EE 121: Coding for Digital Communication & Beyond	Fall 2013

Lecture 14 — October 22

Lecturer: Prof. Anant Sahai *Scribe: Jingyan Wang*

This lecture covers:

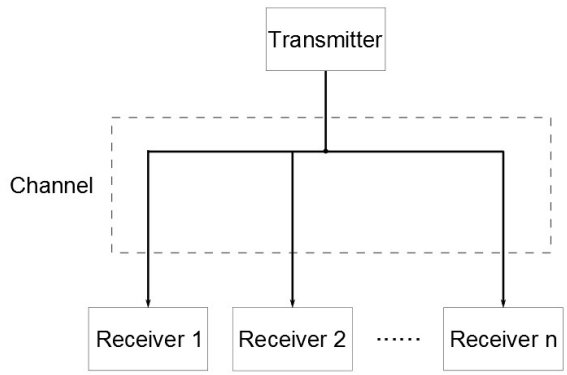
- LT Code
- Ideal Soliton Distribution

14.1 Introduction

So far in this course, we have been working on codes with fixed rate. In traditional linear codes, the length of the message block k, and the length of the codeword n need to be fixed in order to construct the $n \times k$ generator matrix. When n becomes large, the decoding strategy to store in a hashtable the nearest-neighbor codeword for each potential output will be impractical. Rateless codes (also called fountain codes) are a new realm of codes, where the number of the encoded symbols n can be potentially limitless [1]. The receiver will be able to decode the message as long as the number of encoded symbols it receives is slightly larger than the number of symbols in the original message. To appreciate the advantage of rateless codes, consider the following scenario:

One transmitter is sending a message to multiple receivers. In order to have all receivers decode the correct message, the transmitter must send the message at the slowest rate R that the receiver with the most noise will be able to decode. For the good receivers with little noise, most of the received symbols will be redundant. To save energy, we would like these good receivers to switch off as soon as they receive enough information to decode. In the context of linear code, the codewords are generated by $\vec{c} = G\vec{m}$, where $\vec{m}$ is the message. If the channel is an erasure channel (with no error), the received symbols a receiver gets are a subset of the elements in a codeword c. By picking up the corresponding rows in the generator matrix G, we can potentially solve for m if there is still enough information in the equation $\vec{c_{sub}} = G_{sub}\vec{m}$.

In 2002, Michael Luby introduced Luby transform codes (LT code), the first rateless erasure codes that are very efficient as the data length grows [1]. We will discuss the mechanism of LT code in this lecture.



14.2 LT Codes

14.2.1 General Idea

In the last section, we mentioned that a receiver can still recover the message m given only a subset of elements $\vec{x}$ in the received message $\vec{c}$. If the system of linear equations $\vec{x} = G_{sub}\vec{m}$ has a unique solution, the original message $\vec{m}$ is successfully recovered. In this design, there are two problems that we need to address:

- The decoding strategy of solving a system of linear equations by Gaussian elimination has a time complexity of $O(n^3)$, where n is the number of received symbols. This is too slow for practical use.
- We need to guarantee a high probability of successful decoding. For the system of linear equations to have a unique solution, we need a "good" generator matrix G_{sub}.

Hence, the problem reduces to:

- Design a sparse matrix with most 0's and only a few 1's, which has "good" rows.
- Design a "substitutioin" algorithm that leverages the sparsity of the matrix and is significantly faster than Gaussian elimination.

With these ideas in mind, we will develop the construction of LT Code in more detail as we proceed.

