Efficiently decodable codes for the binary deletion channel

Venkatesan Guruswami (venkatg@cs.cmu.edu)
Ray Li * (rayyli@stanford.edu)

Carnegie Mellon University

August 18, 2017
Outline for section 1

1. Introduction

2. Binary deletion channel background

3. Construction ingredients

4. Construction

5. Open questions
Deletion channel

\[ m = 01 \rightarrow \text{Alice} \]
Deletion channel

\[ m = 01 \rightarrow \text{Alice} \rightarrow \text{Bob} \]
Deletion channel

\[ m = 01 \rightarrow \text{Alice} \quad \text{Channel} \quad \text{Bob} \]
Deletion channel

\[ m = 01 \rightarrow \text{Alice} \rightarrow_{01} \text{Channel} \rightarrow_{1} \text{Bob} \]
Deletion channel

\[ m = 01 \rightarrow \text{Alice} \rightarrow_{01} \text{Channel} \rightarrow_{1} \text{Bob} \rightarrow m = 11 \\
\quad m = 10 \\
\quad m = 01 \]
Deletion channel

\[ m = 01 \rightarrow \text{Alice} \rightarrow_{01} \text{Channel} \rightarrow_{1} \text{Bob} \rightarrow \begin{array}{c} m = 11 \\ m = 10 \\ m = 01 \end{array} \]

<table>
<thead>
<tr>
<th>Error type</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Substitution</td>
<td>000</td>
<td>001</td>
</tr>
<tr>
<td>Erasure</td>
<td>000</td>
<td>00?</td>
</tr>
<tr>
<td>Deletion</td>
<td>000</td>
<td>00</td>
</tr>
</tbody>
</table>
Error correcting codes

\[ m = 01 \rightarrow \]

Alice (Enc)

Bob (Dec)

Note: "Order" of redundancy matters for deletions.

\[ m \mapsto 01 \rightarrow 010101, \quad 10 \mapsto 101010. \]
Error correcting codes

\[ m = 01 \rightarrow \text{Alice (Enc)} \rightarrow \text{Channel} \rightarrow \text{Bob (Dec)} \]

Note:
- The "order" of redundancy matters for deletions.
- \( m \mapsto \rightarrow m \) fails for 1 deletion (01 \( \mapsto \rightarrow 010101 \), 10 \( \mapsto \rightarrow 101010 \)).
Error correcting codes

\[ m = 01 \rightarrow \begin{array}{c}
\text{Alice} \\
(\text{Enc})
\end{array} \rightarrow \begin{array}{c}
\text{Channel}
\end{array} \rightarrow \begin{array}{c}
\text{Bob} \\
(\text{Dec})
\end{array} \]

\[ 000111 \rightarrow 00011 \]

Note: "Order" of redundancy matters for deletions. \[ m \mapsto \rightarrow \]

V. Guruswami and R. Li (CMU)
Error correcting codes

\[ m = 01 \rightarrow \text{Alice (Enc)} \xrightarrow{000111} \text{Channel} \xrightarrow{00011} \text{Bob (Dec)} \rightarrow m = 01 \]

Note: “Order” of redundancy matters for deletions. \[ m \mapsto \text{fails for 1 deletion (01} \mapsto \text{010101, 10} \mapsto \text{101010).} \]
Error correcting codes

$m = 01 \rightarrow$ Alice (Enc) $\rightarrow$ Channel $\rightarrow$ Bob (Dec) $\rightarrow m = 01$

Tradeoff between “redundancy” and “robustness” of the code.

Efficient construction, encoding, decoding.
Error correcting codes

\[ m = 01 \rightarrow \text{Alice (Enc)} \rightarrow 000111 \rightarrow \text{Channel} \rightarrow 00011 \rightarrow \text{Bob (Dec)} \rightarrow m = 01 \]

Tradeoff between “redundancy” and “robustness” of the code.

