Introduction to Coding Theory November 16, 2018

Coding for Insertions and Deletions

Lecturer: Bernhard Haeupler *Scribe: Amirbehshad Shahrasbi*

Thus far, the course has discussed several problems with the common theme of protecting information transmissions against errors that are of symbol erasure or symbol substitution types. Today's lecture will be focused on another type of error that occurs in certain communication or storage applications, called symbol insertions and symbol deletions.

A symbol *insertion* error is defined as a symbol being inserted into a string of symbols and shifting all the following symbols one position forward and a symbol *deletion* error is deleting a symbol from a string of symbols without replacing it with any sort of placeholders. For instance, deleting the third position in string `abacbc` turns it into `abcbc` and inserting symbol `c` in the second position gives `acbacbc`. Symbol insertions and symbol deletions, or *synchronization errors* for short, are prevalent in communication situations where the parties of the communication have no means of staying in sync or any sort of application that involves DNAs like design of memories based on synthetic DNA strands.

Note that, as opposed to symbol erasures and symbol substitutions (Hamming-type errors), insertions and deletions (synchronization errors) can potentially shift around uncorrupted symbols as well. This extra complication makes the problem of coding for insertions and deletions a much harder problem. In fact, our understanding of insertion-deletion codes significantly lags behind our thorough understanding of error correcting codes (ECCs).

1 Insertion-Deletion Codes

To properly define insertion-deletion codes, we have to introduce the notion of edit distance first.

Definition 1 (Edit Distance) *The edit distance between two strings is defined as the minimum number of insertions and deletions needed to convert one to the other. We denote edit distance by* ED().

Using the edit distance metric, one can formally define insertion-deletion codes in a similar manner to error correcting codes: For an alphabet Σ and block length n, code $\mathcal{C} \subseteq \Sigma^n$ is an insertion-deletion code with minimum distance d if

$$\forall x, y \in \mathcal{C} : \mathrm{ED}(x, y) \geq d.$$

Note that the edit distance between two strings of length n can be as large as $2n$. Therefore, the relative distance of code $\mathcal{C}$ is defined as $\frac{d}{2n}$. With this normalization, an insertion-deletion code with relative distance δ can be used to correct δn insertions and deletions. Similar to EECs, the rate of the code $\mathcal{C}$ is defined as $\frac{|\mathcal{C}|}{n \log|\Sigma|}$.

Similar to ECCs, the Singleton bound can be proved for insertion-deletion codes:

Theorem 2 (Singleton Bound for Insertion-Deletion Codes) *For any insertion-deletion code $\mathcal{C} \subseteq \Sigma^n$ with relative distance δ and rate r, we have $r \leq 1 - \delta + \frac{1}{n}$.*

This theorem implies that, similar to ECCs, for any infinite family of insertion-deletion codes with relative distance δ, the rate cannot exceed $1 - \delta$.

2 Previous Work

We now review some of the previous work on the constructions of insertion-deletion codes. The study of insertion-deletion codes was initiated by a work of Levenshtein in 1965 [9].

- Zuckerman and Schulman '99 [11]: This paper provided the first asymptotically good efficient family of insertion-deletion codes. In other words, they provided some (small) constants δ and r and some constant alphabet Σ and an infinite family of insertion-deletion codes with relative distance δ and rate r over alphabet Σ. These codes can be efficiently encoded and decoded.

- Guruswami et al. '16 [4, 5]: These works introduced efficient codes for high-rate and high-distance regimes that are only polynomially off from the Singleton bound. For sufficiently small $\varepsilon > 0$, the following are provided:

  - A family of codes with rate $r = 1 - \tilde{O}\left(\sqrt{\varepsilon}\right)$ and relative distance $\delta = \varepsilon$.
  - A family of codes with rate $r = O\left(\varepsilon^5\right)$ and relative distance $\delta = 1 - \varepsilon$.

These codes were the state-of-the-art insertion-deletion codes until 2016 and resemble the qualities of ECCs that were obtained via code concatenation in Forney's doctoral thesis in 1966 [2].

