# Codes for Editing Errors

**Madhu Sudan**
Harvard University

Based on (1) Haeupler and Shahrasbi – FOCS 2017
(2) Haeupler, Shahrasbi, S. – ICALP 2018

# Edit Distance & Codes

- $X \in \Sigma^n$ and $Y \in \Sigma^m$:
  - $X \rightarrow_{\Delta,\Gamma} Y$ : If deleting $\leq \Delta$ symbols from $X$ and then inserting $\leq \Gamma$ symbols gives $Y$.

# Edit Distance & Codes

- $X \in \Sigma^n$ and $Y \in \Sigma^m$:
  - $X \rightarrow_{\Delta,\Gamma} Y$ : If deleting $\leq \Delta$ symbols from $X$ and then inserting $\leq \Gamma$ symbols gives $Y$.
  - Example: $010101010110 \rightarrow_{1,2} 0110100101110$

# Edit Distance & Codes

- $X \in \Sigma^n$ and $Y \in \Sigma^m$:
  - $X \to_{\Delta,\Gamma} Y$ : If deleting $\leq \Delta$ symbols from $X$ and then inserting $\leq \Gamma$ symbols gives $Y$.
  - Example: $0101010110110 \to_{1,2} 0111010010111110$

# Edit Distance & Codes

- $X \in \Sigma^n$ and $Y \in \Sigma^m$:
  - $X \to_{\Delta,\Gamma} Y$ : If deleting $\leq \Delta$ symbols from $X$ and then inserting $\leq \Gamma$ symbols gives $Y$.
  - Example: $0101011010110 \to_{1,2} 0111010010111110$
  - Sanity check: $X \to_{\Delta,\Gamma} Y \Rightarrow Y \to_{\Gamma,\Delta} X$

# Edit Distance & Codes

- $X \in \Sigma^n$ and $Y \in \Sigma^m$:
  - $X \to_{\Delta,\Gamma} Y$ : If deleting $\leq \Delta$ symbols from $X$ and then inserting $\leq \Gamma$ symbols gives $Y$.
  - Example: $0101011010110 \to_{1,2} 0111010010111 10$
  - Sanity check: $X \to_{\Delta,\Gamma} Y \;\Rightarrow\; Y \to_{\Gamma,\Delta} X$
- $(\Delta, \Gamma)$-Edit Distance Codes:
  - $E: \Sigma^k \to \Sigma^n$ ; $D: \Sigma^* \to \Sigma^k$

# Edit Distance & Codes

- $X \in \Sigma^n$ and $Y \in \Sigma^m$:
  - $X \to_{\Delta,\Gamma} Y$ : If deleting $\leq \Delta$ symbols from $X$ and then inserting $\leq \Gamma$ symbols gives $Y$.
  - Example: $0101011010110 \to_{1,2} 01101001011110$
  - Sanity check: $X \to_{\Delta,\Gamma} Y \;\Rightarrow\; Y \to_{\Gamma,\Delta} X$
- $(\Delta, \Gamma)$-Edit Distance Codes:
  - $E: \Sigma^k \to \Sigma^n$ ; $D: \Sigma^* \to \Sigma^k$

    $\forall\, X \in \Sigma^k, Y \in \Sigma^* \text{ s.t. } E(X) \to_{\Delta,\Gamma} Y, \qquad D(Y) = X$

Unique decoding

# Edit Distance & Codes

- $X \in \Sigma^n$ and $Y \in \Sigma^m$:
  - $X \to_{\Delta,\Gamma} Y$ : If deleting $\leq \Delta$ symbols from $X$ and then inserting $\leq \Gamma$ symbols gives $Y$.
  - Example: $0101011010110 \to_{1,2} 0111010010111 10$
  - Sanity check: $X \to_{\Delta,\Gamma} Y \;\Rightarrow\; Y \to_{\Gamma,\Delta} X$
- $(\Delta, \Gamma)$-Edit Distance Codes:
  - $E: \Sigma^k \to \Sigma^n$ ; $D: \Sigma^* \to \Sigma^k$

    $\forall X \in \Sigma^k, Y \in \Sigma^* \text{ s.t. } E(X) \to_{\Delta,\Gamma} Y, \quad D(Y) = X$