Efficient construction, encoding, decoding. \( \leftarrow \) This work
Error correcting codes

Tradeoff between “redundancy” and “robustness” of the code.

Efficient construction, encoding, decoding. ← This work

Note: “Order” of redundancy matters for deletions.

\[ m \mapsto mmm \] fails for 1 deletion (01 \mapsto 010101, 10 \mapsto 101010).
Error correcting codes: Notation

$m = 01 \rightarrow \text{Alice (Enc)} \rightarrow \text{Channel} \rightarrow \text{Bob (Dec)} \rightarrow m = 01$

Alphabet: $\Sigma$ (e.g. $\{0, 1\}$)

(Block) length: $n, m, N$

Codeword: $c \in \Sigma^n$
Error correcting codes: Notation

\[ m = 01 \rightarrow \text{Alice (Enc)} \rightarrow \text{Channel} \rightarrow \text{Bob (Dec)} \rightarrow m = 01 \]

Alphabet: \( \Sigma \) (e.g. \( \{0, 1\} \))

(Block) length: \( n, m, N \)

Codeword: \( c \in \Sigma^n \)

Rate

- \( R = \frac{\log(\# \text{messages})}{N} \in (0, 1) \).
- \( R \) is proportion of non-redundant symbols
- Want families of codes (implicitly \( N \to \infty \))
Binary deletion channel

\[ m = 01 \rightarrow \text{Alice (Enc)} \rightarrow \text{Channel} \rightarrow \text{Bob (Dec)} \rightarrow m = 01 \]

- Adversarial: \# deletions fixed \((pn)\), decoding 100% success
- Random: i.i.d deletions, decoding success w.h.p.
Binary deletion channel

\[ m = 01 \rightarrow \text{Alice (Enc)} \rightarrow \text{Channel} \rightarrow \text{Bob (Dec)} \rightarrow m = 01 \]

- Adversarial: \# deletions fixed \((pn)\), decoding 100% success
- Random: i.i.d deletions, decoding success w.h.p. ← This work

**Definition**

For \( p \in (0, 1) \), the **binary deletion channel with deletion probability** \( p \) \((BDC_p)\) deletes bits of binary strings independently w.p. \( p \).
Outline for section 2

1. Introduction
2. Binary deletion channel background
3. Construction ingredients
4. Construction
5. Open questions
Capacity

**Definition**

Capacity $C_{BDC}(p) = \sup\{R : \text{exists rate } R \text{ code family against } BDC_p\}$.

**Question.** What is the capacity of a binary deletion channel with deletion probability $p$?
Capacity

Definition

Capacity \( C_{BDC}(p) = \sup\{R : \text{exists rate } R \text{ code family against } \text{BDC}_p\} \).

Question. What is the capacity of a binary deletion channel with deletion probability \( p \)?

- Binary Symmetric Channel (BSC): Well understood
- Binary Erasure Channel (BEC): Well understood
- Binary Deletion Channel (BDC): Don’t know capacity

<table>
<thead>
<tr>
<th>Error type</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Substitution</td>
<td>000</td>
<td>001</td>
</tr>
<tr>
<td>Erasure</td>
<td>000</td>
<td>00?</td>
</tr>
<tr>
<td>Deletion</td>
<td>000</td>
<td>00</td>
</tr>
</tbody>
</table>
Capacity

**Definition**

Capacity $C_{BDC}(p) = \sup \{ R : \exists \text{ rate } R \text{ code family against } \text{BDC}_p \}$. 

**Question.** What is the capacity of a binary deletion channel with deletion probability $p$?

- Binary Symmetric Channel (BSC): Well understood
- Binary Erasure Channel (BEC): Well understood
- Binary Deletion Channel (BDC): Don’t know capacity

<table>
<thead>
<tr>
<th>Error type</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Substitution</td>
<td>000</td>
<td>001</td>
</tr>
<tr>
<td>Erasure</td>
<td>000</td>
<td>00?</td>
</tr>
<tr>
<td>Deletion</td>
<td>000</td>
<td>00</td>
</tr>
</tbody>
</table>

Aside: Adversarial deletions, unknown if capacity is 0 for $p \in [\sqrt{2} - 1, \frac{1}{2})$. 
Existing bounds on BDC\(_p\) capacity

**Definition**

Capacity \( C_{BDC}(p) = \sup \{ \mathcal{R} : \text{exists rate } \mathcal{R} \text{ code family against BDC}_p \} \).

