amount of non-random information – they are certainly correlated to one another and close to the annual mean winter temperature for the city.

How should we go about "extracting" the hidden randomness out of this data? Let's make things simple and suppose we want just one random bit, something that is 0 or 1 but we expect it to be unbiased. One idea could be to compute the sum modulo two of the data

$$19 + 15 + 15 + 17 + 18 + 19 + 22 + 20 + 15 + 13 \mod 2 = 1.$$

Now let's look at another data sequence

$$4\ 6\ 5\ 6\ 3\ 3\ 6\ 9\ 4\ 0$$

which records the number of acquaintances of each person among a party of ten people. Surely this data also looks random, so if we want to get a random bit out of it we can again try to compute

$$4 + 6 + 5 + 6 + 3 + 3 + 6 + 9 + 4 + 0 \mod 2 = 0.$$

Now repeat this experiment in different groups of people and you will discover that no matter which group you are looking at you will always get the answer 0, which is not random at all! There is a simple explanation for this: Every pair of acquaintances was counted twice, so it is not surprise the sum is always an even number.

So the "randomness extraction" procedure that gave us a random bit out of a sequence of temperatures did not work for the sequence of acquaintance numbers. As computer scientists, we would like to have a universal procedure that we can apply to any data, provided the data contains some randomness. This is the problem of randomness extraction. Let's come up with a model for it and see what we can do.

Let us represent the random data as a sequence in $\{0,1\}^n$ (as usual in computer science we work with binary representations). There must be some uncertainty in the choice of the sequence, for otherwise there work be no randomness to extract from it. To keep our model simple, we will view this uncertainty as a subset $S \subseteq \{0,1\}^n$ of values that X could be taking. In the acquaintances example, S could comprise all the degree sequences that we may expect to observe at a random 10-person party. What we want to do now is extract some randomness out of the sequence; but we don't really know what the set S is, as this depends on the specifics of the data collection procedure. All we know is that since the sequence has some randomness in it, S should not be too small.

Let's think of the extraction procedure as a function $f\colon \{0,1\}^n \to \{0,1\}$ which maps our data to a supposedly random bit. To begin with let us make a very weak requirement on randomness – just ask that sometimes H takes the value 0 and sometimes it takes the value 1. This suggest the following problem:

> Come up with a function $f\colon \{0,1\}^n \to \{0,1\}$ such that for every subset S $\{0,1\}^n$ of size at least K, f does not take a constant value on S (i.e. there exist $x_0, x_1 \in S$ such that $H(x_0) = 0$ and $H(x_1) = 1$).

A moment's thought shows that this is impossible unless K is very large: No matter what H is, let S be the larger one of the sets $\{x\colon H(x) = 0\}$ and $\{x\colon H(x) = 1\}$. Then S has size 2^{n-1} and H is constant on S. So S must have size at least 2^{n-1}, which is not very interesting.

Two-source hitters This is bad news, so we need to make some additional assumptions about our data if we are to extract any randomness out of it. One possibility is to make some additional assumptions on what the set S should look like. But today we will look at something different.

In the scenario we looked at the function Ext had access to one random sequence. But suppose instead that it had access to two such independent data sets – say a sequence of temperatures as well as a sequence of acquaintance numbers. Can we extract some randomness out of both these sequences?

In its simplest incarnation, we model the data as strings in $\{0,1\}^n$ and the uncertainty as a pair of subsets $S, T \subseteq \{0,1\}^n$, each of size at least K. Now each pair of data items $x \in S, y \in T$ is a possible outcome. Our goal is to "extract" a random bit without knowing what S and T are. As before, we will for now settle with the simpler requirement that both the values 0 and 1 occur as possible outcomes at least once. The object we are looking at is called a two-source hitter in computer science (or bipartite Ramsey graph in mathematics).

Definition 1. An (N, K) two-source hitter is a function $f\colon [N] \times [N] \to \{0,1\}$ such that for every pair of subsets $S, T \subseteq [N]$ of size K, f is not constant on $S \times T$ (i.e. there exist $(x_0, y_0), (x_1, y_1) \in S \times T$ such that $f(x_0, y_0) = 0$ and $f(x_1, y_1) = 1$).

