CS 860 Topics in Coding Theory Oct. 9, 2025

Lecture 11

Lecturer: Elena Grigorescu *Scribe: Nosawaru Efemwonkieke*

1 Insertion Deletion Codes

We begin by giving examples of insertion and deletion:

- Insertion: go $\rightarrow$ g<u>l</u>o<u>w</u>
- Deletion: a<u>n</u>t $\rightarrow$ at
- Insertion + Deletion: a<u>n</u>t $\rightarrow$ at<u>e</u>

Motivation: DNA sequencing and storage/communication systems without synchronization.

1.1 Edit Distance

Definition 1 (Edit Distance) *Let $x \in \Sigma^s$, $y \in \Sigma^t$. The edit distance of x and y is defined as*

$$\mathrm{ED}(x, y) = \textit{minimum \# of insertions and deletions to get from } x \textit{ to } y$$

Example

$$x = \text{“Play␣no␣more”}$$

$$y = \text{“Play␣on␣more”}$$

In this example, $\mathrm{ED}(x, y) = 2$ since we only need to delete ‘n’ and insert an ‘n’ on the other side of the ‘o’ in ‘on’. Observe that we also have that $\mathrm{Ham}(x, y) = 2$, where Ham is the Hamming distance.

Claim 2

$$\mathrm{ED}(x, y) \leq 2\mathrm{Ham}(x, y)$$

This is because for any x_i, y_i such that $x_i \neq y_i$ we can delete x_i from x and insert y_i in its place.

It follows that

$$\mathrm{ED}(x, y) = \Omega(s) \Rightarrow \mathrm{Ham}(x, y) = \Omega(s)$$

So if the edit distances is large then the Hamming distance is also large. Is the converse true?

Question If Hamming distance is large is edit distance also large?

Answer Consider the following case for $x, y \in \Sigma^n$:

$$x = 010101 \ldots 10$$

$$y = 101010 \ldots 01$$

In this case, $\text{Ham}(x, y) = n$ but $\text{ED}(x, y) = 2$ (delete the first 1 in y and insert a 1 at the end). So a large Hamming distance does not imply a large edit distance.

We also have an upper bound on the maximum edit distance.

Claim 3 (Maximum Edit Distance) *For $x, y \in \Sigma^n$, the maximum edit distance is*

$$\text{ED}(x, y) \leq 2n$$

Proof This follows from Claim 2 and the fact that the maximum Hamming distance is n. If we let $x = 000 \ldots 00$ and $y = 111 \ldots 11$ we get a concrete instance with maximum edit distance. ■

We use this fact to define relative distance for insertion-deletion codes such that the relative distance belongs to the range $[0, 1]$.

Definition 4 (Relative Distance)

$$\delta = \frac{\text{ED}(x, y)}{2n} \in [0, 1]$$

1.2 Insertion-Deletion Codes

Let Σ be an alphabet. $C \subseteq \Sigma^n$ is an insertion-deletion code with:

- **Minimum distance** d if $\forall x, y \in C : \text{ED}(x, y) \geq d$
- **Relative distance**: $\delta = \min_{x \neq y} \frac{\text{ED}(x, y)}{2n}$
- **Rate:** $r = \frac{\log |C|}{n \log |\Sigma|}$

Claim 5 (Singleton Bound) *Every insertion-deletion code $C \subseteq \Sigma^n$ with relative distance δ and rate r satisfies $r \leq 1 - \delta + \frac{1}{n}$.*

Levenstein '65 asked: Do there exist insertion-deletion codes with $\delta = \Theta(1)$, $r = \Theta(1)$ that are also efficiently encodable/decodable from a constant fraction of error?

- **Schulman-Zuckerman '99**: gave the first construction of "good" codes with efficent encoding/decoding with a large (but constant) alphabet size.
- The following two results gave binary codes with a rate approaching the singleton bound and efficient encoding/decoding.
  - **Guruswami-Li '16**: $r = 1 - \widetilde{O}(\sqrt{\varepsilon})$, $\delta = \varepsilon$

- **Guruswami-Wang '17**: $r = O\left(\varepsilon^5\right)$, $\delta = 1 - \varepsilon$

- **Haeupler-Shahrasbi '17**: used synchronization strings to show that $R \to 1-(\delta+\gamma)$ where δ and γ depend on insertions and deletions respectively.

- **Haeupler-Shahrasbi-Sudan '18**: showed that $R \to 1 - \delta \cdot \varepsilon$ where δ is a constant fraction of the number of deletions. This indicates that the rate does not depend on the number of insertions.

2 Indexing

We hope to be able to reduce synchronization errors to Hamming errors. Towards this goal, we use the idea of appending the index of each character sent by using an expanded alphabet. For example, suppose Alice wants to send the message "HELLO" to Bob. Alice would instead send: "'(H, 1)', '(E, 2)', '(L, 3)','(L, 4)', '(O, 5)"' by using an appropriately expanded alphabet. Now if after the message is sent, the last two symbols are deleted and '(P, 4)' is inserted, Bob receives "'(H, 1)', '(E, 2)', '(L, 3)','(P, 4)"' and reconstructs the intended message as "HELP␣". This looks like the original message, "HELLO", with the fourth character substituted and the last character erased. Intuitively, we are reducing synchronization errors to substitutions and erasures.