14.2.2 Decoding

Decoding involves solving a system of linear equations

$$\underset{n \times 1}{\vec{x}} = \underset{n \times k}{G} \; \underset{k \times 1}{\vec{b}}$$

where x is the received message, G is a $n \times k$ matrix, and $\vec{b}$ is the unknown message we want to solve. The decoding strategy goes as follows:

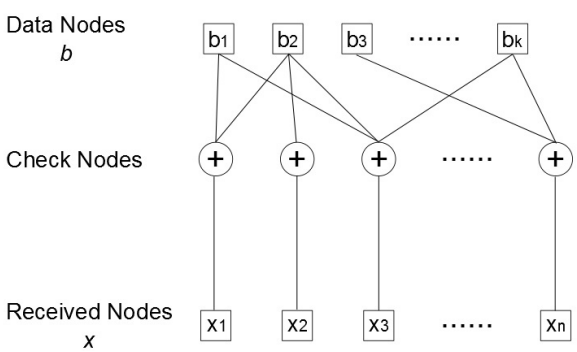
(1) Find an equation of the form $x_i = b_i$

(2) Record x_i as the value for b_i

(3) Substitute x_i in place of b_i in other equations that contain b_i. Simplify.

(4) Go back to (1)

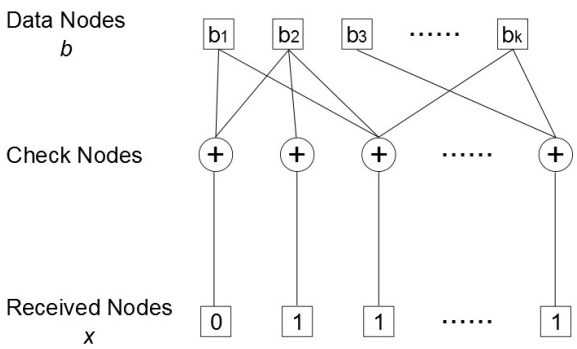
We provide a graph interpretation to this decoding scheme:



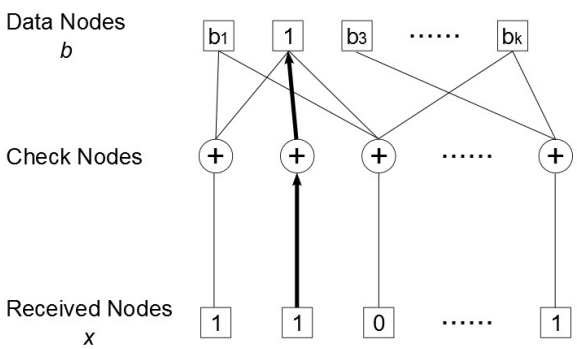
Each data node represents a bit in the original message $\vec{b}$. Each received node represents a bit in the received symbols $\vec{x}$. For each received node x_i, create a corresponding check node and connect them. This check node also connects to all the data nodes that appear in the equation of x_i. Hence, each check node represents an XOR operation of all the data nodes it connects to. For example, the graph above represents the set of linear equations

$$\begin{cases} x_1 = b_1 + b_2 \\ x_2 = b_2 \\ x_3 = b_1 + b_2 + b_k \\ \cdots\cdots \\ x_k = b_3 + b_k \end{cases}$$

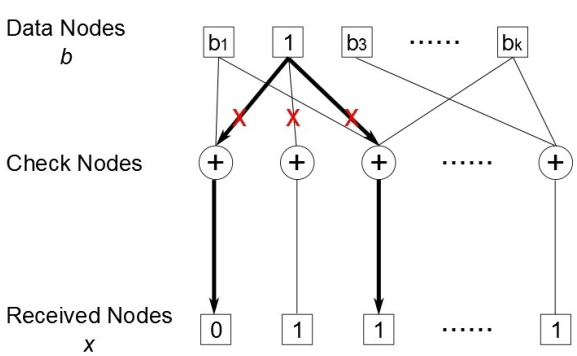
As a running example of the decoding strategy, consider the following situation:



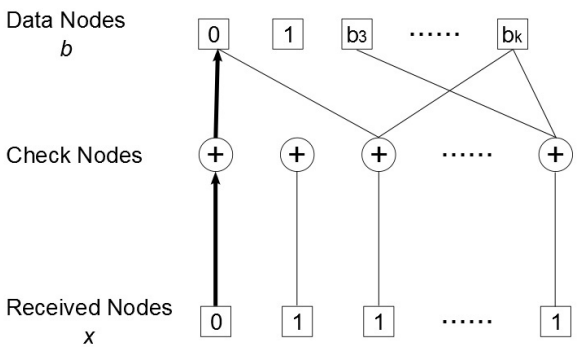
Observe that $b_2 = x_2$ and "pull" the value of x_2 up as the value for b_2:



Substitute $b_2 = 1$ to all check nodes that connect b_2. XOR b_2 with the received nodes of these check nodes. Destroy the edges:



Repeat. Observe that $b_1 = x_1$:



..... so on so forth.