Recall: \( p \in [0, 1] \), \( H(p) = p \log \frac{1}{p} + (1 - p) \log \frac{1}{1-p} \).
Existing bounds on $\text{BDC}_p$ capacity

**Definition**

Capacity $\mathcal{C}_{\text{BDC}}(p) = \sup\{\mathcal{R} : \text{exists rate } \mathcal{R} \text{ code family against } \text{BDC}_p\}$.

Recall: $p \in [0, 1]$, $H(p) = p \log \frac{1}{p} + (1 - p) \log \frac{1}{1-p}$.

- **Lower bounds**
  - $1 - 2H(p)$ (adversarial bound)
  - $1 - H(p)$ [Gallager ’61, Zigangirov ’69]
  - $(1 - p)/9$ [Drinea, Mitzenmacher ’06]

- **Upper bounds**
  - $1 - H(p)$ as $p \to 0$ [Kalai, Mitzenmacher, Sudan ’10]
  - $1 - p$ (BEC Capacity)
  - $.4143(1 - p)$ if $p \geq 0.65$ [Rahmati, Duman ’15]
  - Numerical bounds [Fertonani, Duman ’10]

Capacity understood as $p \to 0 (1 - H(p))$, $p \to 1 (\alpha(1 - p))$. 

---

V. Guruswami and R. Li (CMU)  Efficiently decodable codes for the BDC  August 18, 2017  9 / 31
Existing bounds on $BDC_p$ capacity

**Definition**

Capacity $C_{BDC}(p) = \sup\{R : \text{exists rate } R \text{ code family against } BDC_p\}$.

Recall: $p \in [0, 1]$, $H(p) = p \log \frac{1}{p} + (1 - p) \log \frac{1}{1-p}$.

- **Lower bounds**
  - $1 - 2H(p)$ (adversarial bound)
  - $1 - H(p)$ [Gallager '61, Zigangirov '69]
  - $(1 - p)/9$ [Drinea, Mitzenmacher '06]

- **Upper bounds**
  - $1 - H(p)$ as $p \to 0$ [Kalai, Mitzenmacher, Sudan '10]
  - $1 - p$ (BEC Capacity)
  - $.4143(1 - p)$ if $p \geq 0.65$ [Rahmati, Duman '15]
  - Numerical bounds [Fertonani, Duman '10]

Capacity understood as $p \to 0$ ($1 - H(p)$), $p \to 1$ ($\alpha(1 - p)$).

$(1 - p)/9$ result is non-constructive.
Existing bounds on $BDC_p$ capacity

- **Lower bounds**
  - $1 - H(p)$ [Gallager '61, Zigangirov '69]
  - $(1 - p)/9$ [Drinea, Mitzenmacher '06]

- **Upper bounds**
  - $0.4143(1 - p)$ if $p \geq 0.65$ [Rahmati, Duman '15]
  - **Numerical bounds** [Fertonani, Duman '10]

*Plot made with Mathematica*
Theorem (Guruswami, Li ’17)

Let $p \in (0, 1)$. There is a constant $\alpha > 0$ and an explicit a family of binary codes that

- has rate $\alpha(1 - p)$,
- is constructible, encodable, decodable in poly time on $BDC_p$. 
### New algorithmic result