Here $[N]$ denotes the set $\{1, \ldots, N\}$. For the application we have in mind we set $N = 2^n$ and we identify the set $[N]$ with $\{0,1\}^n$. We just saw that "one-source hitters" do not even exist; how about two-source hitters?

Obtaining two-source hitters The existence of two-source hitters was shown by Erdös in 1946 in one of the earliest uses of the probabilistic method. Erdös showed that for a suitable choice of $K = K(N)$, if the function f is chosen uniformly at random, then the probability that f is a two-source hitter is strictly greater than zero. So at least one f must be a two-source hitter.

A random function f is chosen by picking each value $f(x, y)$ uniformly and independently at random from $\{0,1\}$. Instead of lower bounding the probability that f is a two-source hitter, it will be easier to upper bound the probability that a random f is *not* a two-source hitter. Unwinding the definition, this means that there exists a pair of sets S, T of size K and a number $b \in \{0,1\}$ such that $f(x, y) = b$ for all K^2 pairs $x \in S, y \in T$. We can now write

$$\begin{aligned}\Pr[f \text{ is not a two-source hitter}] &= \Pr[\exists S, T, b\colon f(x,y) = b \text{ for all } x \in S, y \in T] \\ &\leq \sum_{S,T,b} \Pr[f(x,y) = b \text{ for all } x \in S, y \in T] \\ &= \sum_{S,T,b} 2^{-K^2} = 2\binom{N}{K}\binom{N}{K}2^{-K^2}.\end{aligned}$$

The inequality follows from the union bound. To show that two-source hitters exist, we need that this probability be strictly less than one. We use the basic estimate $\binom{N}{K} \leq N^K$ to obtain that

$$\Pr[f \text{ is not a two-source hitter}] \leq 2^{2K \log N - K^2 + 1} < 1$$

as long as $K \geq 2 \log N + 1$. This bound is tight up to the factor 2.

So now we know that $(N, K = 2 \log N + 1)$ two-source hitters exist. But how do we get our hands on one? The simplest thing to do is to go over all possibilities and check which one works. This means going over all possible functions $f\colon [N] \times [N] \to \{0,1\}$ until we come up with the one that has the desired hitting property. There are $2^{N^2} = 2^{2^{2n}}$ such functions, which is a lot of time to waste looking for a hitter. Can we do better?

2 Error-correcting codes

Error-correcting codes concern a simple setting. Alice wants to send Bob a message, but some of the symbols may be corrupted in the transmission. How should Alice "encode" her message to ensure that Bob can recover it despite the errors? This scenario has been studied in detail for more than fifty years, and despite the simplicity many basic questions remain unanswered. Fortunately there have been many insights throughout the years, and we will see how these are relevant for computer science.

We model this problem as follows. Alice wants to send Bob a k-bit string, namely an unknown message x coming from the space $\{0,1\}^k$. To this end Alice sends a longer string $C(x) \in \{0,1\}^n$, which we call the *codeword* of x. When Bob receives $C(x)$ part of it may have been corrupted and his goal is to recover x given this *corrupted codeword* c.

Unless we put some restriction on how corruptions happen, this is certainly impossible even if Alice wants to send just one bit of information. In many applications it makes sense to consider probabilistic noise: The string $C(x)$ is corrupted in some known stochastic manner that models the effect of the environment on the transmission. The simplest such corruption model, proposed by Shannon, changes every bit of $C(x)$ independently of all the others with some small probability p.

Even though probabilistic noise models are adequate for engineering, in computer science applications it is typically more useful to consider noise that is chosen adversarially, but limit the amount of noise. The corrupted codeword c will be obtained from $C(x)$ by changing at most r different coordinates of $C(x)$.

Now fix the parameters n, k, and r and a candidate code C. Recall Bob's objective: No matter which x Alice wants to send to Bob, given a corrupted codeword c that differs from $C(x)$ in at most r positions, Bob should be able to recover x uniquely from c. Obviously, this can happen if and only if every two codewords are at distance at least $2r + 1$. We call this quantity the *(minimum) distance* of the code C.

Definition 2. A *code* with block length n, message length k, and distance d is a function $C\colon \{0,1\}^k \to \{0,1\}^n$ such that for every $x \neq x'$, $C(x)$ and $C(x')$ differ in at least d positions (i.e. have distance at least d).

