15:853: Algorithms in the Real World
Fall 2019

Lecture 1

September 3, 2019

Lecturer: Rashmi Vinayak *Scribe: Ian Chiu*

Overview

- Coding is a way to add redundancy to allow for recovery of message through unreliable transmission
- We introduce the idea of error correcting codes, along with simple examples such as repetition code

Introduction to Codes

Example of a code: Welc∗∗e t∗ t∗e fi∗st clas∗ o∗ t∗s course. Y∗u a∗∗ in f∗r a f∗n rid∗ th∗s se∗est∗r.

Despite the erasures in the phrase, we are able to decipher this due to the redundancy of the English language. This is the idea of coding: *codes are a clever way of adding redundancy to a message so we can recover the original message through a noisy channel.*

The general model is the following:

1. Begin with a message m we have to protect or send
2. Encode: $m \mapsto c$ through some encoder to get a codeword c by adding redundancy
3. Transmit: $c \mapsto c'$ through some noisy channel that corrupts the message to yield a received codeword
4. Decode: $c' \mapsto m$ (ideally), or it yields an error via decoder

There are two types of noise in a channel, errors and erasures.
Definition: An *error* is defined to be a field in c' that is modified. For example, $c = the$ and $c' = tha$, we see that index 2 is an error.
Definition: An *erasure* is defined to be a missing field in c'. For example, $c = the$, and $c' = th*$, where $*$ is a character not in our alphabet Σ. We know that the character at index 2 is an erasure because we can see that the character is removed and filled in with a placeholder character.

By the nature of these types of noise, we see that erasures are much easier to fix than errors. For erasures, we know where the issue occurs. For errors, the first step the decoder must do is identify where the error is, if there is any, before fixing it. Beyond the scope of

the course, there are other types of noise such as deletions and additions. As their names suggest, deletions remove characters (e.g. the $\mapsto$ th) and additions add characters (e.g. the $\mapsto$ thea).

There are many applications to coding theory, such as in storage, wireless communication, satellites, and digital TV to name a few. Usually, we use Reed-Solomon codes, but for 3g or 4g cellular data, we use Low Complexity Parity Check Codes.

Block Codes

$$m = \underbrace{s_1, ..., s_k}_{m_1}, \underbrace{s_{k+1}, ..., s_{2k}}_{m_2}, ...$$

In the above image, we have a complete message m we want to transmit, but we treat each block of k elements as its own message. In this case, we break m into blocks of k characters so $m = m_1 m_2 ...$ where each $|m_i| = k$. (The last message may be padded with zeroes so that it also has length k).

We treat each block of k elements as its own message, and we treat them all independently. Some benefits include increased parallelism, and the fact that if some sections are more important, we can encode the section with a stronger code. To define some terms and vocabulary for the section:

- $k = |m|$, which is the size of the original message
- $n = |c|$, which is the size of the code word
- Σ as the alphabet of the code
- C as the set of code words. $C \subseteq \Sigma^n$
- $q = |\Sigma|$
- $\Delta(x, y)$ as the Hamming distance. This is defined to be the number of positions i such that $x_i \neq y_i$.
- minimum distance of a code C is defined as $d = \min_{x \neq y \in C} \Delta(x, y)$

Some simple examples:

1. Repetition code, with $k = 1$. In general, the way this works is we have a message m of length one. Then our encoding would be repeating the message symbol n times. Then for decoding, we would choose the character that is repeated the most.
   An example, for $k = 1, n = 3$. In this case, our encoding would give us the mapping $0 \mapsto 000$, $1 \mapsto 111$.
   In this example, we see that $\Sigma = \{0, 1\}$, $C = \{000, 111\}$, $q = 2$. With this code, we can recover 2 erasures, detect 2 errors, and correct 1 error.

   - If there are two erasures, for example $0 * *$, we know that the correct code word would be 000
   - If there were two errors, we would know. For example, we would know that 011 is not a code word

- We could also correct it if we knew there was at most one error. For example, 010. We would know that the corresponding codeword must be 000 because there is at most one error, and with one error that's the only codeword we could have started from.

2. Single Parity Check Code. In general, we would have $n = k + 1$. The way that encoding works is we have $m_1...m_kp$ where $p = \bigoplus_{i\in[1,k]} m_i$. The way that decoding would work is if we had an erasure at position j, we would take the xor to recover c_j. $c_j = \bigoplus_{i\neq j\in[1,k+1]} c'_i$. To detect an error, we take the big xor $\bigoplus_{i\in[1,k+1]} c'_i$ and if it is not equal to 0, we know we have an error.
   An example where $k = 2, n = 3$.
   Our mapping would then be $00 \mapsto 000, 01 \mapsto 011, 10 \mapsto 101, 11 \mapsto 110$. For this example, we see that $\Sigma = \{0, 1\}$, $C = \{00, 01, 10, 11\}$, $q = 2$.

   - We can recover 1 erasure, detect 1 error, and correct 0 errors
   - We cannot correct even a single error because for this code, we see that $d = 2$, which we can observe by looking at each pair of code words in C
   - We can correct one erasure as the distance between any two codewords is at least 2. For example if we receive $0 * 0$ we know that the codeword must be 000.
   - We can detect two errors because if we see 011 for example, we know that the original code word must be 000 as it is distance 3 from any other code word
   - We cannot detect any errors. If we receive $c' = 111$, c could originally have been 011, 101, or 110.

With our above vocabulary, we are now able to define a block code.
Definition: A *block code* is any error-correcting code that acts on a block of k bits of input data to produce n bits of output data. We denote a block code C as an $(n, k, d)_q$ code if $C \subseteq \Sigma^n$, with $|\Sigma| = q$ and $|C| = q^k$, with minimum distance d between the codewords.
Definition: A *systematic code* is a code where the message appears in the code word.

Theorem 1.1. *A code C with minimum distance d can*
1. *detect any $d - 1$ errors*
2. *recover any $d - 1$ erasures*
3. *correct any $\left\lfloor \frac{d-1}{2} \right\rfloor$ errors*

Proof: Left as an exercise for homework 1. □

Desiderata for codes: We look for codes with the following properties
1. Good rate: low k/n
2. Good distance: large d
3. Small block size k
4. Fast encoding and decoding
5. Others: want to handle bursty/random errors, local decodability, etc.