6.897 Algorithmic Introduction to Coding Theory	December 5, 2001
Lecture 21	
Lecturer: Madhu Sudan	*Scribe: Prahladh Harsha*

In today's lecture, we shall talk about the applications of coding theory to computer science, with specific examples in the following contexts.

- Probabilistic Communication Complexity
- Secret Sharing
- Pseudo-randomness (limited independence)

To say the very least, coding is one of the most important aspects while building a computer. To merely store data or even compute the identity function, coding is essential. Coding is even more relevant in the context when the basic building blocks of computers are unreliable. There is a whole field devoted to "reliable computing from unreliable components" [von Neumann, Shannon, Moore, Elias, ..., Spielman]. It is a different question as to how realistic these models of computations are and we shall not venture into that debate in this course.

1 Coding Theory in Algorithms/Data Structures

Coding comes up in several places in Algorithms and Data Structures. Instead of going into any detailed applications of coding in Algorithms/data structures for want of time, we shall present two fun examples.
Bounded Size Dictionary: Consider a l-bounded dictionary containing elements from an universe $U = \{1, \ldots, n\}$, that supports the following operations.

- $\textsc{Insert}(x, S)$ (conditioned on $x \notin S$)
- $\textsc{Delete}(x, S)$ (conditioned on $x \in S$)
- $\textsc{Members}(S)$ This function returns the list of all elements of S if $|S| \leq l$ and otherwise signals that "$|S| > l$".

The conditional statements are assumed to be promises. In other words, there is never a call of the function $\textsc{Insert}(x, S)$ when $x \in S$ and similarly there is no call of $\textsc{Delete}(x, S)$ when $x \notin S$.

Ex: Build an implementation of the l-bounded dictionary such that each of the operations can be performed in $O(l \log n)$ time.
(Hint: Use coding theory)
Peter Winkler's hat problem: Our next example is the hat problem, popularized by Peter Winkler.
Imagine an empire of arbitrary rules and justice, in which n prisoners are to be executed. The emperor, driven by his whims and fancies, is "kind" enough to spare all the n prisoners if they pass the following test of hats. A red or black hat is randomly placed on each of the prisoner's head. The prisoners can look at the color of the hats of all the other prisoners except their own. The prisoners have no means of communicating amongst each other. Each prisoner could either guess the color of his hat or choose not to guess. If all the prisoners elect not to guess or even if one of the prisoners wrongly guesses his hat color, then all the prisoners are executed. Otherwise, the emperor excuses all of them. Does there exist a strategy such that the prisoners can minimize the likelihood of their being executed. (Hint: Use Hamming codes]

2 Coding Theory in Cryptography/Complexity Theory

The remaining of today's lecture will be devoted to the application of coding theory in three specific instances in cryptography/complexity theory.

2.1 Communication Complexity

Communication Complexity is the field, introduced by Yao[3], which discusses the amount of information that two parties need to communicate with each other for achieving a specific goal.

The model of communication is as follows: Alice (A) and Bob (B) each has a k-bit string x and y respectively, however neither knows the other's string. They both wish to compute a function $f(x, y)$. What is the minimum number of bits they need to exchange so that at the end of the communication they both know the value $f(x, y)$?

For simplicity, let us assume that f is a boolean function. (i.e., $f : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}$.)

Alice (A)	$\vert$	**Bob (B)**
$x \in \{0,1\}^k$		$y \in \{0,1\}^k$
	$f : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}$	
	$\longleftarrow - - -$	
	$- - - \longrightarrow$	
	$\longleftarrow - - -$	
	$- - - \longrightarrow$	
	$\vdots$	
	$\vdots$	
$f(x, y)$		$f(x, y)$

Communication Complexity of f, denoted by $\mathsf{CC}(f)$, is loosely defined as follows.

$$\mathsf{CC}(f) = \min_{\text{strategies } (A \leftrightarrow B)} \left[\max_{x,y \in \{0,1\}^k} \big(\text{ number of bits exchanged to determine } f(x, y) \big) \right]$$

This definition can be made formal, but we shall not attempt to do so here.