If the system of linear equations has a unique solution, we can hope that all data nodes are decoded at the end. It requires that during each iteration, there always exists a check node that connects to only one data node.

14.2.3 Encoding

We now consider how to construct the generator matrix G (*i.e.* how to connect the check nodes with the data nodes). The encoding strategy goes as follows:

For each row i of the matrix:

- Draw degree D at random from the distribution $P_D(\cdot)$
- Draw D positions uniformly at random from $\{1, 2, \cdots k\}$ to fill in 1's in G
- XOR together these bits as the value for the encoded symbol x_i

The task of generating a "good" matrix G reduces to designing a good distribution $P_D(\cdot)$.

14.3 Degree Distribution $P_D(\cdot)$

14.3.1 All-At-Once Distribution

$$P_D(i) = \begin{cases} 1 & \text{if } i = 1 \\ 0 & \text{otherwise} \end{cases}$$

Every check node connects to only one data node. This distribution has the simplest decoding, because the problem reduces to finding the received nodes that cover all data nodes, and there is no "substitution" work.

Each node connects to a certain data node with probability $\frac{1}{k}$. We look for the number of received nodes we need to have in order to decode all the data nodes. This is essentially

the coupon collector's problem, where each received node is a box, and each data node is a coupon.

Coupon Collector's Problem

We are trying to collect a set of k different cards. We get the cards by buying boxes. Each box contains exactly one card, and it is equally likely to be any of the k cards. How many boxes do you need to buy until we have collected at least one copy of every card?

Expectation

Let X be the number of boxes we need to buy to collect all types of cards. Let X_i be the number of boxes to buy in order to collect the i-th type of card. X_i is a geometric random variable with parameter $1 - \frac{i-1}{k} = \frac{k-i+1}{k}$.

$E[X] = E[\sum_{i=1}^{k} X_i] = \sum_{i=1}^{k} E[X_i] = \sum_{i=1}^{k} \frac{k}{k-i+1} = k \sum_{j=1}^{k} \frac{1}{j} \approx k \ln k$

Fluctuation

How many boxes do you need to buy so that we will collect at least one copy of every card with probability $1 - \delta$?

The probability that we haven't collected a certain card after the r-th box is $(1-\frac{1}{k})^r \leq e^{-\frac{r}{k}}$

Using union bound, the probability that we haven't collected all cards after the r-th box is roughly $ke^{-\frac{r}{k}}$

Assume we buy $k \ln k + m$ boxes to absorb fluctuation. The probability that we haven't collected all cards is $ke^{-\frac{k \ln k + m}{k}}$.

$$ke^{-\frac{k \ln k + m}{k}} = \delta$$

$$ke^{-\ln k} e^{-\frac{m}{k}} = \delta$$

$$e^{-\frac{m}{k}} = \delta$$

$$m = k \ln \frac{1}{\delta}$$

Hence, we need to buy $k \ln k + k \ln \delta$ boxes in order to collect all cards with probability $1 - \delta$.

Going back to the context of the code, we need to receive $k \ln k + k \ln \frac{1}{\delta}$ symbols in order to decode all the data codes. This doesn't look bad, but the $\ln k$ term is still too much overhead in practice, since the rate effectively goes to 0 as k gets large.

14.3.2 Let's improve it

Since All-At-Once distribution doesn't quite work, we need nodes with higher degrees. Let's first introduce some new concepts:

Ripple: the set of all bits not yet decoded, but visible. The ripple is the set of x_i where x_i only connects to one data node b_j, but the value of b_j has not been substituted.