    Unique decoding
  - $E: \Sigma^k \to \Sigma^n$ ; $D: \Sigma^* \to \binom{\Sigma^k}{L}$

    $\forall X \in \Sigma^k, Y \in \Sigma^* \; s.t. E(X) \to_{\Delta,\Gamma} Y, \quad D(Y) \ni X$

    List decoding

# Edit-Distance Codes - 2

- Given family of code $E = \left(E_n : \Sigma^{k_n} \to \Sigma^n\right)_n$
  - Rate = $R = \lim_{n\to\infty} \frac{k_n}{n}$ ;

# Edit-Distance Codes - 2

- Given family of code $E = \left(E_n : \Sigma^{k_n} \to \Sigma^n\right)_n$
  - Rate = $R = \lim_{n\to\infty} \frac{k_n}{n}$ ;
  - $(\delta, \gamma)$-code if $E_n$ is $(\delta n, \gamma n)$-code $\forall n$

# Edit-Distance Codes - 2

- Given family of code $E = \left(E_n : \Sigma^{k_n} \to \Sigma^n\right)_n$
  - Rate = $R = \lim_{n\to\infty} \frac{k_n}{n}$ ;
  - $(\delta, \gamma)$-code if $E_n$ is $(\delta n, \gamma n)$-code $\forall n$
- Questions: what $(\delta, \gamma, R)$ achievable?

# Edit-Distance Codes - 2

- Given family of code $E = \left(E_n : \Sigma^{k_n} \to \Sigma^n\right)_n$
  - Rate = $R = \lim_{n\to\infty} \frac{k_n}{n}$ ;
  - $(\delta, \gamma)$-code if $E_n$ is $(\delta n, \gamma n)$-code $\forall n$
- Questions: what $(\delta, \gamma, R)$ achievable?
  - With list-decoding?
  - While $q = |\Sigma| = O(1)$
  - Algorithmically?

# Brief History

- Notion dates back to ‘70s: Levenstein
- [Schulman-Zuckerman ‘90s]:
  $q = poly(n), \qquad R \to 1 - (\delta + \gamma)$ unique decoding
  - Algorithmic!

# Brief History

- Notion dates back to ‘70s: Levenstein
- [Schulman-Zuckerman ‘90s]:
  $q = poly(n), \qquad R \to 1 - (\delta + \gamma)$ unique decoding
  - Algorithmic!

# Brief History

- Notion dates back to ‘70s: Levenstein
- [Schulman-Zuckerman ‘90s]:
  $q = poly(n), \qquad R \to 1 - (\delta + \gamma)$ unique decoding
  - Algorithmic!

# Brief History

- Notion dates back to ‘70s: Levenstein
- [Schulman-Zuckerman ‘90s]:
  $q = poly(n)$, $R \to 1 - (\delta + \gamma)$ unique decoding
  - Algorithmic!

# Brief History

- Notion dates back to ‘70s: Levenstein
- [Schulman-Zuckerman ‘90s]:
  $q = poly(n), \qquad R \to 1 - (\delta + \gamma)$ unique decoding
  - Algorithmic!
  - List-decoding? $q = O(1)$?

# Brief History

- Notion dates back to '70s: Levenstein
- [Schulman-Zuckerman '90s]:
  $q = poly(n)$, $R \to 1 - (\delta + \gamma)$ unique decoding
  - Algorithmic!
  - List-decoding? $q = O(1)$?
- [Haeupler-Shahrasbi'17]:
  - $q = O(1)$; $R \to 1 - (\delta + \gamma)$ unique decoding

# Brief History

- Notion dates back to '70s: Levenstein
- [Schulman-Zuckerman '90s]:
  $q = poly(n)$, $R \to 1 - (\delta + \gamma)$ unique decoding
  - Algorithmic!
  - List-decoding? $q = O(1)$?
- [Haeupler-Shahrasbi'17]:
  - $q = O(1)$; $R \to 1 - (\delta + \gamma)$ unique decoding
- [Haeupler-Shahrasbi-S.'18]
  - $q = O_{\gamma,\delta,R}(1)$ ; $R \to 1 - \delta$ list decoding