<table>
<thead>
<tr>
<th>Theorem (Guruswami, Li ’17)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let $p \in (0, 1)$. <em>There is a constant</em> $\alpha &gt; 0$ <em>and an explicit a family of binary codes that</em></td>
</tr>
<tr>
<td>- <em>has rate</em> $\alpha(1 - p)$, $(\alpha = \frac{1}{110})$</td>
</tr>
<tr>
<td>- <em>is constructible, encodable, decodable in poly time on</em> $BDC_p$.</td>
</tr>
</tbody>
</table>
New algorithmic result

Theorem (Guruswami, Li ’17)

Let $p \in (0, 1)$. There is a constant $\alpha > 0$ and an explicit family of binary codes that

- has rate $\alpha(1 - p)$, $(\alpha = \frac{1}{110})$
- is constructible, encodable, decodable in poly time on $BDC_p$.

Throughout the talk, think of $p$ close to 1.
Outline for section 3

1. Introduction
2. Binary deletion channel background
3. Construction ingredients
   - Concatenated codes
   - Haeupler-Shahrasbi code
4. Construction
5. Open questions
Ingredient 1: Concatenated codes

Example

\[
\text{Enc}_{\text{out}}(m) = aabc \in C_{\text{out}}
\]
\[
\text{Enc}_{\text{in}}(\cdot): a \mapsto 0011, \ b \mapsto 1100, \ c \mapsto 0101
\]
\[
\text{Enc}(m) = 0011001111000101 \in C
\]
Ingredient 1: Concatenated codes

- $C_{\text{out}}$: alphabet $K$, length $n$
- $C_{\text{in}}$: alphabet $k$, length $m$
- $C$: alphabet $k$, length $N = mn$
  - Message encoded into $c = \sigma_1 \sigma_2 \ldots \sigma_n \in C_{\text{out}}$
  - Each $\sigma_i$ encoded into codeword of $C_{\text{in}}$
- $R = R_{\text{out}} \cdot R_{\text{in}}$

\[ \begin{array}{cccc}
\sigma_1 & \sigma_2 & \cdots & \sigma_n \\
C_{\text{in}}(\sigma_1) & C_{\text{in}}(\sigma_2) & \cdots & C_{\text{in}}(\sigma_n) \\
\end{array} \]

$\in C_{\text{out}}$

$\in C$
Ingredient 2: What do we want in an outer code?

- Rate close to 1
- Tolerate constant fraction of adversarial insertions and deletions
- Fast construction, encoding, decoding
Ingredient 2: Haeupler-Shahrasbi code (STOC ’17)

- $|\Sigma| = \text{poly}(1/\epsilon)$,
- Rate $0.999 - \epsilon$,
- Corrects $0.001n$ adversarial insertions and deletions
- Construction time: $\text{poly}(n)$
- Encoding time: $O(n)$
- Decoding time: $O(n^2)$
Outline for section 4

1. Introduction

2. Binary deletion channel background

3. Construction ingredients

4. Construction
   - First attempts
   - Our construction
   - Remarks on construction

5. Open questions
Theorem (Guruswami, Li ’17)

Let $p \in (0, 1)$. There is a constant $\alpha > 0$ and an explicit $a$ family of binary codes that

- has rate $\alpha (1 - p)$,
- is constructible, encodable, decodable in polynomial time on $BDC_p$. 

Attempt 1: Vanilla concatenation

\( C_{out} \) is Haeupler-Shahrasbi code.
\( C_{in} \in \{0, 1\}^m \) robust against BDC\(_p\)

\[ C_{in}(\sigma_1) \quad C_{in}(\sigma_2) \quad \ldots \quad C_{in}(\sigma_n) \]

- Rate \( \approx 0.999R_{in} \)
- Outer code decodes in \( O(n^2) \) time
Attempt 1: Vanilla concatenation

$C_{out}$ is Haeupler-Shahrasbi code.

$C_{in} \in \{0, 1\}^m$ robust against BDC$_p$ ($m = \text{poly}(1/\epsilon)$)

- Rate $\approx 0.999 R_{in}$
- Outer code decodes in $O(n^2)$ time
- Inner code decodes in constant time
Attempt 1: Vanilla concatenation

$C_{out}$ is Haeupler-Shahrasbi code.