Release: x_i has only one edge and joins the ripple. Some value b_j is known.

Degree: The number of check nodes that connect to a data node. Note that we calculate the degree when a node is created. If some edge on a node is destroyed, we still consider that the degree of the node is unchanged, although the number of edges is smaller.

Initially, all the degree-1 nodes are in the ripple, and we would like to keep the ripple constantly propagating out: degree-1 nodes release some degree-2 nodes. Degree-2 nodes release some other degree-2 nodes, and further release some degree-3 nodes, *etc.* Another metaphor is nuclear chain reaction, we would like the reaction to complete with controlled burn, but not to fizzle or explode.

In All-At-Once distribution, it requires $k \ln k$ received symbols, so there are $k \ln k$ edges on average. We aim for only k received symbols. If we want to cover k received symbols, we need to have $k \ln k$ edges if the edges are chosen independently and randomly. Clearly, when we have higher-degree nodes, the edges are not independently, but it is a good approximation if we keep the total number of edges the same as the all-at-once case. Hence, each check node has an expectation of $\ln k$ edges. We may try:

$\sum_{i=1}^{k} iP_D(i) = \ln k \approx \sum_{i=1}^{k} \frac{1}{i}$

We get a second try on the distribution:

$P_D(i) = \frac{1}{i^2}$ for$1 \leq i \leq k$

This is a converging series, but unfortunately the distribution doesn't sum up to 1. To get around this problem, we can potentially normalize the distribution, but there turns out to be a better way to do it.

14.3.3 Ideal Soliton Distribution

$$P_D(i) = \begin{cases} \frac{1}{k} & \text{if } i = 1 \\ \frac{1}{i(i-1)} & \text{if } 1 < i \leq k \end{cases}$$

This looks like a telescoping series, and we can verify that the distribution does sum up to 1:

$\sum_{i=1}^{k} P_D(i) = \frac{1}{k} + \sum_{i=2}^{k} \frac{1}{i(i-1)}$

$= \frac{1}{k} + (1 - \frac{1}{2}) + (\frac{1}{2} - \frac{1}{3}) + \cdots + (\frac{1}{k-1} - \frac{1}{k})$

$= \frac{1}{k} + (1 - \frac{1}{k}) = 1$

We are going to prove that this distribution leads to "perfect propagation".

Perfect Propagation

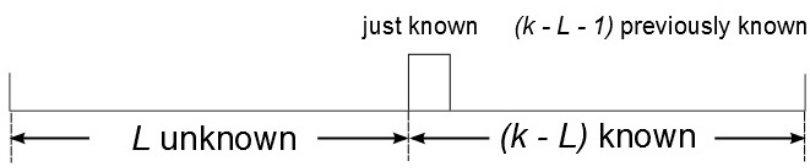
We index the time by L, the number of data symbols still unknown. Let $q(i, L)$ be the probability that an encoded symbol of degree i releases some data symbol when there are L data symbols still unknown.

Initially, all degree-1 nodes release data symbol and all other nodes cannot release:

$q(1, k) = 1$

$q(i, k) = 0$ for $i > 1$

Consider $q(i, L)$. It connects to i nodes. Since it releases at time L, it connects to only 1 unknown. One known node is known at time $(L - 1)$.



For denominator, from the k nodes, we pick i nodes with order, so there are $\prod_{j=0}^{i-1}(k-j)$ combinations in total (The ordering here is not important, so we are artificially over-counting in the denominator. To compensate, we will also over-count in numerator). For numerator, there are i positions and L values for the known node, $(i-1)$ positions for the node just known at the last time step, and $\prod_{j=0}^{i-3}(k-L-1-j)$ combinations to choose and order $(i-2)$ nodes from the $(k-L-1)$ previously known nodes. Hence,

$$q(i, L) = \frac{i(i-1) \cdot L \prod_{j=0}^{i-3}(k-L-1-j)}{\prod_{j=0}^{i-1}(k-j)}$$

Let $r(L)$ be the "release probability" for a random encoded symbol when there are still L unknown symbols.