In this model, we are only interested in the communication complexity of f and not the computational complexity of f. (i.e., we can assume that the players A and B have unlimited computational power.)

A very naive strategy that works for all f is as follows: Alice sends the entire string x to Bob, who then computes $f(x,y)$ and sends the answer to Alice. This, demonstrates that $\mathsf{CC}(f) \leq k + 1$ for all functions f. However, we expect to do much better at least for certain functions f. For example, if $f = \oplus$, the parity function[1], then it is sufficient for Alice to send just the parity of the bits of x (i.e., $\oplus x_i$) instead of all of x. Thus, $\mathsf{CC}(\oplus) \leq 2$. However, some functions do have a very large communication complexity. Define the equality function, EQ, as follows.

$$\mathsf{EQ}(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

Yao [3] demonstrated that $\mathsf{CC}(\mathsf{EQ}) = k+1$, i.e., no matter how good a strategy Alice and Bob adopt, they need to exchange at least $k + 1$ bits to compute $\mathsf{EQ}(x, y)$.

Now, suppose, we alter the model of communication and let both players adopt probabilistic strategies instead of just deterministic ones. In other words, Alice and Bob can also toss random coins with the requirement that they compute the right answer with probability at least 3/4. This is the area of "Probabilistic Communication Complexity" and the corresponding complexity measure is $\mathsf{PCC}(f)$, the probabilistic communication complexity of f. $\mathsf{PCC}(f)$ is defined similarly to $\mathsf{CC}(f)$. The natural question that arises then is "Does probabilistic communication really help?" In other words, is $\mathsf{PCC}(f)$ any smaller than $\mathsf{CC}(f)$ for some f? Yes, in fact, $\mathsf{PCC}(f)$ can be exponentially smaller than $\mathsf{CC}(f)$ as seen from the following instance.

[1] The parity function $\oplus(x, y) = \bigoplus_{i=1}^{k} x_i \oplus \bigoplus_{i=1}^{k} y_i$, computes the parity of all the bits of x and y.

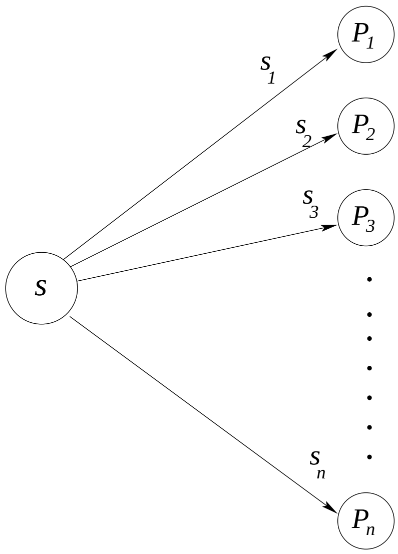

Figure 1: Secret Sharing

Theorem 1 $\mathsf{PCC}(\mathsf{EQ}) = O(\log k)$

The first and natural approach to proving this theorem would be the following probabilistic strategy. Alice chooses a random index $i \in_R \{0,1\}^k$ and sends the pair (i, x_i) to Bob. Bob checks if $x_i = y_i$ and answers accordingly to Alice. However, if x and y differ in just one bit, this strategy succeeds with probability at most $1/k$. In order to increase the number of bits x and y differ in, Alice and Bob would first need to encode the strings according to some code with a large distance and then choose a random position in the encoded string. This increases the probability of the chosen bit being different.

Proof Let $\mathcal{C} = [n,k,d]_q$-code. To start with, Alice and Bob encode x and y according to $\mathcal{C}$ to obtain $\mathsf{Enc}(x)$ and $\mathsf{Enc}(y)$ respectively. Alice then chooses a random $i \in_R \{1,\ldots,n\}$ and sends the pair $(i, (\mathsf{Enc}(x))_i)$ to Bob. Bob then checks if $(\mathsf{Enc}(x))_i = (\mathsf{Enc}(y))_i$ and replies accordingly. Thus, the communication complexity $\mathsf{PCC}(\mathsf{EQ})$ is at most $\log n + \log q$ and they obtain the right answer with probability d/q. Thus, if $\mathcal{C}$ were a RS $= [4k,k,3k]_{4k}$-code, then we have the right answer with probability at least $3/4$ and $\mathsf{PCC}(\mathsf{EQ}) = O(\log k)$. ■

The original proof of this theorem used the Chinese Remainder Theorem (CRT) codes, instead of the Reed-Solomon RS-codes used above. Note that if Alice and Bob shared a common source of randomness, then they don't need to communicate the index i. In that case, they would use a code with constant sized alphabet and large distance, to obtain $\mathsf{PCC}(\mathsf{EQ}) = O(1)$.