# Schulman-Zuckerman Idea

- Indexing:
  - $E: \Sigma^k \to \Sigma^n \quad \to \quad E': \Sigma^k \to (\Sigma \times [n])^n$
    - $E'(x)_i = (E(x)_i, i)$

# Schulman-Zuckerman Idea

- Indexing:
  - $E: \Sigma^k \to \Sigma^n \quad \to \quad E': \Sigma^k \to (\Sigma \times [n])^n$
    - $E'(x)_i = (E(x)_i, i)$
    - $Rate(E') = Rate(E) - \frac{\log n}{\log |\Sigma|}$

# Schulman-Zuckerman Idea

- Indexing:
  - $E: \Sigma^k \to \Sigma^n \quad \to \quad E': \Sigma^k \to (\Sigma \times [n])^n$
    - $E'(x)_i = (E(x)_i, i)$
    - $Rate(E') = Rate(E) - \frac{\log n}{\log |\Sigma|} \qquad \Rightarrow n \ll |\Sigma|$ !

# Schulman-Zuckerman Idea

- Indexing:
  - $E: \Sigma^k \to \Sigma^n \quad \to \quad E': \Sigma^k \to (\Sigma \times [n])^n$
    - $E'(x)_i = (E(x)_i, i)$
    - $Rate(E') = Rate(E) - \frac{\log n}{\log |\Sigma|} \qquad \Rightarrow n \ll |\Sigma|$ !
    - Insertion $(E')$ ⇒ Erasure $(E)$
    - Deletion $(E')$ ⇒ Erasure $(E)$
    - Same location Ins.+Del. $(E')$ ⇒ Error$(E)$

# Schulman-Zuckerman Idea

- Indexing:
  - $E: \Sigma^k \to \Sigma^n \quad \to \quad E': \Sigma^k \to (\Sigma \times [n])^n$
    - $E'(x)_i = (E(x)_i, i)$
    - $Rate(E') = Rate(E) - \frac{\log n}{\log |\Sigma|} \qquad \Rightarrow n \ll |\Sigma|$ !
    - Insertion $(E')$ $\Rightarrow$ Erasure $(E)$
    - Deletion $(E')$ $\Rightarrow$ Erasure $(E)$
    - Same location Ins.+Del. $(E')$ $\Rightarrow$ Error$(E)$
    - $E$ has distance $\gamma + \delta \Rightarrow E'$ is $(\delta, \gamma)$-code.

# Haeupler-Shahrasbi strategy

- Index with string $S = S_1, \ldots, S_n \in [c]^n$, w. $c = O(1) \ll n$
  - $E: \Sigma^k \to \Sigma^n \quad \to \quad E': \Sigma^k \to (\Sigma \times [c])^n$
    - $E'(x)_i = (E(x)_i, S_i)$

# Haeupler-Shahrasbi strategy

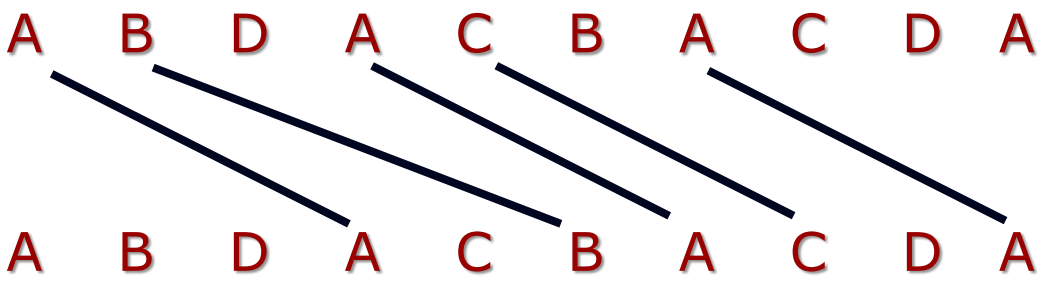
- Index with string $S = S_1, \ldots, S_n \in [c]^n$, w. $c = O(1) \ll n$
  - $E: \Sigma^k \to \Sigma^n \quad \to \quad E': \Sigma^k \to (\Sigma \times [c])^n$
    - $E'(x)_i = (E(x)_i, S_i)$
    - $Rate(E') = Rate(E) - \frac{\log c}{\log |\Sigma|}$