$r(L) = \sum_{i=1}^{k} P_D(i) q(i, L)$

At $L = k$,

$$r(k) = P_D(1) q(1, k) = \frac{1}{k} \cdot 1 = \frac{1}{k}$$

At $L < k$,

$$r(L) = \sum_{i=2}^{k} P_D(i) q(i, L)$$

$$= \sum_{i=2}^{k} \frac{1}{i(i-1)} \cdot \frac{i(i-1)L \cdot \prod_{j=0}^{i-3}(k-L-1-j)}{\prod_{j=0}^{i-1}(k-j)}$$

$$= \sum_{i=2}^{k} \frac{L \cdot \prod_{j=0}^{i-3}(k-L-1-j)}{\prod_{j=0}^{i-1}(k-j)}$$

$$= \sum_{i=2}^{k} \frac{1}{k} \frac{L \cdot \prod_{j=1}^{i-2}(k-L-j)}{\prod_{j=1}^{i-1}(k-j)}$$

$$= \frac{1}{k} \sum_{i=2}^{k} \frac{L \cdot \prod_{j=1}^{i-2}(k-L-j)}{\prod_{j=1}^{i-1}(k-j)} = \frac{1}{k}$$

Professor Sahai foresees that you the reader is not very happy with the last step here, so he makes a story to fully convince you that

$$\sum_{i=2}^{k} \frac{L \cdot \prod_{j=1}^{i-2}(k-L-j)}{\prod_{j=1}^{i-1}(k-j)} = 1$$

The story goes as follows: $(k-1)$ people are standing in a line. L of them are civilians and $(k-L-1)$ of them are millitary targets. A sniper keeps shooting randomly at the people until a civilian is hit. For each bullet, it is equally likely to kill any of the $(k-1)$ people. What is the probability that the $(i-1)$-th bullet is the first bullet to hit a civilian?

Out of $\prod_{j=1}^{i-1}(k-j)$ permutations of the $(i-1)$ people that are hit, select one of the L civilians to be $(i-1)$-th person that is hit. There are $\prod_{j=1}^{i-2}(k-L-j)$ permutations of the $(i-2)$ millitary targets.

Hence, the probability is $\frac{L\cdot\prod_{j=1}^{i-2}(k-L-j)}{\prod_{j=1}^{i-1}(k-j)}$.

The distribution sums up to 1, so

$$\sum_{i=2}^{k-L+1}\frac{L\cdot\prod_{j=1}^{i-2}(k-L-j)}{\prod_{j=1}^{i-1}(k-j)}=1$$

For $k-L+2\leq i\leq k$, $\prod_{j=1}^{i-2}(k-L-j)=0$.

Hence,

$$\sum_{i=k-L+2}^{k}\frac{L\cdot\prod_{j=1}^{i-2}(k-L-j)}{\prod_{j=1}^{i-1}(k-j)}=0$$

Hence,

$$\sum_{i=2}^{k}\frac{L\cdot\prod_{j=1}^{i-2}(k-L-j)}{\prod_{j=1}^{i-1}(k-j)}=1$$

This is exactly what we are trying to prove. This kind of argument is called a *Combinatorial Proof*, since it is proved by a story.

Conclusion

What result do we get? The probability that a symbol is released at any time step is $\frac{1}{k}$. Given that we have k encoded symbols to start with, the expectation of the number of symbols to be released at any time step is 1.

If everything happens as "expected", we will find exactly 1 symbol to release at each time step.

The ideal Soliton distribution describes a ripple expanding at constant speed, but it cannot protect against fluctuation. If the number of symbol to be released at $L=k$ is 1, one can expect that there is a high probability that there is no symbol to trigger the whole process. In the next lecture, we will continue from here and modify the ideal Soliton distribution. The new distribution, called robust Soliton distribution, will be capable of absorbing fluctuation.

References

[1] Michael Luby *LT Codes* Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science 2002