CS70
Error Correcting Codes

Kelvin Lee

UC Berkeley

March 2, 2021

Overview

Intro to Error Correcting Codes

Intro to Error Correcting Codes

- **Goal:** Transmit messages across an **unreliable** communication channel.

Intro to Error Correcting Codes

- **Goal:** Transmit messages across an **unreliable** communication channel.
- The channel may cause **packets**(parts of the message) to be **lost**, or even **corrupted**.

Intro to Error Correcting Codes

- **Goal:** Transmit messages across an **unreliable** communication channel.
- The channel may cause **packets**(parts of the message) to be **lost**, or even **corrupted**.
- **Error correcting code** is an encoding scheme to protect messages against these errors by introducing redundancy.

Erasure Errors

Erasure Errors

- **Erasure errors** refer to some packets being **lost** during transmission.

Erasure Errors

- **Erasure errors** refer to some packets being **lost** during transmission.
- Suppose that the message consists of n packets and at most k packets are lost during transmission.

Erasure Errors

- **Erasure errors** refer to some packets being **lost** during transmission.
- Suppose that the message consists of n packets and at most k packets are lost during transmission.
- To prevent this error, we encode the initial message into a redundant encoding consisting of $n+k$ packets such that the receiver can reconstruct the message from any n received packets using **Lagrange interpolation**.

General Errors

General Errors

- Now suppose the packets are **corrupted** during transmission due to channel noise. Such error is called **general errors**.

General Errors

- Now suppose the packets are **corrupted** during transmission due to channel noise. Such error is called **general errors**.
- Suppose that k out of n characters are corrupted and we have no idea which k these are.

General Errors

- Now suppose the packets are **corrupted** during transmission due to channel noise. Such error is called **general errors**.
- Suppose that k out of n characters are corrupted and we have no idea which k these are.
- To guard against k general errors, we must transmit $n + 2k$ characters.

General Errors

- Now suppose the packets are **corrupted** during transmission due to channel noise. Such error is called **general errors**.
- Suppose that k out of n characters are corrupted and we have no idea which k these are.
- To guard against k general errors, we must transmit $n + 2k$ characters.
- To reconstruct the polynomial, we need to find a polynomial $P(x)$ of degree $n - 1$ such that $P(i) = r_i$ for at least $n + k$ values of i.

Error-locator Polynomial

Error-locator Polynomial

- To efficiently find the polynomial $P(x)$, we need the locations of the k errors.

Error-locator Polynomial

- To efficiently find the polynomial $P(x)$, we need the locations of the k errors.
- Let $e_1, ..., e_k$ be the k locations at which errors occurred. We don't know where these errors are.

Error-locator Polynomial

- To efficiently find the polynomial $P(x)$, we need the locations of the k errors.
- Let $e_1, ..., e_k$ be the k locations at which errors occurred. We don't know where these errors are.
- Guessing where the errors are will take exponential time, which is inefficient, so we use the **error-locator polynomial**:

$$E(x) = (x - e_1)(x - e_2)\dots(x - e_k).$$

Error-locator Polynomial

- To efficiently find the polynomial $P(x)$, we need the locations of the k errors.
- Let $e_1, ..., e_k$ be the k locations at which errors occurred. We don't know where these errors are.
- Guessing where the errors are will take exponential time, which is inefficient, so we use the **error-locator polynomial**:

$$E(x) = (x - e_1)(x - e_2) \dots (x - e_k).$$

- Then we have the following:

Error-locator Polynomial

- To efficiently find the polynomial $P(x)$, we need the locations of the k errors.
- Let $e_1, ..., e_k$ be the k locations at which errors occurred. We don't know where these errors are.
- Guessing where the errors are will take exponential time, which is inefficient, so we use the **error-locator polynomial**:

$$E(x) = (x - e_1)(x - e_2)\dots(x - e_k).$$

- Then we have the following:

$$P(i)E(i) = r_i E(i) \quad \text{for } 1 \leq i \leq n + 2k.$$

Error-locator Polynomial

- To efficiently find the polynomial $P(x)$, we need the locations of the k errors.
- Let $e_1, ..., e_k$ be the k locations at which errors occurred. We don't know where these errors are.
- Guessing where the errors are will take exponential time, which is inefficient, so we use the **error-locator polynomial**:

$$E(x) = (x - e_1)(x - e_2)\dots(x - e_k).$$

- Then we have the following:

$$P(i)E(i) = r_i E(i) \quad \text{for } 1 \le i \le n + 2k.$$

This is known as the **Berlekamp–Welch algorithm**.

Berlekamp–Welch algorithm

Berlekamp–Welch algorithm

- Define $Q(x) = P(x)E(x)$. We have $n + 2k$ equations with $n + 2k$ unknown coefficients:

Berlekamp–Welch algorithm

- Define $Q(x) = P(x)E(x)$. We have $n + 2k$ equations with $n + 2k$ unknown coefficients:

$$Q(i) = r_i E(i) \quad \text{for } 1 \leq i \leq n + 2k.$$

Berlekamp–Welch algorithm

- Define $Q(x) = P(x)E(x)$. We have $n + 2k$ equations with $n + 2k$ unknown coefficients:

$$Q(i) = r_i E(i) \quad \text{for } 1 \leq i \leq n + 2k.$$

- We can solve the systems of linear equations and get $E(x)$ and $Q(x)$.

Berlekamp–Welch algorithm

- Define $Q(x) = P(x)E(x)$. We have $n + 2k$ equations with $n + 2k$ unknown coefficients:

$$Q(i) = r_i E(i) \quad \text{for } 1 \leq i \leq n + 2k.$$

- We can solve the systems of linear equations and get $E(x)$ and $Q(x)$.
- Finally we compute $\frac{Q(x)}{E(x)}$ to obtain $P(x)$.

Problem Time!