2.2 Secret Sharing

The basic idea in secret sharing is to divide a secret s into secret-shares $s_1, \ldots, s_n$ and distribute the shares to different players $P_1, \ldots, P_n$ so that no small subset of the players can get together to recover the secret, however a large subset of players should be able to recover the secret from the shares they collectively possess. More formally, an (l,m)-secret-sharing scheme is a scheme to distribute a secret s from a data set D among n players $P_1, \ldots, P_n$ such that player P_i gets the secret-share s_i. Moreover, the scheme should satisfy the following two properties.

- $\forall S \subseteq \{1, \ldots, n\}, |S| \leq l$, players $\{P_i\}_{i \in S}$ should not be able to compute s from the shares they possess.
- $\forall S \subseteq \{1, \ldots, n\}, |S| \geq l$, players $\{P_i\}_{i \in S}$ should be able to compute s from the shares they possess.

Shamir proposed this model in 1979 [2] and solved it completely providing the following implementation of a $(l, l+1)$-secret-sharing scheme.

Shamir's $(l, l+1)$-secret-sharing scheme

Input: Secret $s \in \mathbb{F}_q, q > n$. (the data set D here is $\mathbb{F}_q$)

- Choose a random polynomial $p \in \mathbb{F}_q[x]$ of degree at most l satisfying $p(0) = s$.
- Set $s_i = (x_i, p_i(x_i))$, where $x_1, \ldots, x_n$ are n distinct non-zero elements of $\mathbb{F}_q$ and send player P_i the share s_i.

Clearly, this scheme satisfies the following two properties.

- Any subset of at most l secret-shares leave the constant term of the polynomial p free. (i.e., with just l secret-shares $(x_i, p(x_i))$, the constant term of p could as well be any element of $\mathbb{F}_q$ equally likely.)
- Given $(l+1)$ shares, by interpolation we can obtain the polynomial p and thus the secret $s = p(0)$.

We shall now view the above scheme from a coding theory perspective. This "coding theory" treatment of secret sharing was first given by [4].

Let $\mathcal{C}$ be any $[n+1, k, d]_q$-code. Based on $\mathcal{C}$, we shall design an (l, m)-secret-sharing scheme for the data set $D = \mathbb{F}_q$ for any $m \geq n - d + 2$ and $l \leq d^\perp - 2$ where $d^\perp$ is the distance of the dual code $\mathcal{C}^\perp$.

Code $\mathcal{C}$'s (l, m)-secret-sharing scheme

Input: Secret $s \in \mathbb{F}_q$.

- Choose a random codeword $c = c_0 c_1 \ldots c_n \in_R \mathcal{C}$ subject to the constraint that $c_0 = s$.
- Set $s_i = c_i$ and send player P_i the share s_i.

This scheme satisfies the two properties of the secret sharing model due to the following reasons.

- From any subset $\{c_i\}$ of at least m shares, we can perform erasure decoding and obtain the entire codeword c as long as the number of erasures is less than d. (i.e., $n + 1 - m < d$) Thus, for any m such that $m \geq n - d + 2$, the secret can be obtained from m shares by erasure decoding.
- The other property of secret sharing requires that no subset of l or less players should be able to compute the secret. In other words, for any $S \subseteq \{0, \ldots, n\}, |S| \leq l$, we should not be able to compute c_0 from $\{c_i\}_{i \in S}$. What are the relations among the c_i's of a codeword that lets us compute c_0 from the rest? The only relations among the c_i's are linear relations given by the parity check matrix or equivalently by the dual code $\mathcal{C}^\perp$. These relations are of the form $\sum_{i=0}^{n} c_i x_i = 0$ where $x_0 x_1 \ldots x_n$ is a codeword of the dual code $\mathcal{C}^\perp$. Thus, if $\mathcal{C}^\perp$ did not have any codeword of weight $l+1$ or less, then given $\{c_i\}_{i \in S}$, c_0 could be equally likely any of the q possible values of $\mathbb{F}_q$ [2] Hence, if the distance $d^\perp$ of the dual code $\mathcal{C}^\perp$ is greater than $l+1$, this requirement would be fulfilled.