$C_{in} \in \{0, 1\}^m$ robust against BDC$_p$ ($m = \text{poly}(1/\epsilon)$)

- Rate $\approx 0.999 R_{in}$
- Outer code decodes in $O(n^2)$ time
- Inner code decodes in \textbf{constant time}
- Issue: where do inner codewords start?
Attempt 1: Vanilla concatenation

\( C_{\text{out}} \) is Haeupler-Shahrasbi code.
\( C_{\text{in}} \in \{0, 1\}^m \) robust against BDC\(_p\) \((m = \text{poly}(1/\epsilon))\)

- Rate \( \approx 0.999R_{\text{in}} \)
- Outer code decodes in \( O(n^2) \) time
- Inner code decodes in **constant time**
- Issue: where do inner codewords start?
Attempt 1: Vanilla concatenation

$C_{out}$ is Haeupler-Shahrasbi code.

$C_{in} \in \{0, 1\}^m$ robust against BDC$_p$ ($m = \text{poly}(1/\epsilon)$)

- Rate $\approx 0.999 R_{in}$
- Outer code decodes in $O(n^2)$ time
- Inner code decodes in constant time
- Issue: where do inner codewords start?

Diagram:

```
\begin{tikzpicture}
  \node (m) at (0,0) {m};
  \node (s1) at (-2,-1) {$\sigma_1$};
  \node (c1) at (-3,-2) {$C_{in}(\sigma_1)$};
  \node (s2) at (0,-1) {$\sigma_2$};
  \node (c2) at (-1,-2) {$C_{in}(\sigma_2)$};
  \node (sn) at (2,-1) {$\sigma_n$};
  \node (cn) at (1,-2) {$C_{in}(\sigma_n)$};
  \draw (m) -- (s1);
  \draw (s1) -- (c1);
  \draw (m) -- (s2);
  \draw (s2) -- (c2);
  \draw (m) -- (sn);
  \draw (sn) -- (cn);
\end{tikzpicture}
```
Attempt 2: Concatenation with buffers

$C_{out}$ is Haeupler-Shahrasbi code.

$C_{in} \in \{0, 1\}^m$ robust against $BDC_p$ ($m = \text{poly}(1/\epsilon)$)

Decoding

- Identify “decoding buffers” as long runs of 0s
- Inner decode the “decoding windows” in between the decoding buffers
Attempt 2: Concatenation with buffers

$C_{out}$ is Haeupler-Shahrasbi code.

$C_{in} \in \{0, 1\}^m$ robust against $\text{BDC}_p$ ($m = \text{poly}(1/\epsilon)$)

Decoding

- Identify “decoding buffers” as long runs of 0s
- Inner decode the “decoding windows” in between the decoding buffers
- Bits of inner codewords not deleted according to $\text{BDC}_p$
- Cannot use DM06 $R = (1 - p)/9$ code as a black box inner code
Concatenation with buffers and duplication

Outer code. Haeupler-Shahrasbi code

Inner code. Word have runs of length 1 and 2 only, start and end with 1, chosen greedily, correct against $\delta n$ adversarial deletions.

$101100110101011001$

Buffer. $0.001m$ 0s between adjacent inner codewords.

Duplication. Duplicate each bit $B(p)$ times ($B(p) = 60/(1 - p)$)

$1 B^B 0 B^B 1^B 2B^B 0 B^B 0^B B^B 0 1^B 0 B^B 1 0 B^B 0^B 1^B 2B^B 0^B B^B$
Concatenation with buffers and duplication

Outer code. Haeupler-Shahrasbi code

Inner code. Word have runs of length 1 and 2 only, start and end with 1, chosen greedily, correct against $\delta n$ adversarial deletions.

$\delta \approx 0.008, R_{in} \approx 0.5$

101100110101011001

Buffer. $0.001m$ 0s between adjacent inner codewords.