In this lecture, we introduce *synchronization strings* (from [7]) that can be used to find a black-box conversion of ECCs into insertion-deletion codes with slightly worse qualities. This will help to translate some of our highly sophisticated understanding of ECCs into the realm of insertions and deletions. Namely, we will show that one can derive a family of codes that can approach the Singleton bound by an arbitrarily small additive constant, i.e., codes that $\forall \varepsilon > 0, \delta \in (0, 1)$ have a relative distance of δ and the rate of $1 - \delta - \varepsilon$ or more over some constant alphabet.

3 Indexing

We start by introducing the simple idea of *indexing* that is the key to reduce synchronization errors to Hamming-type errors. Assume that Alice has a stream of symbols $m_1, m_2, \cdots, m_n$ that she wants to send to Bob through a channel that suffers from synchronization errors. Imagine that the alphabet of the channel is large enough so that she can concatenate the position of each symbol to it, i.e., sends the string $(m_1, 1), (m_2, 2), \cdots$ over the channel. We call this procedure *indexing* the symbols of the communication with string $1, 2, \cdots, n$.

Having the stream of symbols indexed with $1, 2, \cdots, n$, Bob will be able to simply recover the position of the symbols he receives by looking at the index value. However, due to the insertions and deletions that occur in the channel, Bob will not be able to (uniquely) identify the symbols in some positions (e.g. positions 6 and 10 in Fig. 1) and also might recover some positions incorrectly (e.g. position 3 in Fig. 1). Therefore, assuming such indexing scheme, Alice and Bob can reduce adversary's synchronization errors into errors that are of the more benign erasure/substitution type. To be more precise, we define the *half-error* measure. We count any combination of e symbol erasures and s symbol substitutions as $e + 2s$ half-errors. It is easy to verify that an error correcting code of distance d can be used to correct from any d half-errors. Having this notion in mind, we prove the following theorem.

Theorem 3 *In the above-mentioned indexing scheme, if the channel performs k synchronization errors, the string that Bob reconstructs is up to k half-errors far from Alice's original message.*

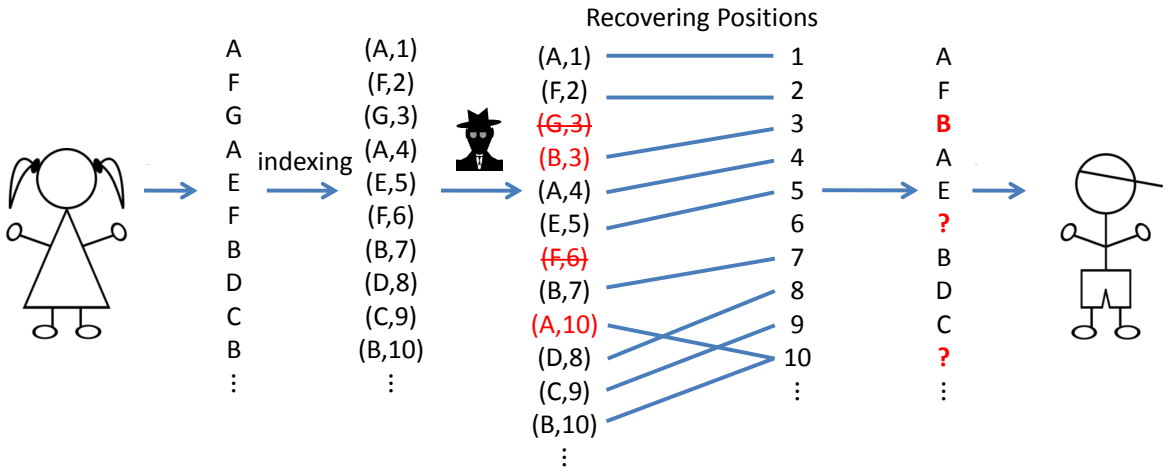

Figure 1: Indexing

Proof Any erasure in the reconstructed string requires at least one synchronization error on the corresponding index in the channel and any symbol substitution (2 half-errors) requires at least two synchronization errors on the corresponding index. ■

A nice complement to Theorem 3 is the following theorem that shows that the problem of insertion-deletion coding is strictly harder than correcting Hamming-type errors.