[2] Actually, it is sufficient if $\mathcal{C}^\perp$ did not have any codeword $x = x_0 \ldots x_n$ of weight $l+1$ or less in which x_0 is non-zero. We are only being stricter and require that $\mathcal{C}^\perp$ has no codeword of weight $l+1$ or less.

The above scheme works as long as $m \geq n - d + 2$ and $l \leq d^{\perp} - 2$. Since we would like m to be as small as possible and l as large as possible, this construction requires both the code and the dual code to have large distance.

Viewed from this coding theory perspective, Shamir's secret sharing scheme is an instance of the above scheme for $\mathcal{C} = \text{RS}[n+1, l+1, n-l+1]_q$. The dual of the Reed-Solomon $\text{RS}[n+1, l+1, n-l+1]_q$-code is also a Reed-Solomon code, namely $\text{RS}[n+1, n-l, l+2]_q$-code. The key property of RS codes being used here is that they are Maximum Distance Separable (MDS) codes. Note, for the above setting $m = n - d + 2 = n - (n - l + 1) + 2 = l + 1$ and $l = d^{\perp} - 2 = l$, which are exactly the same parameters as in Shamir's scheme.

This coding theory perspective offers further advantages. The above construction can be extended to the case when some of the participating players behave maliciously. Malicious players can cheat and lie about the value of their share to other players. As long as they are not too many of such malicious players, we can still compute the secret by unique-decoding in addition to erasure-decoding.

2.3 Pseudo-randomness

There are a plethora of definitions of pseudo-randomness in the literature each tailor-made for the problem under consideration. However, irrespective of the definition, there seems to be a way to construct pseudo-random sets from codes for each of these definitions. For today's lecture, we shall adopt one of the definitions of pseudo-randomness and demonstrate how codes come handy in building such pseudo-random sets.

Let A be any algorithm with 2 inputs – ϕ, the real input and $x \in \{0,1\}^n$, the random input. Let $A_\phi(x)$ denote the output of A on inputs ϕ and x. Our focus of interest is the random input x and so we shall ignore the real input and the subscript ϕ for the purpose of this lecture.

A set $S \subseteq \{0,1\}^n$ is said to be pseudo-random for A if

$$\Pr_{x \in \{0,1\}^n}[A(x) = 1] \approx \Pr_{x \in S}[A(x) = 1]$$

This definition is valid only if A were computing a boolean function. For the more general case, we could define $S \subseteq \{0,1\}^n$ to be pseudo-random for A if

$$E_{x \in \{0,1\}^n}[A(x)] \approx E_{x \in S}[A(x)]$$

A pseudo-random set S is useful only if

- S is small in size compared to 2^n (i.e., the size of $\{0,1\}^n$, the original sample space.)
- S can be computed efficiently.

Consider the problem MAX-3SAT and a probabilistic algorithm A for MAX-3SAT as given below.
MAX-3SAT

Instance: m clauses $c_1, \ldots, c_m$ on n variables $x_1, \ldots, x_n$ such that each clause has 3 distinct literals.

Goal: To find an assignment that satisfies at least 4/5th of all the clauses.

Algorithm A

1. Pick a random assignment for the variables. (i.e., pick a random n-bit string $a_1 \ldots a_n \in \{0,1\}^n$ and set $x_i = a_i, i = 1, \ldots, n$.)
2. Check if this assignment satisfies at least 4/5th of all the clauses. If so, output the assignment, else abort.