(We will call such codes $[n, k, d]$ codes.) Alternatively, we can view an the code C as the set of 2^k points $\{C(x)\colon x \in \{0,1\}^k\}$; then we require that the distance between any pair of points in this set is at least d. Thus a code can be either viewed as a function (from messages to codewords) or a set (of codewords).

The most fundamental question about codes asks which values of n, k, and d can be achieved simultaneously. This answer is not completely known. Let us start with some examples first. The

parity check code

$$x_1, \ldots, x_{n-1}, x_1 \oplus \cdots \oplus x_{n-1}$$

is an $[n, n-1, 2]$ code. It is easy to check that the following construction by Hamming is a $[7, 4, 3]$ code:

$$x_1, x_2, x_3, x_4, x_1 \oplus x_2 \oplus x_3, x_1 \oplus x_2 \oplus x_4, x_1 \oplus x_3 \oplus x_4.$$

To get a better feel about which values of n, k, and d are achievable, let us first identify some natural limitations. In many computer science applications it is natural to view k and d as functions of n and study these parameters asymptotically as n goes to infinity. Two particular regions will be of interest: When d is small (and we try to make k as large as possible) and when k is small (and we try to make d large). Today we focus on small values of d.

Bounds on codes at small distances If we are encoding 2^k different messages in $\{0,1\}^n$, by the pigeonhole principle at least two of the codewords will agree on their first $k-1$ coordinates. Since this pair of codewords must be at distance d we must have $n - k + 1 \geq d$, or $d \leq n - k + 1$. Thus the parity check code is optimal for distance 2.

What about somewhat larger distances? The ball of radius r centered at $x \in \{0,1\}^n$ is the set of all points in $\{0,1\}^n$ within distance r of x. Then the balls of radius $\lfloor (d-1)/2 \rfloor$ centered at the 2^k different codewords cannot overlap, so we must have $2^k V_n(\lfloor (d-1)/2 \rfloor) \leq 2^n$, where $V_n(r)$ is the volume of the ball of radius r. It turns out that no constructions match this bound for $n \geq 24$ and $d \geq 3$.

When d is constant, $V_n(\lfloor (d-1)/2 \rfloor) = \Omega(n^{\lfloor (d-1)/2 \rfloor})$ and we get the upper bound

$$\left\lfloor \frac{d-1}{2} \right\rfloor \leq \frac{n - k + O(1)}{\log n} \tag{1}$$

which is tighter than the bound $d \leq n-k+1$ when n and k are sufficiently large. Can we construct codes that match this bound asymptotically?

Instead of doing this directly, let us first show how to achieve the impossibly tight bound $d = n-k+1$ but in a more relaxed setting. To do so we will introduce a more general setting where the message and more importantly codeword symbols come not from $\{0,1\}$ but from a larger alphabet.

3 Reed-Solomon codes

Just like in the binary case, an $[n, k, d]_\Sigma$ code over alphabet Σ is a function $C \colon \Sigma^k \to \Sigma^n$ such that every two images $C(a), C(a')$, where $a \neq a'$, differ in at least d coordinates. We will particularly be interested in the case when Σ is a finite field $\mathbb{F}$. For now you can think of $\mathbb{F}$ as the prime field $\{0, \ldots, p-1\}$ where p is a prime number with the operations of addition and multiplication modulo p, where $p \geq n$.

Let $S = (s_1, \ldots, s_n)$ be an ordered subset of $\mathbb{F}$. We will view the message $a = (a_0, \ldots, a_{k-1})$ as the coefficients of a polynomial $p_a(x) = a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}$. The Reed-Solomon encoding of a (over S) is the vector $RS(a) = (f_a(s_1), \ldots, f_a(s_n))$ in $\mathbb{F}^n$.

We claim that the Reed-Solomon code has distance $n - k + 1$. To do this first we observe that the code is linear: $RS(a + a') = RS(a) + RS(a')$. (In particular, $RS(0) = 0$.) For linear codes, the

distance of the code equals the hamming weight of the shortest nonzero codeword: if two codewords c and c' are at distance d, then $c-c'$ has hamming weight d; in the other direction, if c has hamming weight d, then the distance between c and the zero codeword is at most d.