Duplication. Duplicate each bit $B(p)$ times ($B(p) = 60/(1 - p)$)
Concatenation with buffers and duplication

Outer code. Haeupler-Shahrasbi code

Inner code. Word have runs of length 1 and 2 only, start and end with 1, chosen greedily, correct against $\delta n$ adversarial deletions. $\delta \approx 0.008, R_{in} \approx 0.5$

101100110101011001

Buffer. $0.001m$ 0s between adjacent inner codewords.

Duplication. Duplicate each bit $B(p)$ times ($B(p) = 60/(1 - p)$)

$1^B 0^B 1^2^B 0^2^B 1^2^B 0^B 1^B 0^B 1^0^B 0^1^B 1^2^B 0^2^B 1^B$

Rate $\approx 1 \times 0.5 \times (1 - p)/60 = \alpha(1 - p)$. 
Concatenation with buffers and duplication

Outer code. Haeuppler-Shahrasbi code
Inner code. Runs of length 1 and 2. Start and end with 1.

101100110101011001

Buffer. \( \eta m \) 0s between inner codewords.
Duplication. \( B = \frac{60}{1 - p} \).

\[
E[|BDC_p(1^B)|] = 60, \quad E[|BDC_p(1^{2B})|] = 120
\]

Idea: Decode \( 1^\alpha \) as \( \langle 1 \rangle \) for \( \alpha \leq 86 \), and \( \langle 11 \rangle \) for \( \alpha > 86 \)
Decoding algorithm

\[ C_{in}^{60}(\sigma_1) \ 0^{0.06m} \ C_{in}^{60}(\sigma_2) \ 0^{0.06m} \ \ldots \ 0^{0.06m} \ C_{in}^{60}(\sigma_n) \]

Decoding algorithm.

- Runs of at least 0.03m zeros (decoding buffers) divide word into decoding windows
Decoding algorithm

\[
\begin{align*}
C_{in}^{60}(\sigma_1) & \quad 0^{0.06m} \quad C_{in}^{60}(\sigma_2) & \quad 0^{0.06m} & \ldots & \quad 0^{0.06m} \quad C_{in}^{60}(\sigma_n)
\end{align*}
\]

**Decoding algorithm.**

- Runs of at least 0.03\( m \) zeros (*decoding buffers*) divide word into **decoding windows**
Decoding algorithm

| $C_{in}^{60}(\sigma_1)$ | $0.06^m$ | $C_{in}^{60}(\sigma_2)$ | $0.06^m$ | ... | $0.06^m$ | $C_{in}^{60}(\sigma_n)$ |

Decoding algorithm.

- Runs of at least 0.03m zeros (decoding buffers) divide word into decoding windows
- Deduplicate the runs: $b^\alpha$ as $\langle b \rangle$ for $\alpha \leq 86$, and $\langle bb \rangle$ for $\alpha > 86$
Decoding algorithm

\[
C_{in}(\sigma_1) \, 0^{0.06m} \, C_{in}(\sigma_2) \, 0^{0.06m} \, \ldots \, 0^{0.06m} \, C_{in}(\sigma_n)
\]

Decoding algorithm.

- Runs of at least 0.03m zeros (decoding buffers) divide word into decoding windows
- Deduplicate the runs: \( b^\alpha \) as \( \langle b \rangle \) for \( \alpha \leq 86 \), and \( \langle bb \rangle \) for \( \alpha > 86 \)
Decoding algorithm

| $C_{in}(\sigma_1)$ | $0^{0.06m}$ | $C_{in}(\sigma_2)$ | $0^{0.06m}$ | ... | $0^{0.06m}$ | $C_{in}(\sigma_n)$ |

Decoding algorithm.

- Runs of at least $0.03m$ zeros (decoding buffers) divide word into decoding windows.
- Deduplicate the runs: $b^\alpha$ as $\langle b \rangle$ for $\alpha \leq 86$, and $\langle bb \rangle$ for $\alpha > 86$.
- For each decoding window recover outer symbol $\sigma'$ from $\text{Dec}_{in}$.
Decoding algorithm

Runs of at least $0.03m$ zeros (decoding buffers) divide word into decoding windows.