Definition 6 (Half-Errors) *We define 1 erasure to be 1 half-error and 1 substitution to be 2 half-errors. If we have e erasures and s substitutions, we have a total of $e+2s$ half-errors.*

Theorem 7 *Using the indexing scheme given above, if the channel makes k synchronization errors, then Bob can reconstruct a string within k half errors from the original message.*

Proof An erasure requires at least 1 deletion (1 sync. error) and a substitution requires at least 1 insertion and 1 deletion (2 sync. errors). ■

Corollary 8 *Given a Hamming error-correcting code C over an alphabet Σ, of length n, distance d, and rate r, we can convert it into an insertion-deletion code C' over the alphabet $\Sigma' = (\Sigma \times [n])$. The new code C' has:*

- $\mathrm{Ham}(C') \geq d$
- *Rate:* $\frac{\log |C|}{n \log (|\Sigma| + \log n)}$
- *If C is efficiently decodable from d half-errors ($s + 2e \leq d$) then C' can correct from d synchronization errors.*

Claim 9 *If we can correct from k insertion-deletion errors then we can correct from k half-errors.*

Proof If we get an erasure, we can view is a deletion with the additional info about the location of the deletion. If we get a substitution error, we can view it as 1 insertion and 1 deletion. ■

Note that the above construction of C' has drawbacks.

- The alphabet size depends on the length of codewords n.
- We lose rate by indexing: $r \to 0$.

Theorem 10 (Schulman-Zuckerman '99) *$\exists \beta > 0$, $\delta > 0$ such that for large enough n there exists an encoder and decoder*

$$E_{SZ} : \{0,1\}^k \to \{0,1\}^{\beta k}$$

$$D_{SZ} : \{0,1\}^{\beta k} \to \{0,1\}^k$$

that can correct from a δ-fraction of insertion-deletion errors in poly(k) time.

Sketch of Proof The idea is to use a combination of indexing, code concatenation, and buffering. We first use code concatenation. For $x \in \{0,1\}^k$ we divide x into $d = k/\log k$ blocks of length $\log k$. The resulting alphabet is $\Sigma_{out} = \{0,1\}^{\log k}$. For C_{out} we use the Reed-Solomon code. Giving us $C_{out} : \left(\{0,1\}^{\log k}\right)^d \to \left(\{0,1\}^{\log k}\right)^{3d}$.

Now let $C_{out}(x) = y_1, y_2 \dots y_{3d}$ where each y_i is a block of length $\log k$. We encode this by indexing each block. Specifically, each y_i becomes (i, y_i). Our overall outer encoding can be viewed as either $T : \{0,1\}^k \to \left(\{0,1\}^{2\log k}\right)^{3d}$ being applied to the entire input or $T : \{0,1\}^k \times [3d] \to \{0,1\}^{2\log k}$ being applied to each block. The upshot is that we can efficiently decode T using Reed-Solomon decoding.

For the inner code we use $C_{in} : \{0,1\}^{2\log k} \to \{0,1\}^{8\log k}$ which can be both constructed and decoded using brute-force search since it is a small length code.

Final encoding:

$$C_{out} \circ C_{in} + 0 \text{ buffers} : C_{in}(T(x,1)),\ 0^{\log k},\ C_{in}(T(x,2)),\ 0^{\log k},\ \dots C_{in}(T(x,3d)),\ 0^{\log k}$$

Decoding: the idea is to find a buffer and then decode the concatenation code by decoding C_{in} and then decoding C_{out}. An issue that arises with this decoding scheme is handling sparse codewords. To resolve this we make the codeword dense by adding 1s between each symbol so that it is easier to find the buffers. ■

Remark The above construction works for local decoding if we use a locally decodable code for the outer encoding. That is to say, we are able to figure out each symbol by looking only at small parts of the codeword rather than the entire codeword.

3 Synchronization Strings

The goal is to use ε-synchronization strings to index data such that k-synchronization errors translate into $k + \varepsilon n$ half-errors over an alphabet of size $O_\varepsilon(1)$.

Definition 11 (Self-Matching) *Suppose we have two copies of a string S. A self-matching is a matching such that there is an edge from each symbol in one copy of S to the same symbol in the other copy of S without any crossing edges. We say an edge is a '**bad edge**' if it connects different positions of S with the same symbol. Otherwise, it is a '**good edge**'.*

Definition 12 (ε-Synchronization String) *S is an ε-synchronization string if for every self-matching*

$$\#\text{ of bad edges} \leq \varepsilon\,(n - \ \#\text{ of good edges})$$

We consider a weaker notion of the above.

Definition 13 (ε-Self Similar String) *S is ε-self similar if for every self-matching*

$$\#\text{ of bad edges} \leq \varepsilon n$$

Observation 14 *A 0-self similar string has no bad edges, so every position of S is a distinct symbol.*

3.1 Existence of Self-Matchings

Theorem 15 *A random string $\in \Sigma^n$ where $|\Sigma| = \Omega\left(\frac{1}{\varepsilon^2}\right)$ is an ε-self matching with high probability.*

Proof Idea Use the probabilistic method with union bound. ■

3.2 Decoding

Theorem 16 *There exists an algorithm which given an ε-self similar string $S = S_1, S_2, \ldots S_n$ and $\left(\widetilde{m}_1, \widetilde{S}_1\right), \left(\widetilde{m}_2, \widetilde{S}_2\right), \ldots, \left(\widetilde{m}_n, \widetilde{S}_n\right)$ can correct all except $O\left(n\sqrt{\varepsilon}\right)$ errors that are not deletions in time $O_\varepsilon(n^2)$.*