So to show that $d \geq n-k+1$, we must argue that for $a \neq 0$ at least $n-k+1$ of the values $f_a(s_i)$ are non zero, or equivalently that at most $k-1$ of the $f_a(s_i)$ are zero. This is a general property of polynomials over finite fields: If f is a nonzero polynomial and $f(s)=0$, then we can write $f(x)$ as $x-s$ times a polynomial of strictly lower degree. If f has k zeros, then $f(x)$ equals $(x-s_1)\dots(x-s_k)$ times some polynomial, so f has degree at least k, a contradiction.

For this construction it was essential that the field $\mathbb{F}$ has at least size n, so that we can choose n distinct points where to evaluate the polynomials. What can we do over smaller alphabets? One simple thing we can do is to represent the field elements as bit strings: For example if $p=5$ we can represent the field elements by the strings $0,1,2,3,4$ by the bit strings $000, 001, 010, 011, 100$, respectively. But when p is prime this representation does not look very efficient as some of the patterns are never used. For this reason – and more important ones that will become apparent later – it will be more convenient to work over fields whose size is a power of two.

Extension field arithmetic Let's do a short review of the arithmetic of finite field $\mathbb{F}_{2^m}$, $m \geq 1$. A good example to keep in mind is the field $\mathbb{F}_4 = \mathbb{F}_{2^2}$. This field has four elements, which we call 0, 1, α, and $1+\alpha$. The elements 0 and 1 come from $\mathbb{F}_2$. Arithmetic is the same in $\mathbb{F}_2$, except that $\alpha^2+\alpha+1=0$; so every time we see α^2 we apply the substitution $\alpha^2 = \alpha+1$. (If you like fancy algebra you can say that $\mathbb{F}_4$ is isomorphic to the quotient field $\mathbb{F}_2[\alpha]/(\alpha^2+\alpha+1)$.) In particular, this gives $\alpha^2 = 1+\alpha$ and $\alpha^3 = 1$, so the sequence $(\alpha, \alpha^2, \alpha^3)$ covers every nonzero element of $\mathbb{F}_4$ exactly once.

In general, we represent the elements in $\mathbb{F}_{2^m}$ as formal linear combinations $x_0+x_1\alpha+\dots+x_{m-1}\alpha^{m-1}$, where $x_0,\dots,x_{m-1}$ take values in $\mathbb{F}_2$ and $q(\alpha)=0$ for some (primitive) polynomial of degree m. Then α is a *generator* of $\mathbb{F}_{2^m}$: The sequence of values $\alpha, \alpha^2, \dots, \alpha^{2^m-1}$ covers all nonzero elements of $\mathbb{F}_{2^m}$ exactly once. This forces $\alpha^{2^m-1} = 1$ and gives a multiplicative inverse for every nonzero element: The inverse of α^i is α^{2^m-i-1}.

Binary codes Now let n be a power of two and consider the $[n,k,d]_{\mathbb{F}}$ Reed-Solomon code over $\mathbb{F}_n$. To obtain a binary code B, represent every symbol $x = x_0 + x_1\alpha + \dots + x_{m-1}\alpha^{m-1}$ in the Reed-Solomon encoding by the binary string $(x_0, x_1, \dots, x_{m-1})$. Since each element of $\mathbb{F}$ represents $\log n$ bits, B has message length $k \log n$ and block length $n \log n$. Moreover the distance of B is at least d: Every disagreement between codewords in the Reed-Solomon encoding spawns at least one disagreement in B.

In other words, B is a $[n' = n\log n, k' = k\log n, d' = d]$ code. Since $d = n-k+1$, we get that

$$d' - 1 \geq \frac{n'-k'}{\log n'}$$

which approaches the bound (1) by factor of two. We will soon see how to improve upon this bound; but before we do that let us see what such codes have to do with two-source hitters. First we have to discuss d-wise independent probability distributions.

4 The parity check matrix and bounded independence

A sequence of correlated random variables $Z_1, \ldots, Z_n$ taking values in $\{0,1\}$ is *t-wise independent* if for every subset $\{i_1, \ldots, i_t\} \subseteq [n]$, the random variables $Z_{i_1}, \ldots, Z_{i_t}$ are independent. Even $(n-1)$-wise independence is a weaker property than full independence: Choose $Z_1, \ldots, Z_{n-1}$ uniformly and independently from $\{0,1\}$ and set $Z_n = Z_1 \oplus \cdots \oplus Z_{n-1}$. Then it is easy to see that any subset of $n-1$ of the Z_i will be independent, although all n of them are not as they satisfy the relation $Z_1 \oplus \cdots \oplus Z_n = 0$.

