An Improvement of Non-binary Code Correcting Single $b$-Burst of Insertions or Deletions

Toyohiko Saeki, Takayuki Nozaki

Yamaguchi University

ISITA2018
Oct. 29th, 2018
Background (1: Purpose)

**b-burst insertion/deletion channel**

Exactly $b$ consecutive insertions/deletions occur to $x$

Example: 3-burst deletion

$$x = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12})$$

$$y = (y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9)$$

Received word $y$ becomes the concatenation of remaining sequence

**Purpose of this research**

Constructing non-binary ($q$-ary) code correcting $b$-burst insertion/deletion
Background (2: Existing works, Contribution, Outline)

Existing works of I/D codes

<table>
<thead>
<tr>
<th></th>
<th>Binary</th>
<th>Non-binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single I/D</td>
<td>VT code [Varshamov1965] (proved by [Levenshtein1966])</td>
<td>qVT code [Tenengolts1984]</td>
</tr>
<tr>
<td>Burst I/D</td>
<td>[Cheng2014] Improve [Schoney2017-04]</td>
<td>[Schoney2017-10] Improve This work</td>
</tr>
</tbody>
</table>

Contributions of this research

- Constructing non-binary burst I/D code with larger cardinality
- Devising decoding algorithm for the code
- Evaluating number of codewords (Numerical and Asymptotic)

Outline

1. Existing works
2. Code construction
3. Number of Codewords
Existing Work (1: Single I/D codes)

**VT code: Binary single I/D code**

$n$: code length, \( a \in [n + 1] := \{0, 1, 2, \ldots, n\} \)

\[
\text{VT}_{a}(n) = \{ \mathbf{x} \in \{0, 1\}^{n} | \sum_{i=1}^{n} ix_i \equiv a \pmod{n + 1} \}.
\]

**Non-binary VT code: non-binary single I/D code**

$q$: number of alphabet, \( a \in [n], c \in [q] \)

\[
\text{qVT}_{a,c}(n, q) = \{ \mathbf{x} \in [q]^{n} | \sum_{i=1}^{n} x_i \equiv c \pmod{q}, \sigma(\mathbf{x}) \in \text{VT}_{a}(n - 1) \}.
\]

Ascent sequence \( \sigma(\mathbf{x}) \)

\[
\sigma : [q]^{n} \rightarrow \{0, 1\}^{n-1}
\]

\[
\sigma(\mathbf{x})_i = \begin{cases} 
1 & (x_i < x_{i+1}) \\
0 & (x_i \geq x_{i+1}) 
\end{cases}
\]

\[
\mathbf{x} = 30133212
\]

\[
\sigma(\mathbf{x}) = 0110001 \text{ ascent sequence}
\]
Existing Work (2: Burst I/D codes 1)

\(b\)-Interleave (Matrix representation of \(x\))

\[
A_b(x) = \begin{pmatrix}
x_1 & x_{b+1} & \cdots & x_{n-b+1} \\
x_2 & x_{b+2} & \cdots & x_{n-b+2} \\
\vdots & \vdots & \ddots & \vdots \\
x_b & x_{2b} & \cdots & x_n
\end{pmatrix}.
\]

Example: 3-interleaving \(x = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12})\)

\[
A_3(x) = \begin{pmatrix}
x_1 & x_4 & x_7 & x_{10} \\
x_2 & x_5 & x_8 & x_{11} \\
x_3 & x_6 & x_9 & x_{12}
\end{pmatrix}.
\]

- Each row contains single deletion
- The deletion position of 1st row gives deletion range of other rows
Existing Work (3: Burst I/D codes 2)

Detection of Deletion Position:
Even if we correct a deletion, we cannot detect deletion position

\[ \mathbf{x} = (1, 0, 0, 1, 1, 1) \]
\[ \mathbf{x}^{-2} = (1, 0, 1, 1, 1) \]
\[ \mathbf{x}^{-3} = (1, 0, 1, 1, 1) \]