# Haeupler-Shahrasbi strategy

- Index with string $S = S_1, \ldots, S_n \in [c]^n$, w. $c = O(1) \ll n$
  - $E: \Sigma^k \to \Sigma^n \quad \to \quad E': \Sigma^k \to (\Sigma \times [c])^n$
    - $E'(x)_i = (E(x)_i, S_i)$
    - $Rate(E') = Rate(E) - \frac{\log c}{\log |\Sigma|}$
- No longer have $i \neq j \Rightarrow S_i \neq S_j$
- What properties of $S$ useful? Attainable? Verifiable/Constructible?

# Synchronization Strings: Defn.

- $\epsilon$-self-matching:
  - $M = \{(i_t, j_t) | \ 1 \leq t \leq m\} \subseteq [n] \times [n]$ is $S$-matching if
    - Matching: $i_1, \ldots, i_m$ distinct, $j_1, \ldots, j_m$ distinct
    - Non-trivially $S$-valid: $S_{i_t} = S_{j_t}$ but $i_t \neq j_t \ \forall t$
    - Monotone: $i_a < i_b \ \Rightarrow j_a < j_b$
  - $S$ $\epsilon$-synch. string if $\forall$ $S$-matching $M$, $|M| \leq \epsilon \cdot n$



# Synchronization Strings

- Thm: $\forall \epsilon > 0 \ \exists \, c < \infty \, \forall n \ \exists \, S \in [c]^n$ $\epsilon$-synch. string.

# Synchronization Strings

- Thm: $\forall \epsilon > 0 \ \exists \, c < \infty \, \forall n \ \exists \, S \in [c]^n$ $\epsilon$-synch. string.

- Random string is $\epsilon$-synch. string w.h.p.
- Can verify if $S$ is $\epsilon$-synch. string in poly time.
- [HS'17]+followups: Poly time explicit constructions.

# Synchronization Strings

- Thm: $\forall \epsilon > 0 \ \exists \, c < \infty \, \forall n \ \exists \, S \in [c]^n$ $\epsilon$-synch. string.

- Random string is $\epsilon$-synch. string w.h.p.
- Can verify if $S$ is $\epsilon$-synch. string in poly time.
- [HS'17]+followups: Poly time explicit constructions.

- How do they help correct decoding errors?

# List-decoding + Synchronization Strings

- Ingredient: List-recoverable codes.

# List-decoding + Synchronization Strings

- Ingredient: List-recoverable codes.
  - $C \subseteq \Sigma^n$ is $(\delta, L)$-list-decodable if

    for every $y = (y_1, \ldots, y_n) \in \Sigma^n$

    if $S = \{x \in C | \ \#\{i | \ x_i \neq y_i\} \leq \delta.n\}$,

    then $|S| \leq L$

    (and $S$ can be computed efficiently given $y$)

# List-decoding + Synchronization Strings

- Ingredient: List-recoverable codes.
  - $C \subseteq \Sigma^n$ is $(\delta, L)$-list-decodable if

    for every $y = (y_1, \ldots, y_n) \in \Sigma^n$

    if $S = \{x \in C | \#\{i | x_i \neq y_i\} \leq \delta.n\}$,

    then $|S| \leq L$

    (and $S$ can be computed efficiently given $y$)
  - $C \subseteq \Sigma^n$ is $(\ell, \delta, L)$-list-recoverable: if

    for every $Y_1 \times \cdots \times Y_n \subseteq \Sigma^n$ s.t. $|Y_i| \leq \ell$

    if $S = \{x \in C | \#\{i | x_i \notin Y_i\} \leq \delta.n\}$,

    then $|S| \leq L$

    ($S$ can be computed efficiently given $Y_1, \ldots, Y_n$)