Deduplicate the runs: $b^\alpha$ as $\langle b \rangle$ for $\alpha \leq 86$, and $\langle bb \rangle$ for $\alpha > 86$.

For each decoding window recover outer symbol $\sigma'$ from $\text{Dec}_{in}$. 
Decoding algorithm

- Runs of at least $0.03m$ zeros (decoding buffers) divide word into decoding windows
- Deduplicate the runs: $b^\alpha$ as $\langle b \rangle$ for $\alpha \leq 86$, and $\langle bb \rangle$ for $\alpha > 86$
- For each decoding window recover outer symbol $\sigma'$ from $\text{Dec}_{in}$
- Use $\text{Dec}_{out}$ to decode $\sigma'_1 \sigma'_2 \ldots \sigma'_{n'}$ into message $m$. 
Decoding algorithm

Decoding algorithm.

- Runs of at least 0.03m zeros (decoding buffers) divide word into decoding windows.
- Deduplicate the runs: $b^\alpha$ as $\langle b \rangle$ for $\alpha \leq 86$, and $\langle bb \rangle$ for $\alpha > 86$.
- For each decoding window recover outer symbol $\sigma'$ from $\text{Dec}_{in}$.
- Use $\text{Dec}_{out}$ to decode $\sigma'_1 \sigma'_2 \ldots \sigma'_{n'}$ into message $m$. 
Decoding algorithm

Decoding algorithm.

- Runs of at least $0.03m$ zeros (decoding buffers) divide word into decoding windows $O(n)$
- Deduplicate the runs: $b^\alpha$ as $\langle b \rangle$ for $\alpha \leq 86$, and $\langle bb \rangle$ for $\alpha > 86$
  
  $n \times O(1)$
- For each decoding window recover outer symbol $\sigma'$ from Dec$_{in}$
  
  $n \times O(1)$
- Use Dec$_{out}$ to decode $\sigma'_1 \sigma'_2 \ldots \sigma'_{n'}$ into message $m$. $O(n^2)$

Total runtime: $O(n^2)$
What could go wrong? Deleted buffer

2 deletions, 1 insertion

\[
\begin{align*}
\sigma_1' & \quad C_{in}^{60}(\sigma_1) \quad 0.01m & \quad C_{in}^{60}(\sigma_2) \quad 0.06m & \cdots & \quad 0.06m \quad C_{in}^{60}(\sigma_n) \\
\sigma_{n-1}' & \quad m
\end{align*}
\]

Decoding algorithm.

- Runs of at least 0.03m zeros (decoding buffers) divide word into decoding windows
- Deduplicate the runs: \(b^\alpha\) as \(\langle b \rangle\) for \(\alpha \leq 86\), and \(\langle bb \rangle\) for \(\alpha > 86\)
- For each decoding window recover outer symbol \(\sigma'\) from Dec_{in}
- Use Dec_{out} to decode \(\sigma_1'\sigma_2'\ldots\sigma_{n'}'\) into message \(m\).
What could go wrong? Created buffer

2 insertions, 1 deletion

Decoding algorithm.

- Runs of at least 0.03m zeros (decoding buffers) divide word into decoding windows
- Deduplicate the runs: \( b^\alpha \) as \( \langle b \rangle \) for \( \alpha \leq 86 \), and \( \langle bb \rangle \) for \( \alpha > 86 \)
- For each decoding window recover outer symbol \( \sigma' \) from Dec\(_{in}\)
- Use Dec\(_{out}\) to decode \( \sigma'_1 \sigma'_2 \ldots \sigma'_{n'} \) into message m.
What could go wrong? Corrupted decoding window

1 insertion, 1 deletion

\[ \sigma_1' \quad \sigma_2' \quad \sigma_n' \]

\[ \begin{array}{c}
0^{0.06m} C_{in}^{60}(\sigma_2') \quad 0^{0.06m} \\
\vdots \\
0^{0.06m} C_{in}^{60}(\sigma_n')
\end{array} \]

Decoding algorithm.