Claim 4 *Correcting from k insertions and deletions is strictly harder than correcting from k half-errors.*

Proof This can be proved by simply simulating any k half-errors with k insertions and deletions. To do so, for any symbol substitution we delete the symbol and insert a symbol the new value at the same position. For a symbol erasure, we delete the symbol and give the receiver the *extra information* of the position where that deletion has occurred. ■

If Alice and Bob have access to a channel with an alphabet large enough to accommodate the indices, they can use this scheme and Theorem 3 guarantees that they can transform the k synchronization errors introduced by the channel to k Hamming-type errors. Then, they can use an ECC with distance k on top of this reduction to conduct a reliable communication. Indeed, Theorem 3 implies that one can take any ECC and index each codeword of it with $1, 2, \cdots$ to obtain an insertion-deletion code over a larger alphabet that has the same distance and a (slightly) smaller rate (since the number of codewords stays the same but the alphabet size grows).

However, this method entails the following two problems: (1) Any family of codes constructed as such has to have an alphabet size that depends on n, and (2) even if one takes a family of ECCs over a constant alphabet and converts it into a family of insertion-deletion codes using this technique, the rate of the resulting family will approach to zero because by increasing the block length the fraction of the alphabet that is used for indexing (and not conveying actual information) tends to one.

Note that one can still take an ECC with increasing alphabet size and use this conversion to obtain good insertion-deletion codes over super-constant alphabets. In fact, indexing the symbols of the codowords of a Reed-Solomon code with $1, 2, \cdots, n$ would give a family of insertion-deletion codes that perfectly attain the Singleton bound.

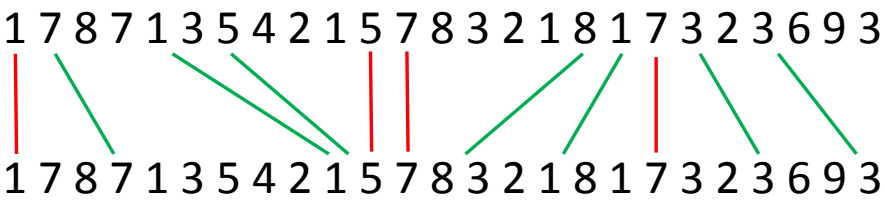

Figure 2: An example of a self-matching. Green edges correspond to good matches and red ones to bad matches.

To make this approach useful in the construction of families of codes over constant-sized alphabets, we replace the string $(1, 2, \cdots, n)$ with strings over some finite alphabet that provide a decent translation from insertions and deletions to half-errors. We introduce ε-synchronization strings as strings that can be used in the indexing scheme to translate any K synchronization errors to $K + n\varepsilon$ half-errors and exist over finite alphabets that are of size $O_\varepsilon(1)$ (the alphabet size is independent of n and only depends on ε).

4 Synchronization Strings

Consider a string S of length n and a matching between two copies of S where symbols at the ends of each edges are identical and edges do not cross each other. We call any such matching a *self-matching* for S. In a self-matching for string S, an edge is called a bad edge if it connects same positions in two copies of S and a good edge otherwise. (see Fig. 2)

Definition 5 (ε-Synchronization String) *S is an ε-synchronization string if for every self-matching of S*

$$\#\ of\ bad\ edges < \varepsilon \cdot (n - \#of\ good\ edges)\,.$$

While synchronization strings are very useful objects in the design of insertion-deletion codes, for the purpose of this lecture, we only use the weaker notion of ε-self-similar defined as follows:

Definition 6 (ε-Self-Similar String) *S is an ε-self-similar string if for every self-matching*

$$\#\ of\ bad\ edges < \varepsilon n.$$

Note that with this definition, a 0-self-similar string is a string with distinct symbols, i.e., $1, 2, 3, \cdots, n$. In the rest of this section, we show that for any $\varepsilon > 0$ and positive integer n, there exists an ε-self-matching string of length n. Also, we will show that using an ε-self-matching in the indexing scheme of Section 3, gives a conversion of K synchronization errors to $K + 4\sqrt{\varepsilon}n$ half-errors.

4.1 Existence

Lemma 7 *A random string $S \in [q]^n$ is $\Theta\left(\frac{1}{\sqrt{q}}\right)$-self-matching with high probability. Therefore, taking $q = \Omega(\varepsilon^{-2})$, the random string over alphabet $[q]$ is ε-self-similar with high probability.*

Proof To prove this, take the union bound over all possible non-crossing matchings of size εn in S. Note that the number of all possible non-crossing matchings of size εn in S is no more than $\binom{n}{n\varepsilon}^2$ since any non-crossing matchings of size εn can be uniquely identified by picking εn positions from each copies.