# List-decoding + Synchronization Strings

- Ingredient: List-recoverable codes.
  - $C \subseteq \Sigma^n$ is $(\ell, \delta, L)$-list-recoverable: if

    for every $Y_1 \times \cdots \times Y_n \subseteq \Sigma^n$ s.t. $|Y_i| \leq \ell$

    if $S = \{x \in C | \ \#\{i | \ x_i \notin Y_i\} \leq \delta.n\}$,

    then $|S| \leq L$

    ($S$ can be computed efficiently given $Y_1, \ldots, Y_n$)

# List-decoding + Synchronization Strings

- Ingredient: List-recoverable codes.
  - $C \subseteq \Sigma^n$ is $(\ell, \delta, L)$-list-recoverable: if
    for every $Y_1 \times \cdots \times Y_n \subseteq \Sigma^n$ s.t. $|Y_i| \leq \ell$
    if $S = \{x \in C | \#\{i | x_i \notin Y_i\} \leq \delta.n\}$,
    then $|S| \leq L$
    ($S$ can be computed efficiently given $Y_1, \ldots, Y_n$)
  - Theorem [Guruswami-Rudra'06]+followups: $\forall \delta, \epsilon > 0, \ell \; \exists \; \Sigma, L$ s.t. $\exists$ a family of $(\ell, \delta, L)$-list-recoverable codes of rate $1 - \delta - \epsilon$
    - "Folded-Reed-Solomon" codes + Guruswami-Indyk alphabet reduction.

# List-decodable codes for editing errors

- Theorem: Let $E: \Sigma^k \to \Sigma^n$ be $(\ell, \delta + \epsilon, L)$-list-recoverable. Let $S \in [c]^n$ be $\epsilon'$-synch string. Then $E': \Sigma^k \to (\Sigma \times [c])^n$ given by $E'(x)_i = (E(x)_i, S_i)$ is $(\delta, \gamma)$-list-decodable-code.

  (For $\ell = \frac{2(1+\gamma)}{\epsilon}$ & $\epsilon' = \frac{\epsilon}{2\ell}$)

# List-decodable codes for editing errors

- Theorem: Let $E: \Sigma^k \to \Sigma^n$ be $(\ell, \delta + \epsilon, L)$-list-recoverable. Let $S \in [c]^n$ be $\epsilon'$-synch string. Then $E': \Sigma^k \to (\Sigma \times [c])^n$ given by $E'(x)_i = (E(x)_i, S_i)$ is $(\delta, \gamma)$-list-decodable-code.

  (For $\ell = \frac{2(1+\gamma)}{\epsilon}$ & $\epsilon' = \frac{\epsilon}{2\ell}$)

- Proof by Algorithm: Given $(a_i, b_i)_{i \in [m]}$, $a_i \in \Sigma, b_i \in [c]$:

# List-decodable codes for editing errors

- Theorem: Let $E: \Sigma^k \to \Sigma^n$ be $(\ell, \delta + \epsilon, L)$-list-recoverable. Let $S \in [c]^n$ be $\epsilon'$-synch string. Then $E': \Sigma^k \to (\Sigma \times [c])^n$ given by $E'(x)_i = (E(x)_i, S_i)$ is $(\delta, \gamma)$-list-decodable-code.

  (For $\ell = \frac{2(1+\gamma)}{\epsilon}$ & $\epsilon' = \frac{\epsilon}{2\ell}$)

- Proof by Algorithm: Given $(a_i, b_i)_{i \in [m]}$, $a_i \in \Sigma, b_i \in [c]$:
  - Let $B = (b_1, \ldots, b_m)$ ; $Y_1 = \cdots = Y_n = \emptyset$

# List-decodable codes for editing errors

- Theorem: Let $E:\Sigma^k \to \Sigma^n$ be $(\ell, \delta+\epsilon, L)$-list-recoverable. Let $S \in [c]^n$ be $\epsilon'$-synch string. Then $E':\Sigma^k \to (\Sigma \times [c])^n$ given by $E'(x)_i = (E(x)_i, S_i)$ is $(\delta, \gamma)$-list-decodable-code.