The received words are same if a deletion occurs in a same run. One cannot detect which symbol in a run is deleted

- We need to limit the run length of 1st row
- We use a code correcting single deletion with given deletion range
Existing Work (4: Burst I/D codes 3)

Framework of $b$-burst deletion correcting code

\[ A_b(x) = \begin{cases} 
\text{Single I/D code with maximum run length } r \\
(r+1)\text{-bounded single deletion correcting code} \\
\vdots \\
(r+1)\text{-bounded single deletion correcting code} \\
n/b 
\end{cases} \]

$(r+1)$-bounded single deletion correcting code

Correcting single deletion with side information of deletion position

(Side information) $(r+1)$-consecutive deletion positions
Existing Work (5: Burst I/D codes 4)

\( b \)-burst I/D code [Schonery2017-04]

\[
C_{2,b} = \{ \mathbf{x} \in \{0, 1\}^n \mid A_b(\mathbf{x})_1 \in VT_a(n/b) \cap S_2(n/b, r), \\
A_b(\mathbf{x})_2 \in SVT_{d,e}(n/b, r + 1), \\
\vdots \\
A_b(\mathbf{x})_n \in SVT_{d,e}(n/b, r + 1) \}
\]

\( S_2(n/b, r) \): set of binary sequence of length \( n/b \) with maximum run length at most \( r \)

**Shifted VT (SVT) code**

\( r \)-bounded single deletion code

\[
SVT_{d,e}(n, r) = \{ \mathbf{x} \in \{0, 1\}^n \mid \sum_{i=1}^{n} ix_i \equiv d \pmod{r}, \\
\sum_{i=1}^{n} x_i \equiv e \pmod{2} \}
\]

d \in [r], e \in \{0, 1\}
Existing Work (6: Non-binary Burst I/D codes)

Non-binary SVT code

\[
q_{SVT_{d,e,f}}(n, r, q) := \{ \mathbf{x} \in [q]^n \mid \sum_{i=1}^{n} x_i \equiv f \pmod{q}, \\
\sigma(\mathbf{x}) \in SVT_{d,e}(n-1, r) \}
\]

[Schoney2017-11, Lemma 1] Correction capability of NB-SVT code

\(q_{SVT_{d,e,f}}(n, r, q)\) is \((r - 1)\)-bounded single deletion correcting code

Non-binary \(b\)-burst I/D code [Schoney2017-11]

\[
\tilde{C}_{q,b} = \{ \mathbf{x} \in [q]^n \mid A_{b}(\mathbf{x})_1 \in qVT_{a,b}(n/b, q) \cap S_q(n/b, r), \\
A_{b}(\mathbf{x})_2 \in qSVT_{d,e,f}(n/b, r + 2, q), \\
\vdots \\
A_{b}(\mathbf{x})_b \in qSVT_{d,e,f}(n/b, r + 2, q) \}
\]
Main Result

Non-binary SVT code

\[ q_{\text{SVT}}_{d,e,f}(n, r, q) := \{ \mathbf{x} \in [q]^n \mid \sum_{i=1}^{n} x_i \equiv f \pmod{q}, \quad \sigma(\mathbf{x}) \in \text{SVT}_{d,e}(n - 1, r) \} \]

[Theorem 2] Correction capability of NB-SVT code

\[ q_{\text{SVT}}_{d,e,f}(n, r, q) \text{ is } r\text{-bounded single deletion correcting code} \]

Non-binary \( b \)-burst I/D code [This work]

\[ C_{q,b} = \{ \mathbf{x} \in [q]^n \mid A_b(\mathbf{x})_1 \in q_{\text{VT}}_{a,b}(n/b, q) \cap S_q(n/b, r), \]
\[ A_b(\mathbf{x})_2 \in q_{\text{SVT}}_{d,e,f}(n/b, r + 1, q), \]
\[ \vdots \]
\[ A_b(\mathbf{x})_b \in q_{\text{SVT}}_{d,e,f}(n/b, r + 1, q) \} \]