Further, the probability of string S being such that the edges of some specific matching of size εn place between identical symbols is $\frac{1}{q^k}$. By taking the union bound over all such possible self-matchings, we have

$$\Pr\{\text{Having a bad matching of size} \geq \varepsilon n\} \leq \binom{n}{\varepsilon n}^2 \left(\frac{1}{q}\right)^{\varepsilon n} \leq \left(\frac{e}{\varepsilon\sqrt{q}}\right)^{2n\varepsilon}.$$

Therefore, if we pick $q > \frac{e^2}{\varepsilon^2}$, the base becomes smaller than 1 and the probability tends to zero as n goes to infinity. ■

4.2 Decoding

We now proceed to describing the decoding procedure, i.e., how Bob can recover the position of the symbols he receives using the indexed ε-self-similar string. Assume that Alice and Bob are using string S for indexing and Bob receives the string $\left\{\left(\tilde{m}_i, \tilde{S}_i\right)\right\}_i$. The algorithm that Bob needs to use to recover the actual position of the symbols is as follows:

1. Compute the longest non-crossing matching between S and $\tilde{S}$.
2. For any symbol in $\tilde{S}$ like $\tilde{S}_i$ that is matched to some S_j, assign the received symbol $\tilde{S}_i$ to position j.
3. Remove the symbols in $\tilde{S}$ that are matched.
4. Repeat this procedure $\frac{1}{\sqrt{\varepsilon}}$ times.

Theorem 8 *Using an ε-self-similar string in the indexing scheme of Section 3 along with the decoding algorithm described above, Bob can recover the string sent by Alice by up to $k + 4n\sqrt{\varepsilon}$ half-errors where k is the number of synchronization errors that happen in the channel.*

Proof Two types of incorrect decoding of received symbols can happen in this procedure:

(1) Some symbol (S_i, m_i) successfully passes through the channel and arrives as $(\tilde{S}_j, \tilde{m}_j)$ at Bob's end without being removed but $\tilde{S}_j$ gets incorrectly matched to some other S'_i.

(2) Some symbol (S_i, m_i) successfully passes through the channel and arrives as $(\tilde{S}_j, \tilde{m}_j)$ at Bob's end without being removed but never matches to any symbols in any of the matchings.

Type I: Any incorrect match between a symbol in $\tilde{S}$ like $\tilde{S}_j$ that corresponds to S_i and some other symbol in S like S_k (where $i \neq k$) implies a pair of identical symbols in S ($S_i = S_k$). Since S is ε-self-similar, there is no more than $n\varepsilon$ such symbols. Since there is a total of $\frac{1}{\sqrt{\varepsilon}}$ rounds, the total mis-decodings of type I is no more than $\varepsilon n \cdot \frac{1}{\sqrt{\varepsilon}} = n\sqrt{\varepsilon}$.

Type II: If there are k undeleted symbols left unmatched at the end of the procedure, then there is a monotone matching between S and the remainder of $\tilde{S}$ at the end of the procedure that is of size k. This implies that the size of each of the previous matchings is at least k. This means that we have matched at least $\frac{k}{\sqrt{\varepsilon}}$ in total. We have that $\frac{k}{\sqrt{\varepsilon}} \leq n \Rightarrow k \leq n\sqrt{\varepsilon}$.

Therefore, the total number of incorrect decodings will not exceed $2n\sqrt{\varepsilon}$. Note that in the case of indexing with trivial string $1, 2, \cdots, n$ where there is no mis-decodings, k insertions and deletions can be converted to k half-errors. It is easy to verify that each mis-decoding can introduce up to two more half-errors. Therefore, the total number of half-errors in the string obtained by this decoding algorithm is $k + 4n\sqrt{\varepsilon}$. ■

Having the decoding guarantee of Theorem 8, one can take the following family of error correcting code by Guruswami and Indyk [3] that approach the singleton bound by an arbitrarily small additive constant and convert it to a family of insertion-deletion codes with the same quality.