  (For $\ell = \frac{2(1+\gamma)}{\epsilon}$ & $\epsilon' = \frac{\epsilon}{2\ell}$)

- Proof by Algorithm: Given $(a_i, b_i)_{i\in[m]}$, $a_i \in \Sigma, b_i \in [c]$:
  - Let $B = (b_1, \dots, b_m)$ ; $Y_1 = \cdots = Y_n = \emptyset$
  - For $\ell$ iterations do:
    - Let $M$ be largest monotone matching between $B$ and $S$
    - Removed matched part from $B$; add matched $a$-symbols into $Y_i$s.
      - $b_j \leftrightarrow S_i \Rightarrow Y_i \leftarrow Y_i \cup \{a_j\}$

# List-decodable codes for editing errors

- Theorem: Let $E:\Sigma^k \to \Sigma^n$ be $(\ell, \delta+\epsilon, L)$-list-recoverable. Let $S \in [c]^n$ be $\epsilon'$-synch string. Then $E':\Sigma^k \to (\Sigma \times [c])^n$ given by $E'(x)_i = (E(x)_i, S_i)$ is $(\delta, \gamma)$-list-decodable-code.

  (For $\ell = \frac{2(1+\gamma)}{\epsilon}$ & $\epsilon' = \frac{\epsilon}{2\ell}$)

- Proof by Algorithm: Given $(a_i, b_i)_{i\in[m]}$, $a_i \in \Sigma, b_i \in [c]$:
  - Let $B = (b_1, \dots, b_m)$ ; $Y_1 = \cdots = Y_n = \emptyset$
  - For $\ell$ iterations do:
    - Let $M$ be largest monotone matching between $B$ and $S$
    - Removed matched part from $B$; add matched $a$-symbols into $Y_i$s.
      - $b_j \leftrightarrow S_i \Rightarrow Y_i \leftarrow Y_i \cup \{a_j\}$
  - List-Recover from $Y_1, \dots, Y_n$

# Analysis of Algorithm

- Say transmitted $E'(x)$. Received $\left(a_j, b_j\right)_{j \in [m]}$
- $i =$ "error" if $E(x)_i \notin Y_i$.

# Analysis of Algorithm

- Say transmitted $E'(x)$. Received $\left(a_j, b_j\right)_{j \in [m]}$
- $i =$ "error" if $E(x)_i \notin Y_i$.
- $\delta n$ deletions $\Rightarrow \delta n$ errors. Any others?

# Analysis of Algorithm

- Say transmitted $E'(x)$. Received $\left(a_j, b_j\right)_{j \in [m]}$
- $i =$ "error" if $E(x)_i \notin Y_i$.
- $\delta n$ deletions $\Rightarrow \delta n$ errors. Any others?
- Say $E'(x)_i = (a_j, b_j)$ is not deleted. Why is $E(x)_i \notin Y_i$?

# Analysis of Algorithm

- Say transmitted $E'(x)$. Received $(a_j, b_j)_{j \in [m]}$
- $i =$ "error" if $E(x)_i \notin Y_i$.
- $\delta n$ deletions $\Rightarrow \delta n$ errors. Any others?
- Say $E'(x)_i = (a_j, b_j)$ is not deleted. Why is $E(x)_i \notin Y_i$?
  - Case 1: $b_j$ matched to $S_{i'}$ for $i' \neq i$:
    - $\epsilon'$-synch string $\Rightarrow \epsilon' n$ such errors per iteration.
    - $\ell$-iterations $\Rightarrow \ell \epsilon' n \leq \frac{\epsilon n}{2}$ such errors.

# Analysis of Algorithm

- Say transmitted $E'(x)$. Received $(a_j, b_j)_{j \in [m]}$
- $i =$ "error" if $E(x)_i \notin Y_i$.
- $\delta n$ deletions $\Rightarrow \delta n$ errors. Any others?
- Say $E'(x)_i = (a_j, b_j)$ is not deleted. Why is $E(x)_i \notin Y_i$?
  - Case 1: $b_j$ matched to $S_{i'}$ for $i' \neq i$:
    - $\epsilon'$-synch string $\Rightarrow \epsilon' n$ such errors per iteration.
    - $\ell$-iterations $\Rightarrow \ell \epsilon' n \leq \frac{\epsilon n}{2}$ such errors.
  - Case 2: $b_j$ unmatched at end:
    - Let $\alpha n$ code symbols be unmatched at end. Then each iteration matched $\geq \alpha n$ symbols. So $\ell \alpha n \leq m \leq (1+\gamma)n \Rightarrow \alpha n \leq \frac{(1+\gamma)n}{\ell} \leq \frac{\epsilon n}{2}$