This example looks a lot like the parity check code of distance two and the connection is not incidental. We will give a general way to obtain d-wise independent distributions from *linear* codes of distance d. A code $C\colon \mathbb{F}_2^k \to \mathbb{F}_2^n$ is linear if the map C is linear. There are two ways to describe such codes. One is by giving the map C; the other is by giving a basis of linear constraints that all codewords in C must satisfy. For example, the parity check code which was given by the map

$$x_1, \ldots, x_{n-1} \to x_1, \ldots, x_{n-1}, x_1 \oplus \cdots \oplus x_{n-1}$$

can also be represented by the linear constraint

$$y_1 \oplus \cdots \oplus y_n = 0.$$

All the codewords satisfy the linear constraint, and conversely all vectors that satisfy the linear constraint are codewords.

Similarly, the $[7,4,3]$ Hamming code can be described by the following three (linearly independent) constraints:

$$\begin{aligned} y_1 \oplus y_2 \oplus y_3 \oplus y_5 &= 0 \\ y_1 \oplus y_2 \oplus y_4 \oplus y_6 &= 0 \\ y_1 \oplus y_3 \oplus y_4 \oplus y_7 &= 0. \end{aligned}$$

In general, a code $C\colon \{0,1\}^k \to \{0,1\}^n$ (of positive distance) can be described by $n-k$ linearly independent constraints. We can write these constraints as an $(n-k) \times n$ matrix H called a *parity check matrix* of C. Then the codewords in C are those vectors $c \in \{0,1\}^n$ that satisfy $Hc = 0$.

Theorem 3. *Suppose C is a linear code of distance $d \geq 2$ and let H be a parity check matrix of C. Let u be a uniformly random row vector in $\{0,1\}^{n-k}$. Then the bits of $Y = uH$ are uniformly random and $(d-1)$-wise independent.*

Proof. Let z be any vector of hamming weight less than d. Then Hz is nonzero, and so the expression $\mathrm{E}[(-1)^{uHz}]$ must equal zero. On the other hand,

$$0 = \mathrm{E}[(-1)^{uHz}] = \mathrm{E}\Big[\prod_{i\colon z_i = 1} (-1)^{Y_i}\Big].$$

This condition is sufficient (and in fact equivalent) to saying that $Y_1, \ldots, Y_n$ are d-wise independent because

$$\begin{aligned} \Pr[Y_{i_1} = a_1 \cdots \text{ and } \cdots Y_{i_{d-1}} = a_{d-1}] &= \mathrm{E}\Big[\prod_{j=1}^{d-1} \frac{1 - (-1)^{Y_{i_j} + a_j}}{2}\Big] \\ &= 2^{-(d-1)} \sum_{S \subseteq [d-1]} \mathrm{E}\Big[\prod_{j \in S} (-1)^{j + Y_{i_j} + a_j}\Big] = 2^{-(d-1)} \end{aligned}$$

because the only surviving term in the product is the term for $S = \varnothing$. □

Improved construction of two-source hitters In our analysis of two-source hitters, we argued that a random function f is a two-source hitter with probability strictly greater than zero. The values of f were chosen uniformly and independently of one another. Looking back at the proof, we can see the only place independence is used was for the equality

$$\Pr[f(x,y) = b \text{ for all } x \in S, y \in T] = 2^{-K^2}$$

since the values $f(x,y)$ as x ranges over S and y ranges over T are independent of one another. But notice that the same equality holds if the values $f(x,y)$ are not fully independent but merely K^2-wise independent!

By Theorem 3, a sequence of N^2 bits that are K^2-wise independent can be obtained from the distribution uH, where H is the parity check matrix of an code of block length N^2 and distance K^2. If we plug in the code B from the previous section, we get that the dimension of u is at most $(1 + o(1))K^2 \log N$.