(Proof of Theorem 2) Straightforward but long... See arXiv:1804.04824
Number of Codewords (1: Lower bound 1)

\[ C_{q,b} = \{ \boldsymbol{x} \in [q]^n \mid A_b(\boldsymbol{x})_1 \in q\text{VT}_{a,b}(n/b, q) \cap S_q(n/b, r), \]
\[ A_b(\boldsymbol{x})_i \in q\text{SVT}_{d,e,f}(n/b, r + 1, q) \text{ for } i \in [2, b] \} \]

[Lemma 5] Lower bound of \# of run length limited sequences

\[ |S_{n,q}(r)| \geq (q^r - n)q^{n-r}. \]

[Lemma 6] Lower bound of \(|A_b(\boldsymbol{x})_1|\)

\[ \max_{a \in [n], c \in [q]} |q\text{VT}_{a,c}(n/b, q) \cap S_q(n/b, r)| \geq \frac{(q^r - n)q^{n-r}}{nq} = \frac{(q^r - n)q^{n-r-1}}{n}. \]

[Lemma 7] Lower bound of non-binary SVT code

\[ \max_{d \in [r+1], e \in [2], f \in [q]} |q\text{SVT}_{d,e,f}(n, r + 1, q)| \geq \frac{q^n}{2(r + 1)q} = \frac{q^{n-1}}{2(r + 1)}. \]

(Lemma 5) From Union bound \hspace{1cm} (Lemma 6,7) Pigeonhole principle
Number of Codewords (2: Lower bound 2)

\[ C_{q,b} = \{ \mathbf{x} \in [q]^n \mid A_b(\mathbf{x})_1 \in q\VT_{a,b}(n/b, q) \cap S_2(n/b, r), A_b(\mathbf{x})_i \in q\VT_{d,e,f}(n/b, r + 1, q) \text{ for } i \in [2, b] \} \]

[Theorem 4] Lower bound of \(|C_{q,b}|\)

For all \(r\), the cardinality of \(C_{q,b}\) satisfies

\[
\max |C_{q,b}| \geq \frac{q^{n-b}}{n} \cdot \frac{b - n q^{-r}}{2^{b-1}(r + 1)^{b-1}}.
\]

Substituting \(r = \log_q n\), we get

\[
\max |C_{q,b}| \geq \frac{2q^{n-b}}{n} \cdot \frac{b - 1}{2^b(\log_q n + 1)^{b-1}}.
\]

Its redundancy (the number of extra symbols: \(n - \log_q |C|\)) is

\[ b + \log_q n - \log_q (b - 1) + (b - 1) \log_q 2 + (b - 1) \log_q (\log_q n + 1). \]
Number of Codewords (3: Upper bound of the best code)

\( M_{q,b}(n) \): \( q \)-ary \( b \)-burst I/D code with largest cardinality

**[Theorem 5]**

For enough large \( n \), the following holds:

\[
|M_{q,b}(n)| \leq \frac{q^{n-b+1}}{(q-1)n}
\]

Upper bound of redundancy for \( M_{q,b}(n) \) is

\[
b - \log_q 2 + \log_q n
\]

Gap of redundancy between \( C_{q,b} \) and \( M_{q,b}(n) \) is

\[
1 - \log_q (q-1) - \log_q (b-1) + (b-1) \log_q 2 + (b-1) \log_q [\log_q n + 1]
\]

\[= O(\log_q \log_q n)\]

Fraction of gap of redundancy \( O\left(\frac{\log_q \log_q n}{n}\right)\)
The number of codewords decreases for small $r$

$\Rightarrow$ For $r = \log_q n$, $|q\text{SVT}_{d,e,f}(n, r, q)| > |q\text{SVT}_{d,e,f}(n, r + 1, q)|$

$\Rightarrow$ The proposed code has larger cardinality than [Schoney2017-11]
Conclusion

- We construct non-binary $b$-burst I/D code
  - We refine the correction capability of non-binary SVT code
- We devise its decoding algorithm
- We evaluate the cardinality of the code