# Analysis of Algorithm

- Say transmitted $E'(x)$. Received $(a_j, b_j)_{j \in [m]}$
- $i =$ "error" if $E(x)_i \notin Y_i$.
- $\delta n$ deletions $\Rightarrow \delta n$ errors. Any others?
- Say $E'(x)_i = (a_j, b_j)$ is not deleted. Why is $E(x)_i \notin Y_i$?
  - Case 1: $b_j$ matched to $S_{i'}$ for $i' \neq i$:
    - $\epsilon'$-synch string $\Rightarrow \epsilon' n$ such errors per iteration.
    - $\ell$-iterations $\Rightarrow \ell \epsilon' n \leq \frac{\epsilon n}{2}$ such errors.
  - Case 2: $b_j$ unmatched at end:
    - Let $\alpha n$ code symbols be unmatched at end. Then each iteration matched $\geq \alpha n$ symbols.

      So $\ell \alpha n \leq m \leq (1+\gamma)n \Rightarrow \alpha n \leq \frac{(1+\gamma)n}{\ell} \leq \frac{\epsilon n}{2}$

# Aside: List-Recoverable Codes

- Recall: "Theorem [Guruswami-Rudra'06]+followups: $\forall \delta, \epsilon > 0, \ell \exists \Sigma, L$ s.t. $\exists$ a family of $(\ell, \delta, L)$-list-recoverable codes of rate $1 - \delta - \epsilon$"
- Where do they come from?
  - 1. Folded-Reed Solomon Codes [GuRu'06]
    - Core ingredient
  - 2. Simplified decoding [Gu-Wang'11]
    - Enabled next step
  - 3. Strong analysis [Kopparty,Ron-Zewi,Saraf,Wootters'19]
    - Better analysis of list-size
  - 4. "AEL/GI"-style concatenation [pre-2000].
    - Reduces alphabet size to $O(1)$

# Folded-Reed Solomon Codes

- Specified by $\mathbb{F}_q, k, n, \gamma, \alpha_1, \ldots, \alpha_n$; $\alpha_i'$s distinct
- Message: $P(X) \in \mathbb{F}_q^{<k}[X]$
- Encoding: $\left\langle \left\langle P(\gamma^j \alpha_i) \right\rangle_{j \in [r]} \right\rangle_{i \in [n]} \in \left(\mathbb{F}_q^r\right)^n$ ;
- Rate $= \frac{k}{n'}$ where $n' \approx \left|\left\{\gamma^j \alpha_i \middle| \, i \in [n], j \in [r]\right\}\right| \leq nr$

# List-recovery Problem + Algorithm

- Problem: Given $\{(\beta_i, \lambda_{i1}, \dots, \lambda_{ir})\} \subseteq \mathbb{F}_q^{r+1}$ find all polynomials $P \in \mathbb{F}_q^{<k}[X]$, s.t. $\left|\left\{i \middle| \forall j\ P\left(\gamma^j \beta_i\right) = \lambda_{ij}\right\}\right| \geq t$
- Algorithm [Guruswami-Wang]:
  - Find any non-zero $Q(X, Y_1, \dots, Y_r)$ of deg 1 in $Y$ such that $Q(\beta_i, \lambda_{i1}, \dots, \lambda_{ir}) = 0$ for all $i$
  - Find all $P$ s.t. $Q\left(X, P(\gamma X), \dots, P(\gamma^r X)\right) \equiv 0$
- Efficiency: Both steps linear algebra!
- Correctness: Exercise. ☺

# Further work

- Editing + Interaction errors!
- Binary codes for edit distance: Positive rate limits known. [Guruswami-Haeupler-Shahrasbi]
- $R\ vs.\ \delta\ vs\ \gamma$ for binary codes?

# Thank You!