Theorem 9 (Theorem 3 from [3]) *For every r, $0 < r < 1$, and all sufficiently small $\epsilon > 0$, there exists a family of codes of rate r and relative distance at least $(1 - r - \epsilon)$ over an alphabet of size $2^{O(\epsilon^{-4} r^{-1} \log(1/\epsilon))}$ such that codes from the family can be encoded in linear time and can also be (uniquely) decoded in linear time from $(1-r-\epsilon)$ fraction of half-errors, i.e., a fraction e of errors and s of erasures provided $2e + s \leq (1 - r - \epsilon)$.*

Theorem 10 *For any $\delta \in (0, 1)$ and sufficiently small $\varepsilon > 0$, there exists a family of insertion-deletion code with distance δ that achieves a rate of $1 - \delta - \varepsilon$ or more over an alphabet of size $\exp\left(\varepsilon^{-4} \log \frac{1}{\varepsilon}\right) = O_\varepsilon(1)$ that are encodable in linear time and decodable in quadratic time in terms of the block length.*

Proof Take a family of error correcting codes $\{\mathcal{C}_n\}$ from Theorem 9 with parameters $\delta_\mathcal{C} = \delta + \frac{\varepsilon}{3}$ and $\varepsilon_\mathcal{C} = \frac{\varepsilon}{3}$. Further, take $\varepsilon_S = \left(\frac{\varepsilon}{12}\right)^2$ and take a ε_S-self-similar string S and index the codewords of codes in $\{\mathcal{C}\}$ to obtain the family of insertion-deletion codes $\{\mathcal{C}'\}$.

Theorem 8 implies that in presence of $n\delta$ synchronization errors, Bob is able to reconstruct the non-index part of Alice's message by up to $n\delta + 4n\sqrt{\varepsilon_S} = n\left(\delta + \frac{\varepsilon}{3}\right) = n\delta_\mathcal{C}$ half-errors. Since the family of codes $\{\mathcal{C}\}$ have Hamming distance of $n\delta_\mathcal{C}$, Bob would be able to uniquely decode Alice's message. This implies that the family of codes $\{\mathcal{C}'\}$ is an insertion-deletion code with distance δ or more.

We now find the rate of the $\mathcal{C}'$. Note that by indexing, the number of codewords stay the same but the rate drops as the size of the alphabet grows. Let us denote the alphabet of codes $\mathcal{C}$ (respectively $\mathcal{C}'$) with $\Sigma_\mathcal{C}$ (resp. $\Sigma_{\mathcal{C}'}$) and the alphabet of S with Σ_S. Then, the rate of $\mathcal{C}'$ can be bounded below as follows.

$$\begin{aligned}
R_{\mathcal{C}'_i} &= \frac{|\mathcal{C}'_i|}{n \log |\Sigma_{\mathcal{C}'}|} \\
&= \frac{|\mathcal{C}_i|}{n \log |\Sigma_\mathcal{C}|} \cdot \frac{n \log |\Sigma_\mathcal{C}|}{n \log |\Sigma_{\mathcal{C}'}|} \\
&= R_{\mathcal{C}_i} \cdot \frac{\log |\Sigma_\mathcal{C}|}{\log |\Sigma_\mathcal{C}| + \log |\Sigma_S|} \\
&= R_{\mathcal{C}_i} \cdot \frac{1}{1 + \frac{\log |\Sigma_S|}{\log |\Sigma_\mathcal{C}|}} \\
&= R_{\mathcal{C}_i} \cdot \frac{1}{1 + o(\varepsilon)} > R_{\mathcal{C}_i} - \frac{\varepsilon}{3} \qquad (1)
\end{aligned}$$

Section 4.2 follows from the fact that $|\Sigma_\mathcal{C}| = 2^{O(\varepsilon^{-4} \log(1/\epsilon))}$ (according to Theorem 9) and $|\Sigma_S| = O\left(\varepsilon^{-2}\right)$ (according to Theorem 7).