What is the advantage of using this distribution of limited independence? By the same analysis as before we are guaranteed that one of the functions f is a two-source hitter. Before we had to sift through 2^{N^2} such functions to find a good one; but now the number of functions we have to look at is determined by the number of possible choices for u. Since u has dimension $(1 + o(1))K^2 \log N$, we only need to consider $2^{(1+o(1))K^2 \log N}$ choices of u. For $N = 2^n$ and $K \approx 2 \log N$ this is a vast improvement: We can find a hitter in singly exponential instead of doubly exponential time.

5 BCH codes

We now show how to improve the distance of B by a factor of two (when d is small) by another construction of codes called BCH codes. To do so it will be useful to first look at the Reed-Solomon code (for a suitable set S) from the parity check perspective.

Now let n be of the form $2^m - 1$ and choose S be the set of all nonzero values in $\mathbb{F}_{2^m}$. We can write $S = \{1, \alpha, \alpha, \ldots, \alpha^{n-1}\}$, where α is a generator for $\mathbb{F}_{2^m}$. Consider the message $a = (0, \ldots, 1, \ldots, 0)$, where the only nonzero entry occurs in the ith coordinate, $0 \le i < k$. The corresponding polynomial is $f_a(x) = x^i$ and the Reed-Solomon encoding is the vector

$$(c_0, \ldots, c_{n-1}) = (1, \alpha^i, \ldots, \alpha^{(n-1)i})$$

which for every $1 \le j \le n - k$ satisfies the constraint

$$c_0 + c_1\alpha^j + \cdots + c_{n-1}\alpha^{(n-1)j} = 0. \tag{2}$$

This is because $1 + x + \cdots + x^{n-1} = 0$ for every $x \neq 0, 1$ in $\mathbb{F}_{2^m}$. It turns out these constraints are linearly independent, and since there are $n - k$ of them they span the parity check matrix of the Reed-Solomon code.

Recall that the Reed-Solomon code is optimal; the only issue is that it works over $\mathbb{F}_{2^m}$ instead of $\mathbb{F}_2$. The BCH code is defined by enforcing the constraints (2) when $c_0, \ldots, c_{n-1}$ comes from $\mathbb{F}_2$. Then we can view each linear constraint in (2) as a collection of n linear constraints over $\mathbb{F}_2$: The constraint $\ell_0(c) + \ell_1(c)\alpha + \cdots + \ell_{n-1}(c)\alpha^{n-1}$ over $\mathbb{F}_{2^m}$ gives rise to the $m = \log(n + 1)$ constraints $\ell_0(c) = \cdots = \ell_{n-1}(c) = 0$ over $\mathbb{F}_2$.

Definition 4. The (binary) BCH code with parameters $n = 2^m - 1$ and d consists of all codewords $c \in \mathbb{F}_{2^n}$ such that

$$c_0 + c_1\alpha^j + \cdots + c_{n-1}\alpha^{(n-1)j} = 0$$

for $1 \leq j < d$, where α is a generator of $\mathbb{F}_n$.

The distance of this BCH code is at least d, since every codeword in the BCH code also satisfies all the constraints (2) for message length $n-d+1$ (as well as the additional constraints $c_i \in \mathbb{F}_2$). To get a lower bound on the message length k, we upper bound the dimension of the parity check matrix of the BCH code. There are $d-1$ constraints over $\mathbb{F}_n$, which gives $(d-1)\log(n+1)$ constraints over $\mathbb{F}_2$, from where $k \geq (d-1)\log(n+1)$. This does not improve our bound on B. The improvement will come from the fact that these constraints are not linearly independent; in fact about half of them are redundant.

To see this, let $c(x)$ be the polynomial $c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$. It is not hard to check that $c(x^2)$ and $c(x)^2$ is the same polynomial, so all the constraints where j is odd are redundant. This leaves out at most $\lceil (d-1)/2 \rceil \log(n+1)$ non-redundant constraints that describe the code, and so the message length is at most $n - \lceil (d-1)/2 \rceil \log(n+1)$.

By Theorem 3, the BCH code can be used to construct a t-wise independent space of random variables $Y_1, \ldots, Y_n$, $n = 2^m - 1$ of size $(n+1)^{\lceil t/2 \rceil}$. This bound is almost tight: For constant t, a t-wise independent set must be of size $\Omega(n^{\lfloor t/2 \rfloor})$.