Analysis of Algorithm A**:** A random assignment satisfies a clause with probability $7/8$. Hence, the expected number of clauses satisfied by a random assignment is $7m/8$. Let X be the random variable denoting the number of clauses satisfied by a random assignment. Hence, $E[X] = 7m/8$. We wish to lower bound the probability $\Pr[X \geq 4m/5]$. We have,

$$\begin{aligned} \frac{4m}{5}\Pr\left[X < \frac{4m}{5}\right] + m\Pr\left[X \geq \frac{4m}{5}\right] &\geq E[X] \\ \frac{4}{5} + \frac{1}{5}\Pr\left[X \geq \frac{4m}{5}\right] &\geq \frac{7}{8} \\ \Pr\left[X \geq \frac{4m}{5}\right] &\geq \frac{3}{8} \end{aligned}$$

Thus, the above algorithm succeeds with probability at least $3/8$.

We would now like to employ codes to build a efficiently computable small pseudo-random set S for A.[3]

If we take a re-look at the analysis of algorithm A, we observe that the complete independence of a_i's is not required. It is sufficient if the n-bits were 3-wise independent instead of being fully independent.

A set $S \subseteq \{0,1\}^n$ is said to be l-wise independent if for all distinct $i_1, \ldots, i_l \in \{1, \ldots, n\}$ and $b_1, \ldots, b_n \in \{0,1\}$,

$$\Pr_{x \in S}\left[\bigwedge_{j=1}^{l}(x_{i_j} = b_j\right] = \frac{1}{2^l}$$

The only fact about randomness that we needed for our analysis of algorithm A to go through was that a random assignment satisfies a clause with probability at least $7/8$. This holds good even when the assignments are chosen randomly from a 3-wise independent set. Thus for the algorithm A, it is sufficient if we work with a smaller 3-wise independent set.

We shall now construct a l-wise independent set starting from a linear code. Under what conditions is a linear code S a l-wise independent set? The fact that S is linear translates the requirement of independence among the bits of the codewords into linear dependence of the bits of the codewords. If S has to be l-wise independent, then no subset $\{i_1, \ldots, i_l\}$ of indices should satisfy

$$x_{i_1} + \ldots + x_{i_l} = 0, \forall x \in S$$

Equivalently, as we observed earlier, no codeword of the dual code $S^\perp$ must have weight less than or equal l. In other words, the dual code $S^\perp$ must have distance at least $l+1$.

We also need S to be small for it be meaningful. Equivalently, we desire $S^\perp$ to be as large as possible. Thus, if $S^\perp$ is a large code with distance $l+1$, then S is a small l-wise independent set.

Now, back to our original goal of designing a small 3-wise independent pseudo-random set S for algorithm A. For this, we need a code with large dimension and distance 4. The Hadamard code adjoined with parity satisfies both these properties. Recall the Hadamard code

$$H = [n-1, n-\log n - 1, 3]_2 - \text{code}$$

When adjoined with parity, this code gives

$$H^\oplus = [n, n-\log n - 1, 4]_2 - \text{code}$$

Now, set $S = (H^\oplus)^\perp$. Thus, S is a $[n, \log n + 1, ?]_2$-code. Hence, S has size $2^{\log n + 1} = 2n$ and is 3-wise independent Clearly, S can be computed efficiently. We now perform algorithm A choosing a

[3]There exists a deterministic algorithm (using the theory of conditional expectations) that guarantees to produce an assignment that satisfies at least 7/8th of all the clauses. For today's lecture, we shall feign ignorance of this result and instead focus on constructing a small pseudo-random set S for A.

random assignment from S as described above instead of $\{0,1\}^n$. As the size of the sample space is only polynomial in n, we can run the modified algorithm for each $x \in S$ and obtain a deterministic algorithm for MAX-3SAT.

This equivalence between l-wise independent sets and dual codes of distance $l+1$ was discovered by Alon, Babai and Itai[1]. They used BCH codes to construct l-wise independent sets of size $n^{l/2}$ for each constant l.

References

[1] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567-583, December 1986

[2] Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612-613, November 1979

[3] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *Proc. of the Eleventh Annual ACM Symposium on Theory of Computing*, pages 209-213, Atlanta, Georgia, 30 April-2 May 1979.

[4] ???????????????????