Therefore, the rate of the family of codes $\mathcal{C}'$ is at least

$$r_{\mathcal{C}'} \geq r_\mathcal{C} - \frac{\varepsilon}{3} \geq \left(1 - \delta_\mathcal{C} - \frac{\varepsilon}{3}\right) - \frac{\varepsilon}{3} \geq 1 - \delta - 3 \cdot \frac{\varepsilon}{3} = 1 - \delta - \varepsilon.$$

Finally, since codes $\{\mathcal{C}_i\}$ are encodable in linear time and indexing only adds linear time to the encoding procedure, the resulting code also has a linear encoding complexity. Further, since $\{\mathcal{C}_i\}$ is decodable in linear time and the extra step of decoding the indices from Section 4.2 performs a constant number of longest common subsequence computations, the decoding complexity of the resulting code is quadratic. ■

We finish this section by adding an interesting remark about the indexing-based construction of insertion-deletion codes. Indexing the codewords of a code over alphabet $\Sigma_{\mathcal{C}}$ with a self-similar string over alphabet Σ_S inherently wastes a $\frac{|\Sigma_S|}{|\Sigma_{\mathcal{C}}|+|\Sigma_S|}$ fraction of the information in each symbol. Therefore, if one wants to stay close to the singleton bound by an ε margin, $\frac{\log|\Sigma_S|}{\log|\Sigma_{\mathcal{C}}|+\log|\Sigma_S|}$ has to be smaller than ε. Given that ε-self-similar strings only exist over alphabets of size $\Omega(\varepsilon^{-1})$, this forces the $|\Sigma_{\mathcal{C}}|$ and consequently the size of the alphabet of the resulting code to be at least exponentially large in $\frac{1}{\varepsilon}$. However, for ordinary error correcting codes, there are families of codes that approach the singleton bound and are over alphabets of size poly $\left(\varepsilon^{-1}\right)$.

While this suggests that large alphabet size is an intrinsic shortcoming of the indexing method, it can be proved that any family of insertion-deletion codes that approaches the singleton bound with an ε error has to have an alphabet that is at least exponentially large in terms of ε^{-1}.

5 Document Exchange

The rest of this lecture is going to be on a problem closely related to insertion-deletion coding called the document exchange problem. Assume that a server holds a file F and its client is keeping an out dated version of F denoted by F'. Further, assume that it is known to both the server and the client that the edit distance of F and F' is no more than k. In the document exchange problem, the goal is for the server to compute a summary S_F and send it to the client so that the client will be able to recover F using S_F and its outdated version F' ($\mathrm{Rec}(F', S_F)$).

Note that the server is not aware of client's out dated file so he cannot just send the position of the edits that are required to turn F' to F. However, since each of the k edits can occur in any of the n positions, the bit-size of the summary has to be at least $\Omega(k \log \binom{n}{k}) = \Omega(k \log \frac{n}{k})$ to guarantee a correct recovery. In 1991, Orlitsky [10] has shown that there exists a deterministic document exchange protocol that achieves $O(k \log \frac{n}{k})$ summary size. However, designing efficient document exchange protocols (deterministic or probabilistic)that yield small message sizes has been an interesting problem since then.

Before discussing document exchange protocols, we remark that the deterministic document exchange and insertion-deletion coding are highly related problems. A systematic error correcting code is an error correcting code whose encoder outputs the input (systematic part) appended by some redundant information (non-systematic part).

Having a systematic synchronization code $\mathcal{C}$ that corrects from k synchronization errors with encoding and decoding functions $\mathrm{Enc}(\cdot)$ and $\mathrm{Dec}(\cdot)$, one can construct a deterministic document exchange scheme with summary S_F being the non-systematic part of $\mathrm{Dec}(\cdot)$ and the recovery procedure $\mathrm{Rec}(S_F) = \mathrm{Dec}(\langle F', S_F\rangle)$. Further, having a deterministic document exchange scheme $(S_F, \mathrm{Rec}(\cdot))$, one can simply construct a systematic synchronization code that can correct from k synchronization errors by taking the input and appending a non-systematic part that consists of the encoding of S_F with a synchronization code that corrects from k errors.

5.1 $O\left(k \log n \log \frac{n}{k}\right)$ Randomized Document Exchange (Irmak et al. [8])

Take string F and split it into $4k$ substrings of length $\frac{n}{4k}$. Since F' is different from F by up to k insertions and deletions, at least $3k$ of such substrings appear in F'. Based on this observation, we present the following interactive protocol for document exchange.