- Runs of at least 0.03\(m\) zeros (decoding buffers) divide word into decoding windows.
- Deduplicate the runs: \(b^\alpha\) as \(\langle b \rangle\) for \(\alpha \leq 86\), and \(\langle bb \rangle\) for \(\alpha > 86\).
- For each decoding window recover outer symbol \(\sigma'\) from \(\text{Dec}_{in}\).
- Use \(\text{Dec}_{out}\) to decode \(\sigma_1'\sigma_2'\ldots\sigma_n'\) into message \(m\).
What could go wrong? Corrupted decoding window

1 insertion, 1 deletion

\[
\begin{array}{c}
\sigma'_1 \\
\times \\
\sigma'_2 \\
0^{0.06m}C_{in}^{60}(\sigma_2) \\
0^{0.06m} \\
\vdots \\
0^{0.06m}C_{in}^{60}(\sigma_n) \\
\sigma'_n
\end{array}
\]

Decoding algorithm.

- Runs of at least 0.03\(m\) zeros (decoding buffers) divide word into decoding windows.
- Deduplicate the runs: \(b^\alpha\) as \(\langle b \rangle\) for \(\alpha \leq 86\), and \(\langle bb \rangle\) for \(\alpha > 86\).
- For each decoding window recover outer symbol \(\sigma'\) from \(Dec_{in}\).
- Use \(Dec_{out}\) to decode \(\sigma'_1\sigma'_2\ldots\sigma'_{n'}\) into message \(m\).

Total number of ins/dels in \(\sigma'_1\ldots\sigma'_{n'}\) is < 0.001\(n\) w.h.p.
Remark: Rate improvement

- Rate is $\frac{1-p}{110}$. Can improve to $\frac{1-p}{60}$ if duplication is Poisson.

- No easy way to use $\frac{1-p}{9}$ directly as a black box inner code (else rate is $\frac{1-p}{9}$).
Remark: Alternative outer codes

- Reed Solomon code (encoding \((i, \alpha_i)\) into the inner code)
  Similar rate, Worse runtime, Inner code is \(\log n\) length

- High rate binary code efficiently decodable against insertions and deletions [Guruswami, Li '16]
  Worse rate and runtime

<table>
<thead>
<tr>
<th>Outer code</th>
<th>Error type</th>
<th>Inner len</th>
<th>Rate</th>
<th>Error frac</th>
<th>Decoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>HS '17</td>
<td>Ins/del</td>
<td>(c)</td>
<td>(1 - \epsilon)</td>
<td>(\Omega(\epsilon))</td>
<td>(O(N^2))</td>
</tr>
<tr>
<td>GL '16</td>
<td>Ins/del</td>
<td>(c)</td>
<td>(1 - \epsilon)</td>
<td>(\Omega(\epsilon^5))</td>
<td>(\text{poly}(N))</td>
</tr>
<tr>
<td>RS</td>
<td>Erase/Sub</td>
<td>(\Omega(\log n))</td>
<td>(1 - \epsilon)</td>
<td>(\Omega(\epsilon))</td>
<td>(O(N^3))</td>
</tr>
</tbody>
</table>
Outline for section 5

1. Introduction
2. Binary deletion channel background
3. Construction ingredients
4. Construction
5. Open questions
Open questions

- Capacity of the binary deletion channel

- Efficiently decodable codes for BDC with rate $\alpha(1 - p)$ for larger $\alpha$, perhaps $\alpha \geq 1/9$

- Efficiently decodable codes for BDC with rate $1 - O(H(p))$ for $p \to 0$

- Capacity of random channels applying insertions and deletions
Thank you!

Research supported in part by NSF grant CCF-1422045