The server splits F into $4k$ substrings of length $\frac{n}{4k}$ and uses an appropriate hash function to compute and send over the hash value for all such substrings. The client can use those hash values to recover the $3n$ uncorrupted substrings of F. We assume that the hash size is

$\Omega(\log n)$ so with high probability, no hash collision happens during the procedure. Then, the client sends the positions of the (up to k) substrings of F that are not recovered, the server splits each of them into two substrings (of length $\frac{n}{2k}$) and sends the hash value for each of those $2k$ new intervals. Repeating this procedure t times, the client will be able to recover F with the exception of k substrings of length $O(\frac{n}{2^t k})$. For $t = \log \frac{n}{k}$, the remaining substrings are of constant size. Then the server sends all those substrings.

This procedure has $\log \frac{n}{k}$ levels and in each level the server sends $O(k \log n)$ bits of information. So, in total, $O\left(k \log n \log \frac{n}{k}\right)$ bits of information are sent from the server to the receiver.

The next step is to convert this interactive protocol into a document exchange scheme. The trick is to use a systematic erasure code to convey the information that the client needs to know in each level without knowing which parts of F has been reconstructed. More precisely, in ith level, the server splits F into substrings of length $\frac{n}{2^i k}$, computes the hash value for each substring $H^i = \langle h^i_1, \cdots, h^i_{2^i k} \rangle$, then uses a systematic erasure code that corrects from $2k$ erasures to compute the non-systematic part for input H^i, N^i, and sends it to the client. On the other end, at level i, the client has already constructed all of F except $2k$ substrings of length $\frac{n}{2^i k}$. So, the client can compute H^i by up to $2k$ erasures. Therefore, using the non-systematic part N^i, the client can compute the hash values for the unknown substrings and move on to the next level. Overall, this gives an efficient randomized document exchange scheme with message size $O\left(k \log n \log \frac{n}{k}\right)$.

5.2 $O\left(k \log \frac{n}{k}\right)$ Deterministic Document Exchange [6, 1]

The document exchange protocol described above can be improved to achieve asymptotically optimal communication efficiency $O\left(k \log \frac{n}{k}\right)$ by reducing the hash sizes to some constant independent of n. This will be possible by modifying the protocol using the following ideas to make it resilient to hash collisions:

1. With smaller hash sizes, hash collisions will be inevitable. However, even if the client incorrectly reconstructs a substring of F, the hash values corresponding to its substrings in next level will not agree with server's message. In fact, it can be shown that if server uses a systematic ECC instead of the systematic erasure code, the client will be able to discover and correct the incorrectly reconstructed substrings that occur due to hash collisions in earlier levels.

2. Take into account the fact that uncorrupted substrings of F appear in F' in the same order when trying to reconstruct F on the client side using hash values.

It can be proved that applying these ideas, the summary size of the document exchange protocol can be reduced to $O\left(k \log \frac{n}{k}\right)$ and, further, the protocol can even be derandomized. [6, 1]

References

[1] Kuan Cheng, Zhengzhong Jin, Xin Li, and Ke Wu. Deterministic document exchange protocols, and almost optimal binary codes for edit errors. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2018.

[2] G David Forney. Concatenated codes. 1965.

[3] Venkatesan Guruswami and Piotr Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transactions on Information Theory*, 51(10):3393–3400, 2005.

[4] Venkatesan Guruswami and Ray Li. Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes. In *Information Theory (ISIT), 2016 IEEE International Symposium on*, pages 620–624. IEEE, 2016.

[5] Venkatesan Guruswami and Carol Wang. Deletion codes in the high-noise and high-rate regimes. *IEEE Transactions on Information Theory*, 63(4):1961–1970, 2017.

[6] Bernhard Haeupler. Optimal document exchange and new codes for small number of insertions and deletions. *arXiv preprint arXiv:1804.03604*, 2018.

[7] Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: Codes for insertions and deletions approaching the singleton bound. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, 2017.

[8] Utku Irmak, Svilen Mihaylov, and Torsten Suel. Improved single-round protocols for remote file synchronization. In *INFOCOM*, pages 1665–1676, 2005.

[9] Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR 163*, 4:845–848, 1965.

[10] Alon Orlitsky. Interactive communication: Balanced distributions, correlated files, and average-case complexity. In *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on*, pages 228–238. IEEE, 1991.

[11] Leonard J. Schulman and David Zuckerman. Asymptotically good codes correcting insertions, deletions, and transpositions. *IEEE transactions on information theory*, 45(7):2552–2557, 1999.